VLSI DESIGN

About the Author

Partha Pratim Sahu received his MTech degree from the Indian Institute of Technology Delhi and is a PhD degree holder in engineering from Jadavpur University, Kolkata. In 1991, he joined Haryana State Electronics Development Corporation Limited, where he was engaged in R&D works related to optical fiber components and telecommunication instruments. In 1996, he joined North Eastern Regional Institute of Science and Technology as a faculty member. At present, he is working as Professor in the Department of Electronics and Communication Engineering, Tezpur Central University, Assam, India. His field of interest includes integrated electronic circuits and optic circuits, wireless and optical communication networks, optical sensor, Oscan electronics and neuro-engineering. He has published more than 42 papers in peer review international journals and presented 32 papers in international conferences. He is a Fellow of the Optical Society of India, a Life Member of Indian Society for Technical Education, and a member of Optical Society of America and the IEEE Communication Society.

VLSI DESIGN

Partha Pratim Sahu

Professor
Department of Electronics and Communication Engineering
Tezpur University
Tezpur, Assam

McGraw Hill Education (India) Private Limited
NEW DELHI

McGraw Hill Education Offices

New Delhi New York St Louis San Francisco Auckland Bogotá Caracas
Kuala Lumpur Lisbon London Madrid Mexico City Milan Montreal
San Juan Santiago Singapore Sydney Tokyo Toronto

McGraw Hill Education (India) Private Limited

Published by McGraw Hill Education (India) Private Limited
P-24, Green Park Extension, New Delhi 110 016

VLSI Design

Copyright © 2013 by McGraw Hill Education (India) Private Limited.

No part of this publication may be reproduced or distributed in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise or stored in a database or retrieval system without the prior written permission of the publishers. The program listings (if any) may be entered, stored and executed in a computer system, but they may not be reproduced for publication.

This edition can be exported from India only by the publishers,
McGraw Hill Education (India) Private Limited.

ISBN (13 digit): 978-1-25-902984-4
ISBN (10 digit): 1-25-902984-0

Vice President and Managing Director: *Ajay Shukla*

Head—Higher Education (Publishing and Marketing): *Vibha Mahajan*

Publishing Manager (SEM & Tech. Ed.): *Shalini Jha*
Editorial Executive: *Koyel Ghosh*
Manager—Production Systems: *Satinder S Baveja*
Assistant Manager—Editorial Services: *Sohini Mukherjee*
Senior Production Executive: *Suhaib Ali*

Assistant General Manager (Marketing)—Higher Education: *Vijay Sarathi*
Senior Product Specialist: *Tina Jajoriya*
Senior Graphic Designer—Cover: *Meenu Raghav*

General Manager—Production: *Rajender P Ghansela*
Manager—Production: *Reji Kumar*

Information contained in this work has been obtained by McGraw Hill Education (India), from sources believed to be reliable. However, neither McGraw Hill Education (India) nor its authors guarantee the accuracy or completeness of any information published herein, and neither McGraw Hill Education (India) nor its authors shall be responsible for any errors, omissions, or damages arising out of use of this information. This work is published with the understanding that McGraw Hill Education (India) and its authors are supplying information but are not attempting to render engineering or other professional services. If such services are required, the assistance of an appropriate professional should be sought.

Typeset at Tej Composers, WZ 391, Madipur, New Delhi 110 063 and printed at
A.P. Offset Pvt. Ltd., 25/487, Zulfe Bengal, Dilshad Garden, New Delhi-110095

Cover Printer: A.P Offset Pvt. Ltd.

RAXCRRAODDLDY

Contents

1. Introduction of MOS Technology to Integrated Circuit **1**

1.1 Evolution of the Integrated Circuit *1*
1.2 Introduction of MOS Technology *3*
1.3 Basic IC Design Flow Chart *3*
1.4 Basic MOS Transistor *5*
References 12
Exercises 12

2. MOSFET and CMOS: Basic Electrical Properties and Circuit Design **13**

2.1 Drain-to-Source Current I_{ds} vs V_{ds} Characteristics of *n*MOS *13*
2.2 Second-Order Effects *20*
2.3 Drain to Source Current I_{ds} vs V_{ds} of *p*MOS *23*
2.4 The *p*MOS Transistor's Threshold Voltage, V_{THP} *23*
2.5 Scaling of MOS Circuits *24*
2.6 Design Process of MOSFET-Based Devices *30*
2.7 Design Rules for Layout *36*
2.8 Translation of Stick Diagram to Lambda-Based Layout *44*
2.9 Translation of Symbolic Diagram into Lambda-Based Layout *44*
2.10 Layout of Resistance and Capacitance *46*
2.11 More Examples of Mask Layout *47*
References 47
Exercises 48

3. CMOS-Based Digital Design **52**

3.1 Digital MOSFET Model *52*
3.2 CMOS Inverter *54*
3.3 CMOS NAND Gate *59*
3.4 CMOS NOR Gate *62*
3.5 Other Logic Gates Using NAND Gate *64*
3.6 Combinational Digital Circuit *66*
3.7 Sequential Digital Circuit *76*
3.8 CMOS Transmission Gate *83*
3.9 Dynamic Logic Gates *85*
3.10 Memory Circuits *88*
3.11 Special Digital Circuits *92*

3.12 CMOS Digital System Design by Using FSM *96*
3.13 Bit Shifter Circuit *100*
3.14 Combinational PLDs *102*
References *108*
Exercises *109*

4. CMOS-Based Analog Circuit **114**

4.1 Passive Components *114*
4.2 Analog MOSFET Models *115*
4.3 Current Source/Sink *117*
4.4 Voltage Dividers *120*
4.5 MOS Amplifiers *121*
4.6 Operational Amplifier *140*
References *152*
Exercises *153*

5. CMOS Mixed Signal Circuit **164**

5.1 Adaptive Biasing *164*
5.2 CMOS Comparator Design *166*
5.3 Analog Multipliers *169*
5.4 Level Shifting *171*
5.5 Dynamic Mixed Signal Circuit *171*
5.6 Data Converter Circuits *179*
5.7 Bit Synchronization/Data Recovery Circuit *195*
5.8 Spread Spectrum Signaling *199*
References *209*
Exercises *211*

6. BiCMOS Circuit **216**

6.1 Modeling of *npn* BJT *216*
6.2 The BiCMOS Inverter *217*
6.3 BiCMOS NAND Gate *219*
6.4 BiCMOS NOR Gate *220*
6.5 CMOS and ECL Conversions using BiCMOS *221*
References *222*
Exercises *223*

7. Design of Testability **224**

7.1 Fault Models *225*
7.2 Test Generation (Stuck-at Faults) *228*
7.3 Path Sensitization *231*
7.4 *D*-algorithm *231*

7.5 Test Generation for other Fault Models *235*
7.6 Test Generation Example *236*
7.7 Sequential Circuit Testing *239*
7.8 Design-for-Testability *240*
7.9 Built-in Self-Test *241*
7.10 Enhancing Testability *246*
References *249*
Exercises *250*

8. Physical Design of VLSI Circuits **251**

8.1 Layout Methodologies *251*
8.2 Partitioning *253*
8.3 Floor Plans *260*
8.4 Placement *269*
8.5 Routing *275*
8.6 Performance in Circuit Layout *290*
References *296*
Exercises *300*

9. Designing of Digital Circuits Using VHDL Programs **307**

9.1 Digital Design Flow by using VHDL Codes *307*
9.2 VHDL Languages or Codes *319*
9.3 Representation of Combinational Logic using VHDL Codes *332*
9.4 Representation of Synchronous Logic using VHDL Codes *340*
9.5 Representation of Three-State Buffers and Bidirectional Signals *344*
9.6 Designing FIFO using VHDL Code *346*
9.7 Hierarchy in a Large Design *354*
9.8 Functions and Procedures *358*
9.9 Pipelining *361*
References *365*
Exercises *366*

10. Top-level System Design: CPU **370**

10.1 CPU : 16-Bit Microprocessor *370*
10.2 Instructions *371*
10.3 Block-Copy Operation *378*
10.4 ALU *382*
10.5 Comparator *384*
10.6 Control *386*
10.7 Reg *394*
10.8 Regarray *395*
10.9 Shift *397*

10.10 Trireg *398*
10.11 Verification of RTL Description *400*
References 413
Exercises 413

11. VLSI Process Technology **415**

11.1 Silicon-Wafer Preparation *415*
11.2 Wafer Etching, Polishing, and Cleaning *419*
11.3 Thermal Oxidation and Oxidation System *421*
11.4 Diffusion *423*
11.5 Implantation Systems *431*
11.6 Chemical Vapour Deposition *437*
11.7 Flame Hydrolysis Deposition (FHD) *442*
11.8 Epitaxy *443*
11.9 Lithography *447*
11.10 Metallization *454*
11.11 Etching *458*
11.12 Assembly and Packaging *464*
11.13 Fabrication of a Typical Circuit *467*
References 470
Exercises 470
Index *473*

Preface

Overview

In the last few years, electronic-chip revenues have increased by over 40 percent, and this growth has become exponential as per a recent report by the Semiconductor Industry Association (SIA). Nowadays, VLSI design and related technology have become indispensable to accommodate the skyrocketing increase in chip-circuit complexity, and integration scale required in present day's high-speed communication instrumentation and other electronics processing systems. In these systems, use of either digital circuits or analog circuits, separately or in combination (called mixed-signal circuits), is an essential requirement. Developing complicated circuits in integrated form essentially requires proper design and analysis in the range of Very large Scale Integration (VLSI). In this direction, the major portion of total worldwide chip sales is dominated by the MOS market. Because of the worldwide demand of VLSI chips, huge workforce is required in the industry as well as R&D labs for design of these chips along with fabrication process technology. Keeping these in mind, VLSI design and technology has become a compulsory course for both undergraduate and postgraduate electronics engineering and science programmes in technical universities, NITs and IITs in India, and also foreign universities/institutes.

Salient Features

- Span of coverage of VLSI design fundamentals as per course requirements
- Inclusion of applications and latest developments in the subject:
    - Nanometer CMOS design issues
    - Submicron Technology
- Excellent use of VHDL programs for digital design and top-level system design
- Focus on design aspects for power consumption optimization
- Chapter on "VLSI Process Technology"
- Rich pedagogy:
    - Over 400 Diagrams
    - 50 Solved Examples
    - Over 220 Exercises

Chapter Organization

The book covers all the aspects of design and analysis of VLSI circuits starting from preliminary designs to layout design as well as an introduction to processing technology. The following is a brief highlight of the main topics covered in each chapter.

In **Chapter 1**, the book starts with an overview of micro-electronics and introduction to MOS technology which describes the basics of MOSFET, CMOS and BiCMOS. In this chapter, basic IC design flowchart is also mentioned.

Chapter 2 describes the basic electrical properties of MOSFET devices with body effects and second-order effects. The scaling of MOSFET circuits is also discussed in this chapter. The design rules of MOSFET circuits along with stick diagrams and layouts are also mentioned.

Since CMOS circuits have very high packing density, preferred for VLSI, so, **Chapter 3** starts with the digital MOSFET model. Different basic digital circuit modules based on CMOS devices are also discussed. It also includes memory-based CMOS devices and special digital circuits.

Chapter 4 describes the basic analog circuit components such as resistances, capacitances, sources and sinks, and amplifiers. It also mentions op-amp structures based on CMOS devices and related circuits.

Nowadays, circuits dealing with both analog and digital signals have been focused on applications such as high-speed wireless communication and instrumentation circuits. The designing of these circuits are called mixed-signal circuits. **Chapter 5** discusses mixed-signal circuits that include voltage comparators, adaptive biasing, ADC, DAC analog multiplexer, etc.

Chapter 6 describes BiCMOS based NAND, NOR and NOT gates. It also includes ECL conversion using BiCMOS.

Testability of chips is an important part of VLSI circuits. **Chapter 7** starts with different fault models of chips. It also discusses test generation of these fault models with examples.

As reduction of both area and connection-wire lengths are essential before making the layout of VLSI chips, **Chapter 8** describes physical design of VLSI.

Chapter 9 covers VHDL approach for the design of circuits. Since one can implement digital circuits in FPGA platforms, this chapter includes different FPGA architectures with routing.

A simple example of a top-level system design is the central processing unit (CPU). **Chapter 10** covers CPU design, starting from VHDL representation to verification of its functionality synthesis with VHDL programming.

Finally, **Chapter 11** mentions different steps such as silicon-wafer preparation, wafer cleaning, oxidation of silicon, diffusion, ion implantation, epitaxial growth, lithography, metallization, etching, etc. used in VLSI IC processing.

The book may be used as a textbook covering the syllabi of basic VLSI design, physical design of VLSI, VLSI technology and VHDL courses in both undergraduate and postgraduate levels.

Online Learning Center

The text is supported by an Online Learning Center, available at *https://www.mhhe.com/sahu/vlsid* This contains links for extra reading **for students**; and the Solution Manual and PowerPoint slides **for instructors**.

Acknowledgements

It is my great pleasure to acknowledge the help of many individuals in the writing of this book. I have been teaching VLSI design for over 16 years and doing research on circuit design (especially communication circuits) in VLSI for 12 years. The writing of this book is the result of the above and has itself taken over five years. During this period, I have had close interaction with many senior students working in reputed universities/institutes and industries/R&D organizations. I am deeply indebted to them for many enlightening discussions that have enriched my understanding of the subject. The many stimulating discussions with my colleagues in Tezpur University and especially my friends Prof. M K Naskar of Jadavpur University and Prof. Utpal Biswas of Kalyani University are gratefully acknowledged.

Special thanks go to my PhD and PG students, especially Mr Bijoy Chatterjee, Mr Bidyut Deka and Mr Mahipal Singh, for their help and support in the preparation of this book. I also remain grateful to Prof. M K Chaudhuri, Vice Chancellor, Tezpur University, for his encouragement and support.

The reviewers are greatly appreciated for their valuable suggestions and comments, which led to the improvement and modification of the book. Their names are given below:

Amit Naik	*Shri Govindam Seksaria Institute of Technology and Science (SGSITS), Indore, Madhya Pradesh*
Kamal Prakash Pandey	*Shambhunath Institute of Engineering and Technology, Allahabad, Uttar Pradesh*
Neelesh Srivastava	*Krishna Institute of Engineering and Technology (KIET), Ghaziabad, Uttar Pradesh*
Manoj Kumar	*BSA College of Engineering and Technology, Mathura, Uttar Pradesh*
Mrinal Kanti Naskar	*Jadavpur University, Kolkata, West Bengal*
Utpal Biswas	*University of Kalyani, Nadia, West Bengal*
Pinaki R. Ghosh	*Adamas Institute of Technology, Barabaria, West Bengal*
Soumen Khatua	*Sir J C Bose School of Engineering, Kolkata, West Bengal*
Debarshi Datta	*Brainware Group of Institutions, Kolkata, West Bengal*
Harpal Thetti	*KIIT University* (*Kalinga Institute of Industrial Technology*), *Bhubaneswar, Odisha*
Jitendra Patel	*C K Pithawalla College of Engineering and Technology, (CKPCET), Surat, Gujarat*
Malhar Chauhan	*Narnarayan Shastri Institute of Technology, Ahmedabad, Gujarat*
Dhiren Mehta	*Veermata Jijabai Technological Institute, Mumbai, Maharashtra*
Anil Suthar	*Laxminarayan Institute of Technology (LCIT), Nagpur, Maharashtra*
Vijay Chavda	*Government Engineering College (GEC), Modasa, Gujarat*
S M Joshi	*JSPM's Bhivrabai Sawant Institute of Technology and Research, Pune, Maharashtra*

Nilesh Kalani
RK University
Rajkot, Gujarat

L S Biradar
Poojya Dodappa Appa (PDA) College of Engineering
Gulbarga, Karnataka

G Dhanabalan
Kamaraj College of Engineering and Technology
Virudhunagar, Tamil Nadu

M Madhavi Lathi
Jawaharlal Nehru Technological University (JNTU) College of Engineering
Kukatpally, Hyderabad

V S Kanchana Bhaaskaran
Vellore Institute of Technology (VIT) University
Chennai, Tamil Nadu

P V Sree Devi
Andhra University
Hyderabad, Andhra Pradesh

A Ananthi
Thiagrajar College of Engineering
Madurai, Tamil Nadu

N Balaji
Vignana Jyothi Institute of Engineering and Technology
Hyderabad, Andhra Pradesh

R Renugadevi
Kalasalingam University
Virudhunagar, Tamil Nadu

Last but not the least, I am greatly indebted to my parents for their constant support and encouragement to complete this project. The writing of this book used many of my holidays and vacations I normally would have spent with my family, and it is difficult to acknowledge their sacrifice. My special gratitude goes to my wife, Arpita, and my daughters, Prakriti (Mum) and Ritushree (Bubun).

I am thankful to the entire publishing team of McGraw Hill Education, India, particularly Ms Koyel Ghosh for initiating this project and Ms Sohini Mukherjee for her continuous interaction in editing the content of this book. The input from the marketing team has also been very useful. I also thank the editorial team of McGraw Hill Education, India, for committing to the timely revision of the text.

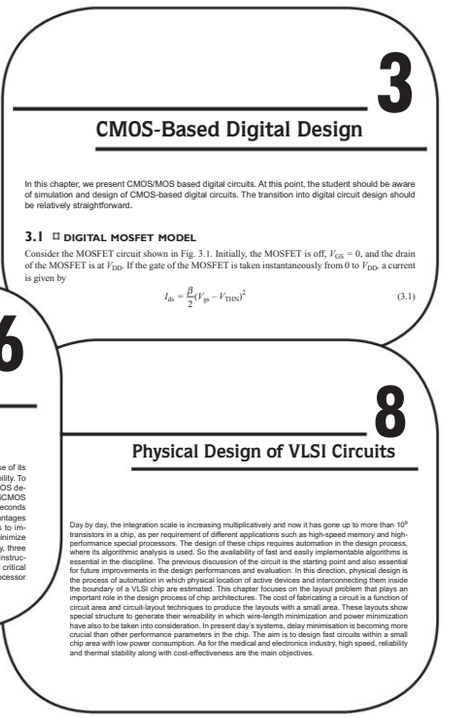
Partha Pratim Sahu

Publisher's Note

Do you have any further request or a suggestion? We are always open to new ideas (the best ones come from you!). You may send your comments to *tmh.elcefeedback@gmail.com*. Piracy-related issues may also be reported!

GUIDED TOUR

Inclusion of applications and latest developments in the subject



3

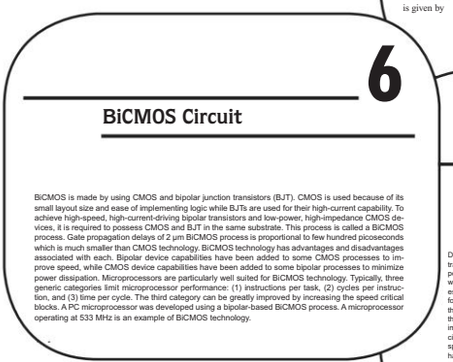
CMOS-Based Digital Design

In this chapter, we present CMOS/MOS based digital circuits. At this point, the student should be aware of simulation and design of CMOS-based digital circuits. The transition into digital circuit design should be relatively straightforward.

3.1 ☐ DIGITAL MOSFET MODEL

Consider the MOSFET circuit shown in Fig. 3.1. Initially, the MOSFET is off, $V_{GS} = 0$, and the drain of the MOSFET is at V_{DD}. If the gate of the MOSFET is taken instantaneously from 0 to V_{DD}, a current is given by

$$I_{ds} = \frac{\beta}{2}(V_{gs} - V_{TH})^2 \qquad (3.1)$$



6

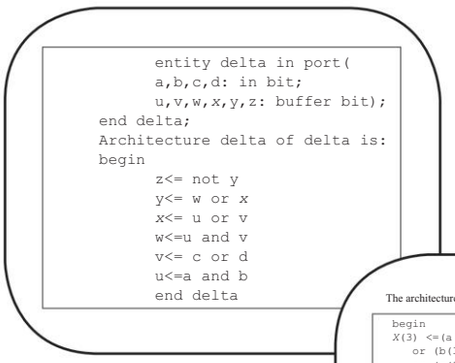
BiCMOS Circuit

BiCMOS is made by using CMOS and bipolar junction transistors (BJT). CMOS is used because of its small layout size and ease of implementing logic while BJTs are used for their high-current capability. To achieve high-speed, high-current-driving bipolar transistors and low-power, high-impedance CMOS devices, it is required to possess CMOS and BJT in the same substrate. This process is called a BiCMOS process. Gate propagation delays of 2 μm BiCMOS process is proportional to few hundred picoseconds which is much smaller than CMOS technology. BiCMOS technology has advantages and disadvantages associated with each. Bipolar device capabilities have been added to some CMOS processes to improve speed, while CMOS device capabilities have been added to some bipolar processes to minimize power dissipation. Microprocessors are particularly well suited for BiCMOS technology. Typically, three generic categories limit microprocessor performance: (1) instructions per task, (2) cycles per instruction, and (3) time per cycle. The third category can be greatly improved by increasing the speed critical blocks. A PC microprocessor was developed using a bipolar-based BiCMOS process. A microprocessor operating at 533 MHz is an example of BiCMOS technology.

8

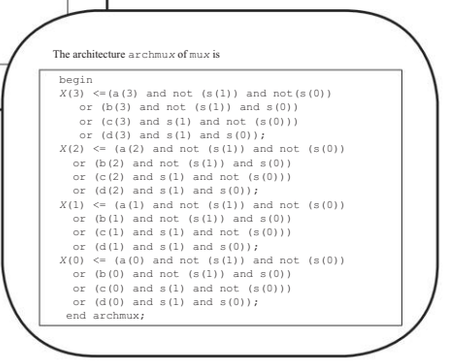
Physical Design of VLSI Circuits

Day by day, the integration scale is increasing multiplicatively and now it has gone up to more than 10^6 transistors in a chip, as per requirement of different applications such as high-speed memory and high-performance special processors. The design of these chips requires automation in the design process, where its algorithmic analysis is used. So the availability of fast and easily implementable algorithms is essential in the discipline. The previous discussion of the circuit is the starting point and also essential for future improvements in the design performances and evaluation. In this direction, physical design is the process of automation in which physical location of active devices and interconnecting them inside the boundary of a VLSI chip are estimated. This chapter focuses on the layout problem that plays an important role in the design process of chip architectures. The cost of fabricating a circuit is a function of circuit area and circuit-layout techniques to produce the layouts with a small area. These layouts show special structure to generate their wireability in which wire-length minimization and power minimization have also to be taken into consideration. In present day's systems, delay minimisation is becoming more crucial than other performance parameters in the chip. The aim is to design fast circuits within a small chip area with low power consumption. As for the medical and electronics industry, high speed, reliability and thermal stability along with cost-effectiveness are the main objectives.



```
        entity delta in port(
        a,b,c,d: in bit;
        u,v,w,x,y,z: buffer bit);
end delta;
Architecture delta of delta is:
begin
        z<= not y
        y<= w or x
        x<= u or v
        w<=u and v
        v<= c or d
        u<=a and b
        end delta
```

Excellent use of VHDL programs for digital design and top-level system design



The architecture `archmux` of `mux` is

```
begin
X(3) <=(a(3) and not (s(1)) and not(s(0))
   or (b(3) and not (s(1)) and s(0))
   or (c(3) and s(1) and not (s(0)))
   or (d(3) and s(1) and s(0));
X(2) <= (a(2) and not (s(1)) and not (s(0))
  or (b(2) and not (s(1)) and s(0))
  or (c(2) and s(1) and not (s(0)))
  or (d(2) and s(1) and s(0));
X(1) <= (a(1) and not (s(1)) and not (s(0))
  or (b(1) and not (s(1)) and s(0))
  or (c(1) and s(1) and not (s(0)))
  or (d(1) and s(1) and s(0));
X(0) <= (a(0) and not (s(1)) and not (s(0))
  or (b(0) and not (s(1)) and s(0))
  or (c(0) and s(1) and not (s(0)))
  or (d(0) and s(1) and s(0));
end archmux;
```

CONTD.

Focus on design aspects for optimization of power consumption

8.6 □ PERFORMANCE IN CIRCUIT LAYOUT

Recent advances in VLSI technology allow the fabrication of more than 10^6 transistors in a single chip. Before making layout of a VLSI chip, it is required to test layout performances. In this direction, we consider interconnection delay problem via minimization problem, power minimization, and training-driven routing problem which are required to be solved.

8.6.1 Delay Models

Delay models of ICs are based on timing-analysis methods which are analog simulation and timing simulation for digital circuits. Analog simulation is based on IV characteristics of the devices. Worse-case delay analog need finding of worse-case input patterns. The analysis is done by using numerical integration for analog circuits. For most cases, timing analysis is made for digital circuits in which gate delay and interconnection models are used.

1. Gate-delay Model

The simplest gate-delay model is based on propagation delay or gate delay T_D from a gate input to a gate output. Figure 8.30 shows gate-delay model consisting of input gate loading or gate capacitance, C_g, output gate loading capacitance, C_g, and the gate output on resistance, R_{on}.

The gate output begins changing at twice $t + T_D$. If the driving load capacitance is C_L then the rise time of the output is $\tau = R_{on}\,(C_d + C_L)$. The on resistance R_{on} is a function of transistor length L and width w and is given by

$$R_{on} \approx \frac{1}{\mu C_{gox}(V_{DD} - V_T)} \cdot \frac{L}{W}$$

Chapter on VLSI Process Technology

11

VLSI Process Technology

Silicon VLSI IC processes include basic processes involved such as silicon wafer preparation, wafer cleaning, oxidation of silicon, diffusion, ion implantation, epitaxial growth, lithography, metallization, and etching. In this chapter, we have discussed their basic steps.

11.1 □ SILICON-WAFER PREPARATION

Silicon is abundant in nature in the form of silica and silicates (i.e., sand). Normally, in integrated circuits, silicon should be a single crystal of very high purity in wafer form obtained from quartzite. It can be refined by using simple purification and crystal-growth techniques. Silicon also exhibits desirable physical properties and good electrical characteristic when used to fabricate active and passive devices on the same substrate. Initially, the starting material silicon is chemically treated to form a high-purity polycrystalline semiconductor from which single crystals are grown. The single crystals are shaped to define the diameter of the material and then sawed into wafers. These wafers are polished to provide smooth and clean surfaces on which devices will be made and then be interconnected to form a monolithic IC.

CONTD.

Over 400 diagrams and illustrations are present to clarify the concepts

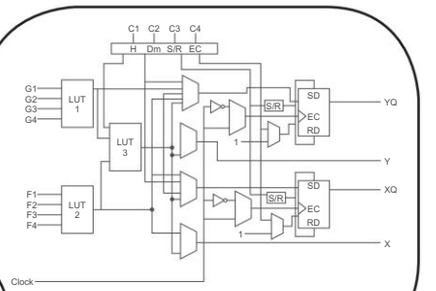


Fig. 9.7 XC-4000 configurable logic block

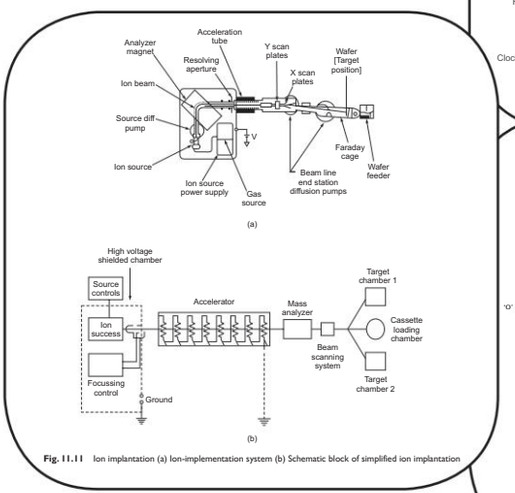


Fig. 11.11 Ion implantation (a) Ion-implementation system (b) Schematic block of simplified ion implantation

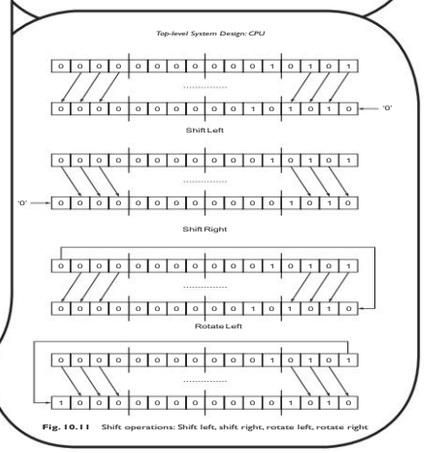


Fig. 10.11 Shift operations: Shift left, shift right, rotate left, rotate right

50 Solved Examples spread across the text to help students solve numerical problems related to the topics

Example 2.1 ◻ *For a substrate doping of 10^{15} atoms/cm^3, $V_{gs} = V_{THN}$ and V_{sb} = Substrate-to-source bias voltage = 0, estimate the electrostatic potential in the substrate region and at the oxide-semiconductor interface.*

Solution: The electrostatic potential of the substrate is given by

$$\frac{kT}{q}\ln\frac{N_A}{n_i} = -26\text{ mV}\ln\frac{10^5}{14.5\times10^9}$$

where n_i = Intrinsic carrier concentration at room temperature (25^0 C) = 14.5×10^9 atom/cm^3

Example 2.2 ◻ Consider the *n*-channel MOS process in Example 2.1. One may examine how a non-zero source-to-substrate voltage V_{SB} influences the threshold voltage of an *n*MOS transistor.

One can calculate the substrate-bias coefficient γ using the parameters provided in Example 2.1 as follows:

$$\nu = \frac{\sqrt{2qN_A\varepsilon_S}}{C_{ox}} = \frac{\sqrt{2\times1.6\times10^{-19}\times10^6\times11.7\times8.85\times10^{-14}}}{7.03\times10^{-18}} = 0.82V^{\frac{1}{2}}$$

One is now in a position to determine the variation of threshold voltage V_T as a function of the source-to-substrate voltage V_{SB}. Assume the voltage V_{SB} to range from 0 to 5 V.

$$V_T = V_{T0} + \gamma\left(\sqrt{|2\phi_F + V_{SB}|} - \sqrt{|2\phi_F|}\right) = 0.40 + 0.82\left(\sqrt{0.7 + V_{SB}} - \sqrt{0.7}\right)$$

CONTD.

Over 220 Exercise problems to help students have practice in solving numerical problems

□ EXERCISES

7.1 Find the pseudo-random sequences in 4-bit LFSRs defined by the following polynomials: (a) $x^4+x^3+x^2+1$ (b) x^4+x^2+x (c) x^4+x^3+1 (d) $x^4+x^3+x^2$

7.2 Verify the 5-value logic operation for *D*-algorithm given in Fig. 7.5.

7.3 Develop a test set that detects all single stuck-at faults in Fig. P7.1.

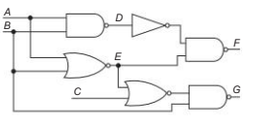


Fig. P7.1

7.4 Find the singular cover for a logic function $Z = \overline{a.b + c}$

7.5 Find the propagation *D*-cube for the logic function $Z = \overline{a.b + c}$

7.6 Find the primitive *D*-cube for $Z = \overline{a.b + c}$ when *Z*: *s*-*a*-1

7.7 Show that $\frac{d(F \oplus G)}{dx} = \frac{df}{dx} \oplus \frac{dG}{dx}$

8.31 Solve an instance of the channel-routing problem employing the Greedy algorithm. Use the same input formats as in the previous exercise.

8.32 Consider a set of modules in an FPGA environment. Find placement and routing. The main objective is to find a routing and to minimize the total length of the nets.
Input format:
16 (* number of modules *)
4 (* number of rows and columns of cells *)
N1 2 3, (* net 1 interconnects modules 2 and 3 *)
N2 1 3 5,...
As shown in Fig. P8.6, the number of tracks in each channel is always 5 and the width of each cell is always 8.

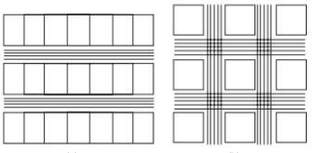


Fig. P8.6

References in each chapter for additional reading

□ REFERENCES

8.1. Agarwal, P.K. and M.T. Shing, Algorithm for Special Cases of Rectilinear Steiner Trees: I. Points on the Boundary of a Rectilinear Rectangle, Networks 20(4):453–485, 1990.

8.2. Aho, A.V., M.R. Garey, and F.K. Hwang, 'Rectilinear Steiner Trees: Efficient Special-Case Algorithm, Networks 7:35–58, 1977.

8.3. Aho, A.V., J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, MA, 1974.

8.4. Akama, T., H. Suzuki, and T. Nishizeki, *Finding Steiner Forests in Planar Graphs,* in The First Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 444–453, ACM, 1990.

8.5. Akers, S.B., M.E. Geyer, and D.L. Roberts, *IC Mask Layout with a Single Conductor Layer*, in Design Automation Conference, pp. 7–16, IEEE/ACM, 1970.

8.6. Antreich, K.J., F.M. Johannes, and F.H. Kirsch, *A New Approach for Solving the Placement Problem Using Force Models*, in International Symposium on Circuits and Systems, pp. 481–486, IEEE, 1982.

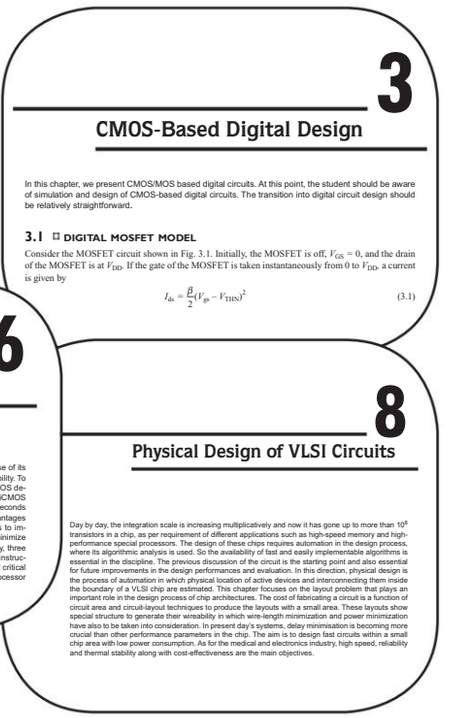
8.7. Anway, H., G. Farnham, and R. Reid, *Plint Layout System for VLSI Chip Design*, in Design Automation Conference, pp. 449–452, IEEE/ACM, 1985.

8.8. Asano, T., and H. Imai, *Partioning a Polygon Region into Trapezoids*, Association for Computing Machinery 33(2):290–312, 1986.

8.9. Baker, B.S., S.N. Bhatt, and F.T. Leighton, *An Approximation Algorithm for Manhattan Routing*, in Proc. 15th Annual Symp. Theory of Computing, pp. 477–486, ACM; 1983.

GUIDED TOUR

Inclusion of applications and latest developments in the subject



3

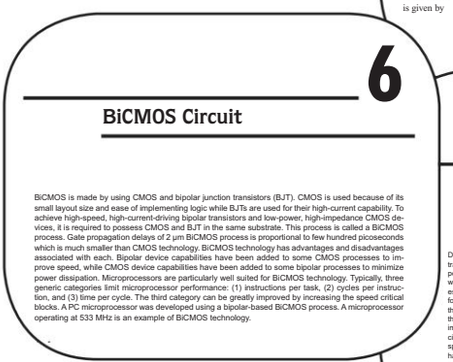
CMOS-Based Digital Design

In this chapter, we present CMOS/MOS based digital circuits. At this point, the student should be aware of simulation and design of CMOS-based digital circuits. The transition into digital circuit design should be relatively straightforward.

3.1 ☐ DIGITAL MOSFET MODEL

Consider the MOSFET circuit shown in Fig. 3.1. Initially, the MOSFET is off, $V_{GS} = 0$, and the drain of the MOSFET is at V_{DD}. If the gate of the MOSFET is taken instantaneously from 0 to V_{DD}, a current is given by

$$I_{ds} = \frac{\beta}{2}(V_{gs} - V_{TH})^2 \qquad (3.1)$$



6

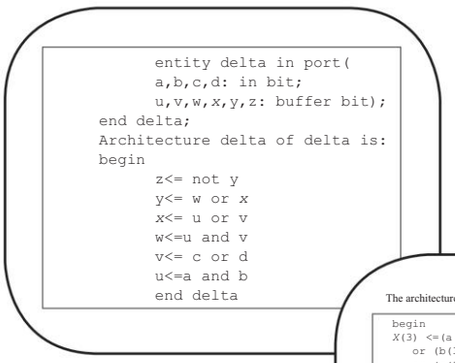
BiCMOS Circuit

BiCMOS is made by using CMOS and bipolar junction transistors (BJT). CMOS is used because of its small layout size and ease of implementing logic while BJTs are used for their high-current capability. To achieve high-speed, high-current-driving bipolar transistors and low-power, high-impedance CMOS devices, it is required to possess CMOS and BJT in the same substrate. This process is called a BiCMOS process. Gate propagation delays of 2 μm BiCMOS process is proportional to few hundred picoseconds which is much smaller than CMOS technology. BiCMOS technology has advantages and disadvantages associated with each. Bipolar device capabilities have been added to some CMOS processes to improve speed, while CMOS device capabilities have been added to some bipolar processes to minimize power dissipation. Microprocessors are particularly well suited for BiCMOS technology. Typically, three generic categories limit microprocessor performance: (1) instructions per task, (2) cycles per instruction, and (3) time per cycle. The third category can be greatly improved by increasing the speed critical blocks. A PC microprocessor was developed using a bipolar-based BiCMOS process. A microprocessor operating at 533 MHz is an example of BiCMOS technology.

8

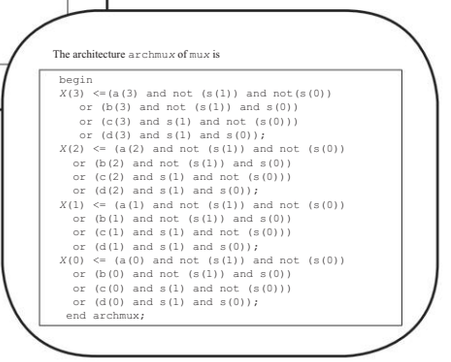
Physical Design of VLSI Circuits

Day by day, the integration scale is increasing multiplicatively and now it has gone up to more than 10^6 transistors in a chip, as per requirement of different applications such as high-speed memory and high-performance special processors. The design of these chips requires automation in the design process, where its algorithmic analysis is used. So the availability of fast and easily implementable algorithms is essential in the discipline. The previous discussion of the circuit is the starting point and also essential for future improvements in the design performances and evaluation. In this direction, physical design is the process of automation in which physical location of active devices and interconnecting them inside the boundary of a VLSI chip are estimated. This chapter focuses on the layout problem that plays an important role in the design process of chip architectures. The cost of fabricating a circuit is a function of circuit area and circuit-layout techniques to produce the layouts with a small area. These layouts show special structure to generate their wireability in which wire-length minimization and power minimization have also to be taken into consideration. In present day's systems, delay minimisation is becoming more crucial than other performance parameters in the chip. The aim is to design fast circuits within a small chip area with low power consumption. As for the medical and electronics industry, high speed, reliability and thermal stability along with cost-effectiveness are the main objectives.



```
        entity delta in port(
        a,b,c,d: in bit;
        u,v,w,x,y,z: buffer bit);
end delta;
Architecture delta of delta is:
begin
        z<= not y
        y<= w or x
        x<= u or v
        w<=u and v
        v<= c or d
        u<=a and b
        end delta
```

Excellent use of VHDL programs for digital design and top-level system design



The architecture `archmux` of `mux` is

```
begin
X(3) <=(a(3) and not (s(1)) and not(s(0))
   or (b(3) and not (s(1)) and s(0))
   or (c(3) and s(1) and not (s(0)))
   or (d(3) and s(1) and s(0));
X(2) <= (a(2) and not (s(1)) and not (s(0))
  or (b(2) and not (s(1)) and s(0))
  or (c(2) and s(1) and not (s(0)))
  or (d(2) and s(1) and s(0));
X(1) <= (a(1) and not (s(1)) and not (s(0))
  or (b(1) and not (s(1)) and s(0))
  or (c(1) and s(1) and not (s(0)))
  or (d(1) and s(1) and s(0));
X(0) <= (a(0) and not (s(1)) and not (s(0))
  or (b(0) and not (s(1)) and s(0))
  or (c(0) and s(1) and not (s(0)))
  or (d(0) and s(1) and s(0));
end archmux;
```

CONTD.

Focus on design aspects for optimization of power consumption

8.6 □ PERFORMANCE IN CIRCUIT LAYOUT

Recent advances in VLSI technology allow the fabrication of more than 10^6 transistors in a single chip. Before making layout of a VLSI chip, it is required to test layout performances. In this direction, we consider interconnection delay problem via minimization problem, power minimization, and training-driven routing problem which are required to be solved.

8.6.1 Delay Models

Delay models of ICs are based on timing-analysis methods which are analog simulation and timing simulation for digital circuits. Analog simulation is based on IV characteristics of the devices. Worse-case delay analog need finding of worse-case input patterns. The analysis is done by using numerical integration for analog circuits. For most cases, timing analysis is made for digital circuits in which gate delay and interconnection models are used.

1. Gate-delay Model

The simplest gate-delay model is based on propagation delay or gate delay T_D from a gate input to a gate output. Figure 8.30 shows gate-delay model consisting of input gate loading or gate capacitance, C_g, output gate loading capacitance, C_g, and the gate output on resistance, R_{on}.

The gate output begins changing at twice $t + T_D$. If the driving load capacitance is C_L then the rise time of the output is $\tau = R_{on}\,(C_d + C_L)$. The on resistance R_{on} is a function of transistor length L and width w and is given by

$$R_{on} \approx \frac{1}{\mu C_{gox}(V_{DD} - V_T)} \cdot \frac{L}{W}$$

Chapter on VLSI Process Technology

11

VLSI Process Technology

Silicon VLSI IC processes include basic processes involved such as silicon wafer preparation, wafer cleaning, oxidation of silicon, diffusion, ion implantation, epitaxial growth, lithography, metallization, and etching. In this chapter, we have discussed their basic steps.

11.1 □ SILICON-WAFER PREPARATION

Silicon is abundant in nature in the form of silica and silicates (i.e., sand). Normally, in integrated circuits, silicon should be a single crystal of very high purity in wafer form obtained from quartzite. It can be refined by using simple purification and crystal-growth techniques. Silicon also exhibits desirable physical properties and good electrical characteristic when used to fabricate active and passive devices on the same substrate. Initially, the starting material silicon is chemically treated to form a high-purity polycrystalline semiconductor from which single crystals are grown. The single crystals are shaped to define the diameter of the material and then sawed into wafers. These wafers are polished to provide smooth and clean surfaces on which devices will be made and then be interconnected to form a monolithic IC.

CONTD.

Over 400 diagrams and illustrations are present to clarify the concepts

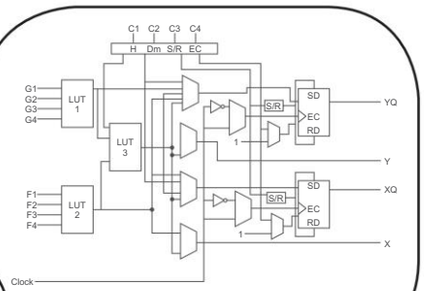


Fig. 9.7 XC-4000 configurable logic block

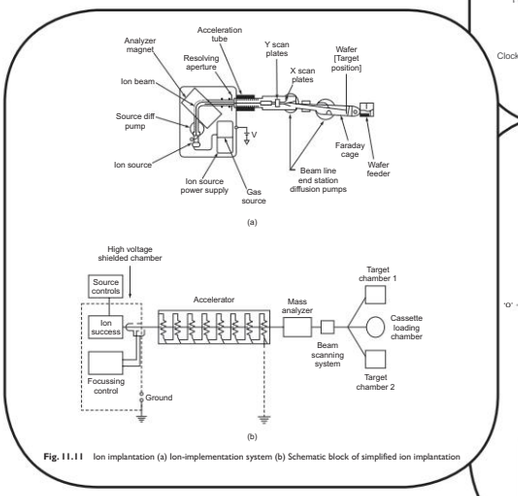


Fig. 11.11 Ion implantation (a) Ion-implementation system (b) Schematic block of simplified ion implantation

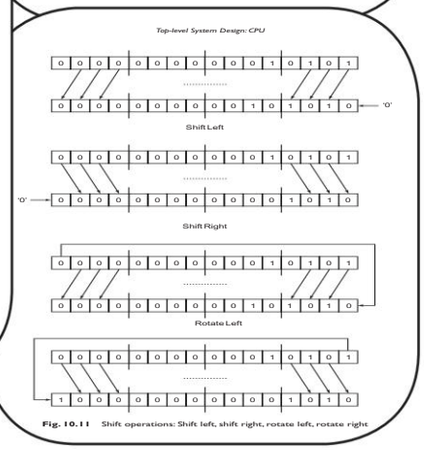


Fig. 10.11 Shift operations: Shift left, shift right, rotate left, rotate right

50 Solved Examples spread across the text to help students solve numerical problems related to the topics

Example 2.1 ◻ *For a substrate doping of 10^{15} atoms/cm^3, $V_{gs} = V_{THN}$ and V_{sb} = Substrate-to-source bias voltage = 0, estimate the electrostatic potential in the substrate region and at the oxide-semiconductor interface.*

Solution: The electrostatic potential of the substrate is given by

$$\frac{kT}{q}\ln\frac{N_A}{n_i} = -26\text{ mV}\ln\frac{10^5}{14.5\times10^9}$$

where n_i = Intrinsic carrier concentration at room temperature (25^0 C) = 14.5×10^9 atom/cm^3

Example 2.2 ◻ Consider the *n*-channel MOS process in Example 2.1. One may examine how a non-zero source-to-substrate voltage V_{SB} influences the threshold voltage of an *n*MOS transistor.

One can calculate the substrate-bias coefficient γ using the parameters provided in Example 2.1 as follows:

$$\nu = \frac{\sqrt{2qN_A\varepsilon_S}}{C_{ox}} = \frac{\sqrt{2\times1.6\times10^{-19}\times10^6\times11.7\times8.85\times10^{-14}}}{7.03\times10^{-18}} = 0.82V^{\frac{1}{2}}$$

One is now in a position to determine the variation of threshold voltage V_T as a function of the source-to-substrate voltage V_{SB}. Assume the voltage V_{SB} to range from 0 to 5 V.

$$V_T = V_{T0} + \gamma\left(\sqrt{|2\phi_F + V_{SB}|} - \sqrt{|2\phi_F|}\right) = 0.40 + 0.82\left(\sqrt{0.7 + V_{SB}} - \sqrt{0.7}\right)$$

CONTD.

Over 220 Exercise problems to help students have practice in solving numerical problems

□ EXERCISES

7.1 Find the pseudo-random sequences in 4-bit LFSRs defined by the following polynomials: (a) $x^4+x^3+x^2+1$ (b) x^4+x^2+x (c) x^4+x^3+1 (d) $x^4+x^3+x^2$

7.2 Verify the 5-value logic operation for *D*-algorithm given in Fig. 7.5.

7.3 Develop a test set that detects all single stuck-at faults in Fig. P7.1.

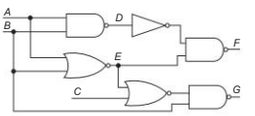


Fig. P7.1

7.4 Find the singular cover for a logic function $Z = \overline{a.b + c}$

7.5 Find the propagation *D*-cube for the logic function $Z = \overline{a.b + c}$

7.6 Find the primitive *D*-cube for $Z = \overline{a.b + c}$ when *Z*: *s*-*a*-1

7.7 Show that $\frac{d(F \oplus G)}{dx} = \frac{df}{dx} \oplus \frac{dG}{dx}$

8.31 Solve an instance of the channel-routing problem employing the Greedy algorithm. Use the same input formats as in the previous exercise.

8.32 Consider a set of modules in an FPGA environment. Find placement and routing. The main objective is to find a routing and to minimize the total length of the nets.
Input format:
16 (* number of modules *)
4 (* number of rows and columns of cells *)
N1 2 3, (* net 1 interconnects modules 2 and 3 *)
N2 1 3 5,...
As shown in Fig. P8.6, the number of tracks in each channel is always 5 and the width of each cell is always 8.

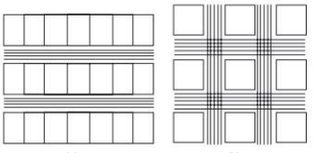


Fig. P8.6

References in each chapter for additional reading

□ REFERENCES

8.1. Agarwal, P.K. and M.T. Shing, Algorithm for Special Cases of Rectilinear Steiner Trees: I. Points on the Boundary of a Rectilinear Rectangle, Networks 20(4):453–485, 1990.

8.2. Aho, A.V., M.R. Garey, and F.K. Hwang, 'Rectilinear Steiner Trees: Efficient Special-Case Algorithm, Networks 7:35–58, 1977.

8.3. Aho, A.V., J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, MA, 1974.

8.4. Akama, T., H. Suzuki, and T. Nishizeki, *Finding Steiner Forests in Planar Graphs,* in The First Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 444–453, ACM, 1990.

8.5. Akers, S.B., M.E. Geyer, and D.L. Roberts, *IC Mask Layout with a Single Conductor Layer*, in Design Automation Conference, pp. 7–16, IEEE/ACM, 1970.

8.6. Antreich, K.J., F.M. Johannes, and F.H. Kirsch, *A New Approach for Solving the Placement Problem Using Force Models*, in International Symposium on Circuits and Systems, pp. 481–486, IEEE, 1982.

8.7. Anway, H., G. Farnham, and R. Reid, *Plint Layout System for VLSI Chip Design*, in Design Automation Conference, pp. 449–452, IEEE/ACM, 1985.

8.8. Asano, T., and H. Imai, *Partioning a Polygon Region into Trapezoids*, Association for Computing Machinery 33(2):290–312, 1986.

8.9. Baker, B.S., S.N. Bhatt, and F.T. Leighton, *An Approximation Algorithm for Manhattan Routing*, in Proc. 15th Annual Symp. Theory of Computing, pp. 477–486, ACM; 1983.

1

Introduction of MOS Technology to Integrated Circuit

Day by day, every moment of our lives is significantly influenced by electronics engineering technology as seen in domestic use, professional activities and workspace, and leisure time. This is due to the development of micro-electronics technology. Micro-electronics devices have advantages such as reliability, low power dissipation, extremely low weight, less volume and low cost coupled with a high degree of accuracy and complexity. Particularly, integrated-circuit-based micro-electronic components have made possible the design of the powerful and flexible processors which provide highly intelligent processing and are suited for customer use. After invention of transistors by W B Schottky, W H Brattain and J Bardeen at Bell Laboratories in 1947, electronic active device technology has started dominating over vacuum technology in the electronics industry and has also generated a powerful platform for emergence of modern automation industry. This was possible after the development of nano-electronics devices. In this direction, the first Integrated Circuit (IC) emerged at the beginning of 1960. Since then, the effort to increase the number of devices per chip has provided rapid development of four generations of ICs—SSI, LSI, VLSI, and ULSI. ULSI is the next generation IC aiming to realize new application domains currently not accessible to MOS technology. The new version of IC technology such as ULSI require smaller, faster, more reliable and less expensive electronic components to increase integration scale. Such evolutionary development is shown in Table 1.1 that also provides future development of silicon technology.

1.1 ◻ EVOLUTION OF THE INTEGRATED CIRCUIT

For the development of integrated circuits, the most significant material technologies used are silicon technology and GaAs technology. Silicon-based technology has become a dominant fabrication process for relatively high performance and cost-effective integrated circuits. On the other hand, emerging GaAs-based technology has been used in the area of ultra-high-speed logic/digital processors. Further, GaAs technology provides optoelectronic devices such as receivers and transmitters of light. GaAs in combination with silicon will provide the designer with some existing possibilities.

Silicon integrated-circuit technology has grown rapidly to increase the number of integration scale due to its well-established fabrication processing. Table 1.1 shows year-wise lowering of channel length, increase in number of transistors per chip and typical products, developments. The first transistor was invented in 1947 and integrated circuit developed in 1961. After 1970 till date, it has been the era of the integrated circuit. The table shows year-wise transition of Small-Scale Integration (SSI), Large-Scale Integration (LSI), Very Large-Scale Integration (VLSI), Ultra Large-Scale Integration (ULSI), and

Table 1.1 Yearurise lowering of channel length

Year	1947	1950–1961	1961–1966	1966–1971	1971–1980	1980–1990	1990–1999	1999–2002	2002–2005	2005–2008	2008–2011	2011–2014	2014–2016
Technology	Invention of transistors	Discrete components	SSI	MSI	LSI	VLSI	ULSI	GSLI	GSLI	GSLI	GSLI	GSLI	GSLI
Channel length	—	—	Few microns	Few microns	Few microns	Sub-microns	180 nm	130 nm	100 nm	70 nm	50 nm	30 nm	20 nm
Number of transistors per chip	1	1	~10	~100 –1000	~1000 -2×10^6	2×10^5 -10^6	10^6–2.8 $\times 10^7$	2.8×10^7 -6.4×10^7	6.4×10^7 -10^8	10^8–3.5 $\times 10^8$	>3.5 $\times 10^8$	>3.5 $\times 10^8$	>3.5 $\times 10^8$
Typical applications	—	Bipolar junction transistors	Planar devices, logic gates, flip-flops, etc.	Counters, multiplexers, adders, etc.	8-bit microprocessors, ROM, RAM, etc.	16 and 32-bit microprocessors, sophisticated RAM, etc.	Special processors/virtual real smart sensors, comm. IC, etc.	Image capture IC processors, mixed signal comm. IC etc.	Special DSP IC processor, etc.	Special DSP IC processor, etc.	High speed memory, etc.	—	—

Giant Large-Scale Integration (GLSI). The relationship between number of transistors per chip versus year is known as Moore's law after the prediction of Gordon Moore, in 1960. Out of these integrations in the table, VLSI technology finds more application than others. So, this book mostly concentrates on design and technology of VLSI related IC.

The applications such as wired communication, wireless communication, high-performance imaging system, and smart appliances require high performance, high reliability, low power dissipation, and thermal stability. In this direction, the dominant technology is silicon CMOS technology because of its relatively high performance, reliability and cost effectiveness. Although technology is continuously improving to produce smaller systems with minimum power dissipation, the IC industry faces major challenges due to thermal instability, high dynamic and static power dissipation and crosstalk. So it is required to overcome these challenges through improvement in design, material, and manufacturing processes.

1.2 ◻ INTRODUCTION OF MOS TECHNOLOGY

Integrated circuit design and implementation requires minimum power dissipation, smaller chip area, lower time delay, low production cost, higher stability, testability, and higher reliability. In this direction, silicon technology is continuously evolving to produce smaller size ICs with minimized power dissipation. Apart from this, it requires choice of proper devices for integration of the devices. Here, MOS technology is a promising technology for IC design and implementation. Within the bounds of MOS technology, the possible circuits are based on *p*MOS, *n*MOS, CMOS, and BiCMOS. Although CMOS (combination of *p*MOS and *n*MOS) is the dominant technology in VLSI design, our discussion will start with NMOS and BiCMOS. But before that, we prefer to mention advantages of CMOS over bipolar technology for VLSI design as follows:

1. CMOS technology has low-state power dissipation whereas bipolar technology has high power dissipation.
2. CMOS has high input impedance whereas bipolar devices have low input impedance.
3. CMOS has high noise margin whereas bipolar devices have low noise power margin.
4. CMOS technology has high packing density whereas bipolar technology has low packing density. High packing density of CMOS devices leads to smaller size of chips using CMOS technology.
5. Threshold voltage of CMOS devices is highly scalable in comparison to bipolar devices.
6. CMOS devices have high delay sensitivity to load whereas bipolar devices have low delay sensitivity to load.
7. CMOS devices have bidirectional capability (drain and source are interchangeable) whereas bipolar devices are essentially unidirectional.
8. CMOS devices have low transconductance whereas bipolar devices have high transconductance.
9. CMOS devices have low output drive current whereas bipolar devices have high output drive current.

1.3 ◻ BASIC IC DESIGN FLOW CHART

The CMOS circuit design consists of selection of circuit specifications including inputs and outputs, hand calculations, circuit simulations, layout design of the circuits including parasitic evaluation, fabrications, and testing. The flowchart of the same is shown in Fig. 1.1. The layout design includes area minimizations, wire-length minimizations, and routing.

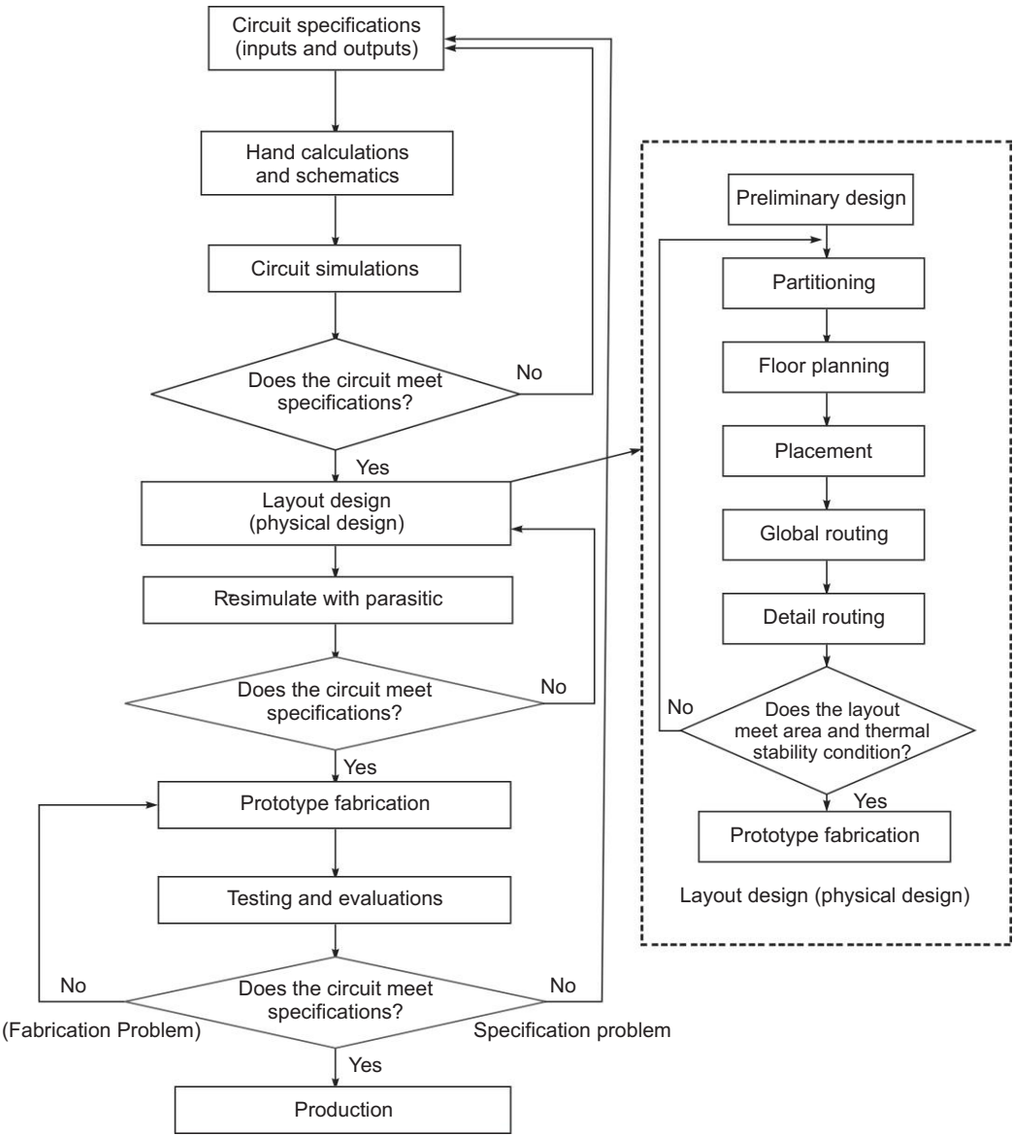


Fig. 1.1 Flow chart of CMOS IC design process

The circuit specifications are set as per requirements of applications/projects. This can be a result of trade-off between cost and performances and changes in customer needs. The circuit-design process in the figure is followed in custom IC designed chip which is also called **Application Specific Integrated Circuits (ASIC)**. Other noncustom methods of chip design use FPGA and standard cell libraries where low volume and quick implementations are important. The custom chip-design method is mainly used for development of mass-produced chips such as microprocessors, central processing unit (CPU), memory, etc.

As mentioned earlier, the layout design consists of area minimization, wire-length minimizations, and routing. So the layout design includes partitioning, floor planning and placement for area minimization, and wire length minimization and routing for delay minimization of signals as shown in Fig. 1.1. The layout design is also called a physical design. The details of this design are discussed in Chapter 8.

1.4 □ BASIC MOS TRANSISTOR

Although CMOS technology is dominant in VLSI design process, it is necessary to start from NMOS device as NMOS allows a relatively easy transition to CMOS technology. Moreover, the design methodology and design rules make the readers understand easily. NMOS technology is an excellent introduction to structured design in VLSI.

1.4.1 *n*MOS Transistor

*n*MOS devices are fabricated in *p*-substrate and the source and drain are formed by diffusing *n*-type impurities into the regions shown in Fig. 1.2, and *n*-type regions are extended mainly in lightly doped *p*-substrate. Two *p*-*n* junctions are formed by the source with *p*-substrate and drain with *p*-substrate. The establishment of current between source and drain and its control are made in two ways—enhancement mode and depletion mode. Figure 1.2(a) shows enhancement mode of an *n*MOS device whereas Fig. 1.2(b) represents depletion mode of a *n*MOS device.

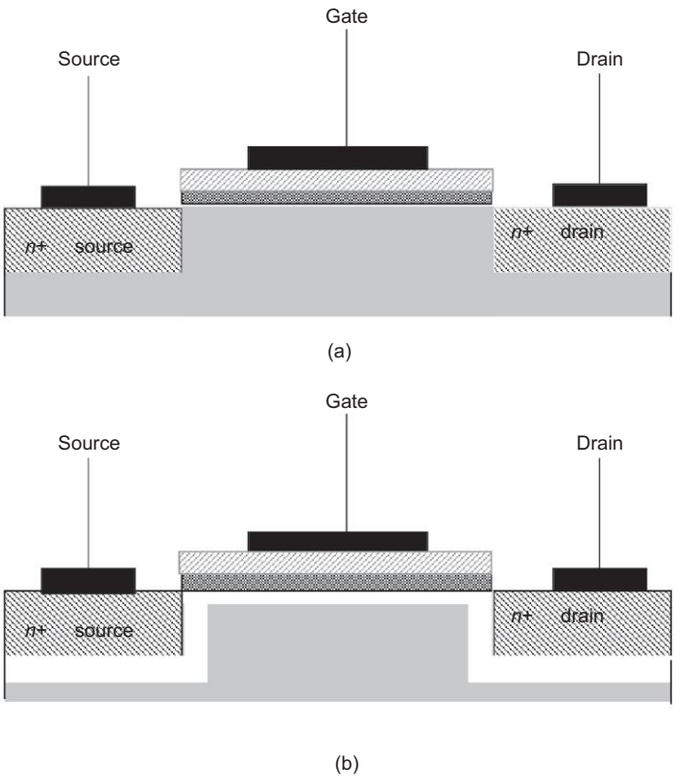
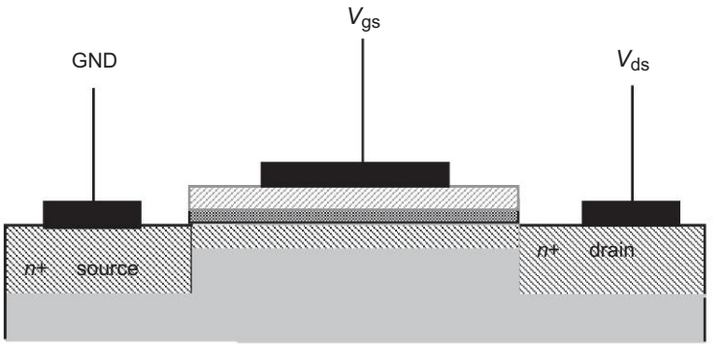


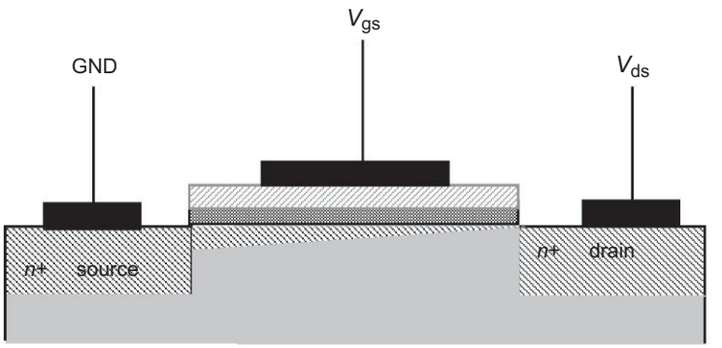
Fig. 1.2 *n*MOS transistor: (a) Enhancement mode (b) Depletion mode

1. Enhancement Mode

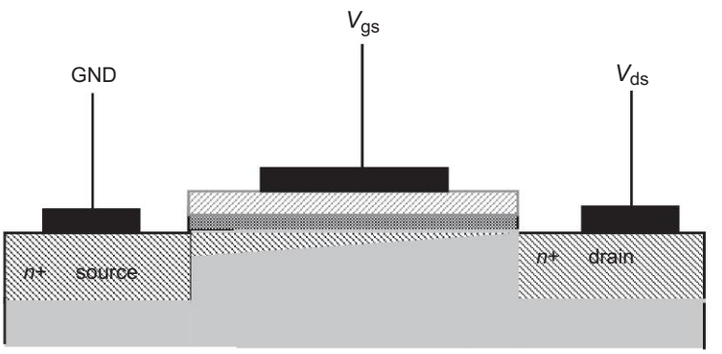
In enhancement mode, the current is established between source and drain after formation of channel. When the gate-to-source voltage $V_{gs} = V_{ds} = 0$, no channel is established and the device is in nonconducting stage. When the gate is connected to positive voltage with respect to source



(a) $V_{gs} > V_{th}$ and $V_{ds} = 0$ V



(b) $V_{gs} > V_{th}$ and $V_{ds} = V_{gs} - V_{th}$



(c) $V_{gs} > V_{th}$ and $V_{ds} > (V_{gs} - V_{th})$

Fig. 1.3 Enhancement mode MOSFET for different V_{ds} with V_{gs}

($V_{gs} > 0$), the negative charges are induced to substrate and these induced charges make the charge inversion region in the substrate in between source and drain. As a result, a conducting channel is formed in between source and drain. There are three conditions in enhancement mode. To make inversion layer for channel formation, a minimum voltage is required between gate and source and the voltage is called a **threshold voltage** (V_{th}). Figure 1.3(a) shows the condition prevailing with a channel established between source and drain, but no current flows between source and drain ($V_{ds} = 0$).

When V_{ds} is applied between source and drain in the NMOS having channel, the effective gate voltage $V_g = V_{gs} - V_{th}$ and no current flows if $V_{gs} < V_{th}$. When $V_{ds} = V_{gs} - V_{th}$ then the device is nonsaturated and the condition of the device is shown in Fig. 1.3(b).

When V_{ds} increases to be greater than $V_{gs} - V_{th}$, there is an insufficient electric field available to give creation of channel. The channel is, therefore, punched off, as shown in Fig. 1.3(c). In this condition, the diffusion current completes the path between source and drain and behaves as a constant-current source having a constant resistance. This condition is known as a **saturation condition**. In all cases, the channel will not exist and no current will flow if $V_{gs} < V_{th}$. Typically $V_{th} = 0.22\ V_{DD} \approx 1$ volt where $V_{DD} = 5$ volts.

2. Depletion Mode

For depletion mode of an *n*MOS device, the channel is established because of the implant even when $V_{gs} = 0$ and for the channel to cease to exist, a negative voltage V_{thd} must be applied between gate and source. V_{thd} is typically $< -0.8\ V_{DD}$, depending on the implant and substrate bias.

3. nMOS Fabrication

In this section, we discuss different steps used in *n*MOS fabrication. These fabrication steps are also used in CMOS and BiCMOS process along with additional fabrication steps. Figure 1.4 shows the fabrication steps used for *n*MOS fabrication and these steps are mentioned below:

(a) Processing is carried out on a thin silicon wafer cut from a single crystal doped with *p*-type impurities of concentration $10^{15}/cm^3$ to $10^{16}/cm^3$

(b) A layer of SiO_2 (typically, 1 μm thick) is grown all over the surface of the wafer by using wet thermal diffusion method. It acts as a barrier to dopants during processing.

(c) The surface is then deposited with photoresists by a spin-coating machine with uniform thickness.

(d) The photoresist layers on the wafer surface are then exposed to the ultraviolet light through the mask containing transistor channels. The exposed areas of the layers are polymerized and unexposed areas are unaffected. After the development of these layers, unaffected areas are dissolved. This process is called **photolithography**.

(e) By SiO_2 etch ant, SiO_2 layers are removed from unexposed areas.

(f) A thin layer of SiO_2 is again grown over the chip and polysilicon is deposited on top of this to form gate structure by using Chemical Vapour Deposition (CVD).

(g) The $n+$ diffusion layer is made through a mask containing the source and drain by using photolithography. The $n+$ diffusion is achieved by heating the wafer to a high temperature and passing a gas having *n*-type impurity (phosphorous) over the surface.

(h) Again, a thick SiO_2 layer is grown over the surface.

(i) Using a mask containing metallic connection for source, drain and gate, and photolithography, the aluminum connection pads are made through deposition.

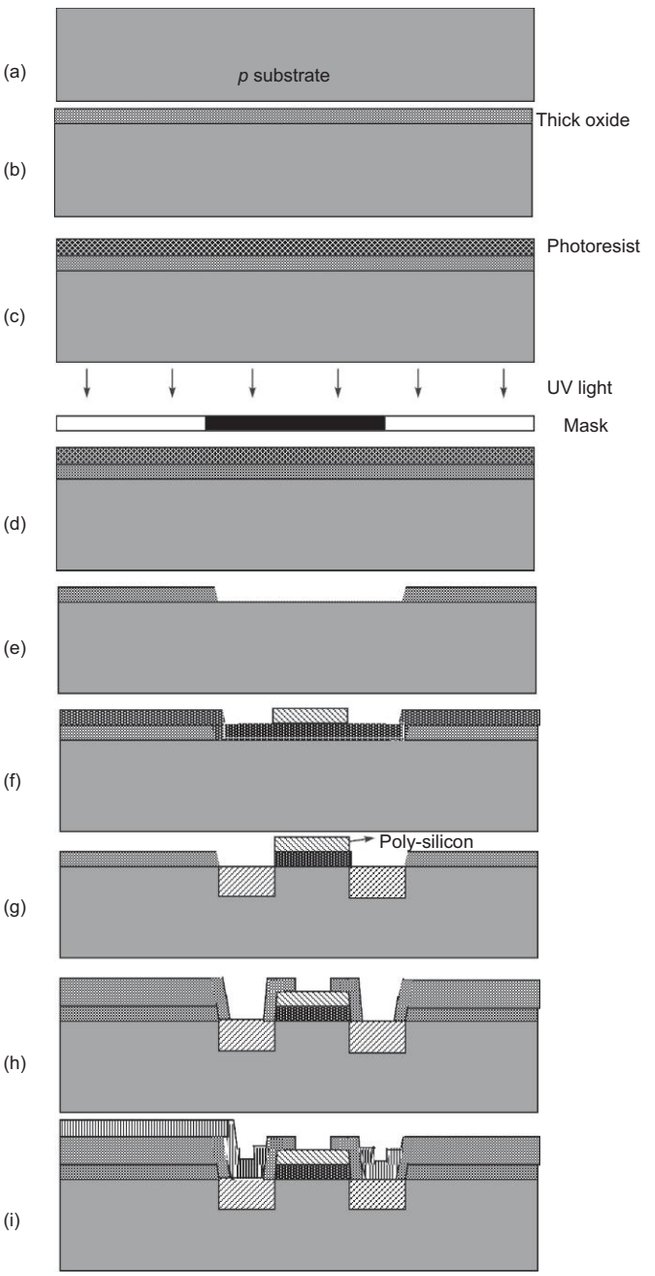


Fig. 1.4 *n*MOS fabrication steps

1.4.2 *p*MOS Device

A *p*MOS device consisting of *p*+ source, *p*+ drain and gate is made on *n*-type substrate as shown in Fig. 1.5. Like an *n*MOS device, *p*MOS device has enhancement and depletion modes. The gate-to-source voltage to achieve these modes is opposite to those of an *n*MOS device.

The fabrication steps required for *p*MOS are similar to those of *n*MOS devices. The difference is that: *n*-substrate is used instead of *p*– substrate and *p*+ diffusions are made to form source and drain in *p*MOS.

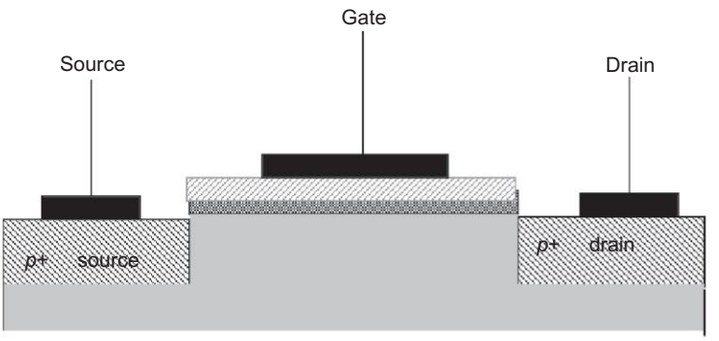


Fig. 1.5 *p*MOS device

1.4.3 CMOS Device Processing

CMOS device is a combination of *p*MOS and *n*MOS. There are two types of device processing—*n*-well CMOS and *p*-well CMOS device processing. Although *p*-well fabrication is widely used, *n*-well fabrication has advantages such as lower substrate bias requirement, lower threshold voltage, and lower parasitic capacitances associated with source and drain regions. In *p*-well CMOS, *n*-type substrate is used whereas in *n*-well CMOS, *p*-substrate is used.

Figure 1.6 shows the basic processing steps used for *p*-well processing. As mentioned earlier, the structure consists of *n*-type substrate in which a *p*-well device is formed by using suitable mask and diffusion. For *n*MOS device fabrication, a deep *p*-well diffusion is made in *n*-substrate and to achieve threshold voltage of 0.6 volt to 1 volt, we need diffusion of *p*-type impurity of high resistivity for *p* well. Typical processing steps of masking, patterning, and diffusion are given below:

1. p-well CMOS Device Fabrication

Step 1: The *p*-well region is made by using mask-1 and diffusion of deep *p*-well impurity into *n*-type substrate.

Step 2: *n*MOS and *p*MOS active regions are formed by using mask-2.

Step 3: Gate oxidation (thinox) region is defined.

Step 4: Formation and patterning of polysilicon layer are made by using mask-3.

Step 5: Mask-4 having *p*+ diffusion layer is used to define all areas of *p*+ diffusion and *p*+ diffusion is made on these regions.

Step 6: Mask-5 having *n*+ diffusion layer is used to define all areas of *n*+ diffusion and *n*+ diffusion is made on these regions.

Step 7: Contact cut areas are defined by using mask-6 and contacts are made.

Step 8: The metal layers are formed by using mask-7.

Step 9: Overall glass with cuts for bonding pads are made by using mask-8.

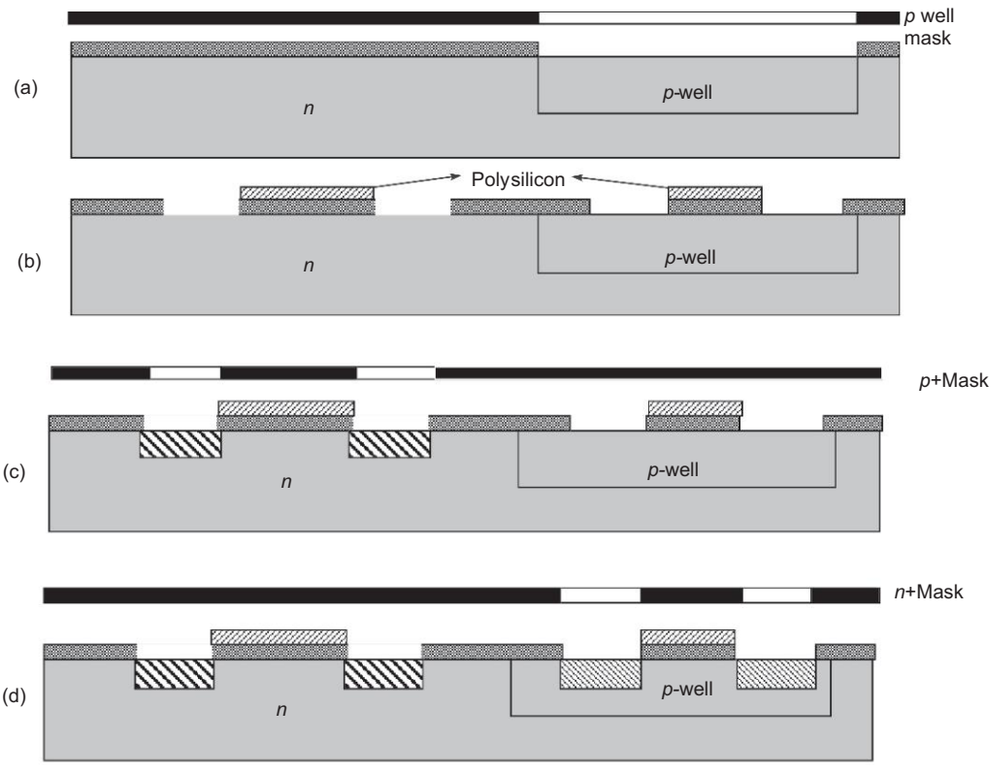


Fig. 1.6 CMOS fabrication process steps: (a) *p*-well formation (b) polysilicon layer for GATE formation (c) *p*+ diffusion for *p*MOS (d) *n*+ diffusion for *n*MOS

2. *n-well Process*

The fabrications steps for the *n*-well process are almost same as the *p*-well process except the following. In an *n*-well CMOS device, *n*-well deep diffusion is made inside the *p*-substrate. The steps are given below:

Step 1: The *n*-well region is formed by using mask-1 and diffusion of deep *p*-well impurity into *p*-type substrate.

Step 2: *n*MOS and *p*MOS active regions are formed by using mask-2.

Step 3: Gate oxidation (thinox) region is defined.

Step 4: Formation and patterning of polysilicon layers are made by using mask-3.

Step 5: Mask-4 having *n*+ diffusion layer is used to define all areas of *n*+ diffusion and *n*+ diffusion is made on these regions.

Step 6: Mask-5 having *p*+ diffusion layer is used to define all areas of *p*+ diffusion and *p*+ diffusion is made on these regions.

Step 7: Contact cut area are defined by using mask-6 and contacts are made.

Step 8: The metal layers are formed by using mask-7.

Step 9: Overall glass with cuts for bonding pads are made by using mask-8.

1.4.4 BiCMOS Device Processing

There is a deficiency of MOS technology due to the limited load-driving capabilities which is because of limited current-sourcing and current-sinking abilities associated with *p*- and *n*-transistors. Bipolar transistors also provide higher gain and have generally better nose and high-frequency characteristics than MOS transistors. CMOS combined with bipolar transistors may be an effective way of speeding up of VLSI circuits. This combined device is called a BiCMOS. By using BiCMOS technology, we can improve the speed of ALU, ROM, and barrel switch, etc. There are two types of BiCMOS—*n*-well and *p*-well BiCMOS devices. The fabrication steps of BiCMOS are same as CMOS device fabrication with additional steps required for fabrication for bipolar transistors. The fabrication steps for *n*-well process are almost same as *p*-well process except the following. In *n*-well BiCMOS device, *n*-well deep diffusion is made inside *p*-substrate. The steps are given below:

Step 1: *n*-well region is formed by using mask-1 and diffusion of deep *p*-well impurity into *p*-type substrate.

Step 2: *n*MOS and *p*MOS active regions are formed by using mask-2.

Step 3: Gate oxidation (thinox) region is defined.

Step 4: Formation and patterning of polysilicon layers are made by using mask-3.

Step 5: Mask-4 having *n*+ diffusion layer is used to define all areas of *n*+ diffusion and *n*+ diffusion is made on these regions.

Step 6: Mask-5 having *p*+ diffusion layer is used to define all areas of *p*+ diffusion and *p*+ diffusion is made on these regions.

Step 7: Contact cut areas are defined by using mask-6 and contacts are made.

Step 8: The metal layers are formed by using mask-7.

Step 9: Overall glass with cuts for bonding pads are made by using mask-8.

REFERENCES

1.1 Hutchby J., Bourian off G. Zhrirnor, and Brewer J., "Extending the road beyond CMOS", *IEE Circuits Devices and Systems* March 2002, pp 28–41.

1.2 Website: http://public.itrs.net.

1.3 N.H.E Weste and E. Eshraghian, *Principles of CMOS VLSI Design*, Addison Wesley, 2nd ed. 1993, ISBN 0-2d-53376-6.

EXERCISES

1.1 Explain why CMOS is preferred for IC design over bipolar transistors.

1.2 Design different steps for fabrication of BiCMOS devices. What are the additional steps required in BiCMOS apart from the CMOS processing steps?

1.3 Describe different steps used for layout design with a flow chart.

1.4 Design different steps for fabrication of a CMOS device. What are additional steps required in CMOS apart from *n*MOS processing steps?

1.5 Design different steps for fabrication of an *n*MOS device.

1.6 How is *p*MOS operated in enhancement mode? How is *p*MOS operated in depletion mode?

2

MOSFET and CMOS: Basic Electrical Properties and Circuit Design

VLSI IC design based on MOS/CMOS technology and performances of the circuits are properly understood and realized only if MOSFET devices are known. Before discussion of CMOS devices, we should understand basic electrical properties of *n*MOS transistors. The expressions and discussions related to *p*MOS transistors are same as *n*MOS with a reversal of voltage and current exchange of μ_n for μ_p and electrons for holes.

2.1 ▫ DRAIN-TO-SOURCE CURRENT I_{ds} VS V_{ds} CHARACTERISTICS OF *n*MOS

Figure 2.1 shows *n*MOS consisting of *p*-substrate, *n* diffusion source and drain, and oxide-layer gate with a polysilicon layer in between source and drain. The electrical properties/concept of MOS transistors come from application of a voltage on the gate to induce a change in the channel between the source and drain which may then be caused to move from source to drain under the influence of an electrical field created by a voltage V_{ds} applied between the drain and source. Since the change induced is independent of the gate-to-source, current I_{ds} is dependent on applied V_{gs} and V_{ds} voltages and is given by

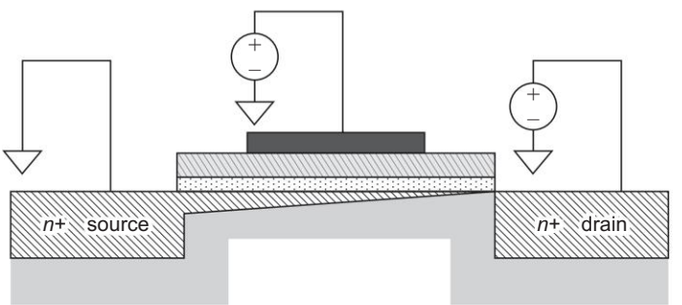


Fig. 2.1 Cross-sectional view of *n*MOSFET

$$I_{ds} = \frac{\text{Charge induced in channel } Q_I}{\text{Electron transit true } (\tau_{sd})} \tag{2.1}$$

where I_{ds} current is flowing in opposite direction to flow of electrons which are charge carriers between source and drain.

The velocity of electrons is written as

$$v = \mu E_{ds} = \mu_n \frac{V_{ds}}{L} \tag{2.2}$$

where L = Length of channel, μ_n = Mobility of electrons, and E_{ds} = Electric field applied between drain and source. The electron transit true τ_{sd} is written as

$$\tau_{sd} = \frac{L}{v} = \frac{L^2}{\mu_n V_{ds}} \tag{2.3}$$

The charge induced in the channel due to gate voltage is due to the voltage difference between the gate and channel at a distance x away from the source which is labeled as $V(x)$. So the potential difference between the gate and channel distance from the source is given by

$$Q'_{ch} = C'_{ox} [V_{gs} - V_{(x)}] \tag{2.4}$$

where C'_{ox}, $\frac{\varepsilon_{ox}}{D}$, D = Width of oxide layer, and ε = Dielectric constant of oxide layer.

We already know that Q_b is present in the inversion layer from the application of threshold voltage V_{THN} which is necessary for making inversion channel between the drain and source. The Q'_b is given by

$$Q'_b = C'_{ox} V_{THN} \tag{2.5}$$

So, effectively the charge participated for conduction of a current between drain and source is given by

$$\begin{aligned} Q'(x) &= Q'_{ch} - Q'_b \\ &= C'_{ox} [V_{gs} - V(x) - V_{THN}] \end{aligned} \tag{2.6}$$

The differential resistance of channel region of length dx and width W is given by

$$dR = \frac{1}{\mu_n Q'(x)} \frac{dx}{W} \tag{2.7}$$

where $\frac{1}{\mu_n Q'(x)}$ = Effective sheet resistance.

The differential voltage drop is given by

$$dV(x) = I_{ds}\, dR = \frac{I_{ds}}{W \mu_n Q'_{(n)}}.\, dx \tag{2.8}$$

We can write from equations (2.6), (2.7), and (2.8)

$$I_{ds}\, .\, dx = W \mu_n\, C'_{ox} [V_{gs} - V(x) - V_{THN}]\, dV(x) \tag{2.9}$$

The current can be obtained by integrating the left-hand side of Eq. (2.9) from 0 to L and right-hand side from 0 to L and right-hand side from 0 to V_{ds} and is given by

$$I_{ds} \int dx = W \mu_n\, C'_{ox} \int [(V_{gs} - V(x) - V_{THN})]\, dv(x)$$

$$I_{ds}\,L = W\mu_n\,C'_{ox}\left[(V_{gs}-V_{THN})V_{ds}-\frac{V_{ds}^2}{2}\right]$$

where $V_{gs} \geq V_{THN}$ and $V_{ds} \leq V_{gs} - V_{THN}$

So
$$I_{ds} = \frac{W\mu_n C'_{ox}}{L}\left[(V_{gs}-V_{THN})V_{ds}-\frac{V_{ds}^2}{2}\right] \tag{2.10}$$

where $K_n = \mu_n\,C'_{ox}$ = Transconduction parameter. W/L is the parameter which is defined from the geometry of *n*MOS and it is a common practice to define the parameter as

$$\beta_n = \frac{K_n W}{L}$$

So I_{ds} from Eq. (2.10) is written as

$$I_{ds} = \beta_n\left[(V_{gs}-V_{THN})V_{ds}-\frac{V_{ds}^2}{2}\right]$$

For nonsaturated or resistive region $V_{ds} < V_{gs} - V_{THN}$

$$I_{ds} = \beta_n\left[(V_{gs}-V_{THN})V_{ds}-\frac{V_{ds}^2}{2}\right] \tag{2.11}$$

Saturation Region

For saturation region, $V_{s.} = V_{s.} - V_{THN}$. At the saturation region, the drain source voltage (IR drop) is equal to effective gate-to-channel voltage. The current I_{ds} at this region is independent of V_{ds}. So I_{ds} is written as

$$I_{ds} = \frac{K_n W}{L}\cdot\frac{(V_{gs}-V_{THN})^2}{2}$$

In terms of β, we can write

$$I_{ds} = \frac{\beta}{2}(V_{gs}-V_{THN})^2 \tag{2.12}$$

At the saturation region, as V_{ds} increases, the current I_{ds} remains constant. But V_{ds} increases further and further, the depletion region increases from drain to source. The device is said to be punched through. The voltage V_{ds} is called punch-through voltage. It also seems that at saturation region, the current I_{ds} includes also channel-length modulation where the depletion-layer width increases with increase of V_{ds}. The electrical channel length is written as

$$L_{eleu} = L - X_{dl}$$

where X_{dl} = Depletion-layer length between drain layer and channel.

So we can write

$$I_{ds} = \frac{K_n W}{2L_{eleu}}(V_{gs}-V_{THN})^2$$

So change of I_{ds} with respect to V_{ds} can be written as

$$\frac{\partial I_{ds}}{\partial V_{ds}} = -\frac{K_n W}{2L_{eleu}^2}(V_{gs}-V_{THN})^2\cdot\frac{\partial L_{eleu}}{\partial V_{ds}}$$

$$= I_{ds} \cdot \frac{1}{L_{eleu}} \cdot \frac{dX_{dl}}{\partial V_{ds}}$$

We define channel-length modulation parameters $\lambda_C = \frac{1}{L_{eleu}} \cdot \frac{dX_{dl}}{\partial V_{ds}}$

So we can write

$$I_{ds} = \frac{\beta}{2} (V_{gs} - V_{THN})^2 [1 + \lambda_c (V_{ds} - V_{ds,Sat})] \tag{2.13}$$

where $V_{ds,Dat} = V_{gs} - V_{THN}$. In case of digital application, we assume $\lambda_c = 0$ but in case of analog application, λ_c is considered for analog MOSFET circuit analysis. Figure 2.2(a) shows typical characteristics for *n*MOS transistors providing I_{ds} versus V_{ds} at different V_{gs}. The figure shows saturation $V_{ds} = V_{gs} - V_{THN}$ at which the velocity of electrons saturates. As V_{ds} increases above $V_{gs} - V_{THN}$, the mobility of electrons decreases and it causes reduction of saturated values of V_{ds} and I_{ds}.

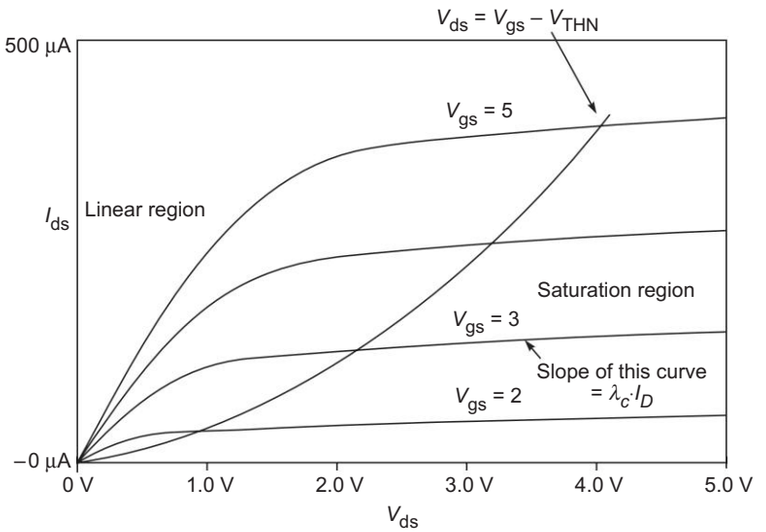


Fig. 2.2 (a) Typical I_{ds} vs V_{ds} characteristics of *n*MOS

It is also observed that the second-order current-voltage equation as given in Eq. (2.11) gives rise to a set of inverted parabolas for each constant V_{GS} value.

2.1.1 Transconductance g_m and Output Conductance of *n*MOS

The transconductance relationship between output current I_{ds} and input voltage V_{gs} is defined as

$$g_m = \frac{\partial I_{ds}}{\partial V_{gs}} \Big| V_{ds} = \text{Constant}$$

$$= \frac{\partial}{\partial V_{gs}} \left[\beta \{ (V_{gs} - V_{THN}) V_{ds} - V_{ds}^2 / 2 \} \right]$$

$$= \frac{\beta}{2}[V_{ds}] = \beta V_{ds} \tag{2.14}$$

where $$I_{ds} = \beta\left[(V_{gs} - V_{THN})V_{ds} - \frac{V_{ds}^2}{2}\right]$$

At saturation region,

$$I_m = \beta(V_{gs} - V_{THN})\,[1 + \lambda_c(V_{ds} - V_{ds,Sat})]$$

The output conductance is written as

$$g_{ds} = \frac{\partial I_{ds}}{\partial V_{ds}}\ V_{gs} = \text{Constant}$$

$$= \beta\,[(V_{gs} - V_{THN}) - V_{ds}]$$

At saturation region,

$$g_{ds} = \frac{\partial I_{ds}}{\partial V_{ds}}\Big|_{V_{gs}\,=\,\text{constant}}$$

$$= \frac{\lambda_C \beta}{2}(V_{gs} - V_{THN})^2 \tag{2.15}$$

Frequency response of MOS transistor is estimated from the parameter ω_o which is called **figure of merit**. It represents the switching speed depending on gate voltage above threshold and on carrier mobility, and is inversely proportional to the square of channel length. It is expressed as

$$\omega_o = \frac{g_m}{C_g} = \frac{\mu}{L^2}(V_{gs} - V_{THN}) \tag{2.16}$$

where C_g = Gate capacitance.

2.1.2 Body Effect

The transistors in a MOS device seen so far are built on a common substrate. Thus, the substrate voltage of all such transistors is equal. However, while one designs a complex gate using MOS transistors, several devices may have to be connected in series. This will result in different source-to-substrate voltages for different devices. For example, in the NAND gate (as discussed in Chapter 3), the *n*MOS transistors are in series, whereby the source-to-substrate voltage V_{SB} of the device corresponding to the input A is higher than that of the device for the input B. Under normal conditions ($V_{GS} > V_T$, V_T = Threshold voltage of a MOS transistor), the depletion-layer width remains unchanged and the charge carriers are drawn into the channel from the source. As the substrate bias V_{SB} is increased, the depletion-layer width corresponding to the source-substrate field-induced junction also increases. This results in an increase in the density of the fixed charges in the depletion layer. For charge neutrality to be valid, the channel charge must go down. The consequence is that the substrate bias V_{SB} gets added to the channel-substrate junction potential. This leads to an increase of the gate-channel voltage drop. This is called **body effect** which mainly influences threshold voltage—the minimum amount of the gate-to-source voltage V_{GS} necessary to cause surface inversion so as to create the conducting channel between the source and the drain. For $V_{GS} < V_{THN}$, no current can flow between the source and the drain. For $V_{GS} > V_T$, a larger number of minority carriers (electrons in case of an *n*MOS transistor) are drawn to the surface,

increasing the channel current. However, the surface potential and the depletion-region width remain almost unchanged as V_{GS} is increased beyond the threshold voltage.

The physical components determining the threshold voltage are the following:

- Work-function difference between the gate and the substrate
- Gate-voltage portion spent to change the surface potential
- Gate-voltage part accounting for the depletion region charge
- Gate-voltage component to offset the fixed charges in the gate oxide and the silicon-oxide boundary

Although the following analysis pertains to an *n*MOS device, it can be simply modified to reason for a *p*-channel device. The work function difference ϕ_{GS}, between the doped polysilicon gate and the *p*-type substrate, which depends on the substrate doping, makes up the first component of the threshold voltage. The externally applied gate voltage must also account for the strong inversion at the surface, expressed in the form of surface potential $2\phi_F$, where ϕ_F denotes the distance between the intrinsic energy level E_I and the Fermi level E_F of the *p*-type semiconductor substrate. The factor 2 comes due to the fact that in the bulk, the semiconductor is *p*-type, where E_I is above E_F by ϕ_F, while at the inverted *n*-type region at the surface E_I is below E_F by ϕ_F, and thus the amount of the band bending is $2\phi_F$. This is the second component of the threshold voltage. The potential difference ϕ_F between E_I and E_F is given as

$$\phi_F = \frac{kT}{q} \ln\left(\frac{N_A}{n_i}\right)$$

where k = Boltzmann constant, T = Temperature, q = Electron charge, N_A = Acceptor concentration in the *p*-substrate and n_i = Intrinsic carrier concentration. The expression $\frac{kT}{q}$ is 0.02586 volt at 300 K. The applied gate voltage must also be large enough to create the depletion charge. Note that the charge per unit area in the depletion region at strong inversion is given by

$$Q_{d0} = -2(\varepsilon_s q N_A \phi_F)^{1/2}$$

where ε_s is the substrate permittivity. If the source is biased at a potential V_{SB} with respect to the substrate then the depletion charge density is given by

$$Q_d = -2(\varepsilon_s q N_A (\phi_F + V_{SB}))^{1/2}$$

The component of the threshold voltage that offsets the depletion charge is then given by $-Q_d/C_{ox}$, where C_{ox} is the gate oxide capacitance per unit area, or $C_{ox} = \varepsilon_{ox}/t_{ox}$, (ratio of the oxide permittivity and the oxide thickness). A set of positive charges arises from the interface states at the Si–SiO_2 interface. These charges, denoted as Q_i, occur from the abrupt termination of the semiconductor crystal lattice at the oxide interface. The component of the gate voltage needed to offset this positive charge (which induces an equivalent negative charge in the semiconductor) is $-Q_i/C_{ox}$. On combining all the four voltage components, the threshold voltage V_{TO}, for zero substrate bias, is expressed as

$$V_{T0} = \phi_{GS} - 2\phi_F - \frac{Q_{d0}}{C_{ox}} - \frac{Q_i}{C_{ox}}$$

For non-zero substrate bias, however, the depletion charge density needs to be modified to include the effect of V_{SB} on that charge, resulting in the following generalized expression for the threshold voltage, namely

$$V_T = \phi_{GS} - 2\phi_F - \frac{Q_d}{C_{ox}} - \frac{Q_i}{C_{ox}}$$

The generalized form of the threshold voltage can also be written as

$$V_T = \phi_{GS} - 2\phi_F - \frac{Q_{d0}}{C_{ox}} - \frac{Q_i}{C_{ox}} - \frac{Q_d - Q_{d0}}{C_{ox}} = V_{T0} - \frac{Q_d - Q_{d0}}{C_{ox}}$$

The threshold voltage differs from V_{TO} by an additive term due to substrate bias. This term, which depends on the material parameters and the source-to-substrate voltage V_{SB}, is given by

$$\frac{Q_d - Q_{d0}}{C_{ox}} = -\frac{\sqrt{2qN_A\varepsilon_S}}{C_{ox}}\left(\sqrt{|2\phi_F + V_{SB}|} - \sqrt{|2\phi_F|}\right)$$

Thus, in its most general form, the threshold voltage is determined as

$$V_T = V_{T0} + \gamma\left(\sqrt{|2\phi_F + V_{SB}|} - \sqrt{|2\phi_F|}\right)$$

in which the parameter γ, known as the substrate-bias *(*or body-effect) coefficient, is given by

$$\gamma = \frac{\sqrt{2qN_A\varepsilon_S}}{C_{ox}}$$

Example 2.1 ¤ *For a substrate doping of 10^{15} atoms/cm^3, $V_{gs} = V_{THN}$ and V_{sb} = Substrate-to-source bias voltage = 0, estimate the electrostatic potential in the substrate region and at the oxide-semiconductor interface*.

Solution: The electrostatic potential of the substrate is given by

$$\frac{kT}{q}\ln\frac{N_A}{n_i} = -26\text{ mV}\ln\frac{10^5}{14.5\times10^9}$$

where n_i = Intrinsic carrier concentration at room temperature (25^0 C) = 14.5×10^9 atom/cm^3

Example 2.2 ¤ Consider the *n*-channel MOS process in Example 2.1. One may examine how a non-zero source-to-substrate voltage V_{SB} influences the threshold voltage of an *n*MOS transistor.

One can calculate the substrate-bias coefficient γ using the parameters provided in Example 2.1 as follows:

$$v = \frac{\sqrt{2qN_A\varepsilon_S}}{C_{ox}} = \frac{\sqrt{2\times1.6\times10^{-19}\times10^6\times11.7\times8.85\times10^{-14}}}{7.03\times10^{-18}} = 0.82V^{\frac{1}{2}}$$

One is now in a position to determine the variation of threshold voltage V_T as a function of the source-to-substrate voltage V_{SB}. Assume the voltage V_{SB} to range from 0 to 5 V.

$$V_T = V_{T0} + \gamma\left(\sqrt{|2\phi_F + V_{SB}|} - \sqrt{|2\phi_F|}\right) = 0.40 + 0.82\left(\sqrt{0.7 + V_{SB}} - \sqrt{0.7}\right)$$

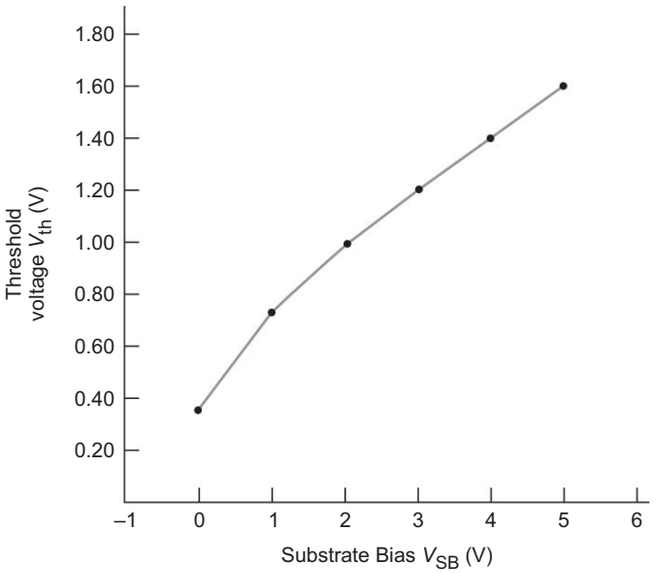


Fig. 2.2 (b) Variation of threshold voltage in response to change in source-to-substrate voltage V_{SB}

Figure 2.2(b) depicts the manner in which the threshold voltage V_{th} varies as a function of the source-to-substrate voltage V_{SB}. As may be seen from the figure, the extent of the variation of the threshold voltage is nearly 1.3 volts in this range. In most digital circuits, the substrate-bias effect (also referred to as the body effect) is inevitable. Accordingly, appropriate measures have to be adopted to compensate for such variations in the threshold voltage.

2.2 □ SECOND-ORDER EFFECTS

The current-voltage equations discussed in Section 2.1 are ideal in nature and have been derived keeping various secondary effects out of consideration. In this section, these secondary effects such as body effect, drain punch-through effect, and subthreshold region conduction are discussed.

2.2.1 Threshold Voltage and Body Effect

As discussed in Section 2.1.2, the threshold voltage V_{THN} does vary with the voltage difference V_{SB} between the source and the body (substrate). Thus, including this difference, the generalized expression for the threshold voltage is written as

$$V_T = V_{T0} + \gamma\left(\sqrt{|2\phi_F + V_{SB}|} - \sqrt{|2\phi_F|}\right)$$

in which the parameter γ is known as the **substrate-bias** (or **body effect**) coefficient and is given by

$$\gamma = \frac{\sqrt{2qN_A\varepsilon_S}}{C_{ox}}$$

Typical values of γ range from 0.4 to 1.2.

2.2.2 Drain Punch-through

In a MOSFET device with improperly scaled small channel length and too low channel doping, undesired electrostatic interaction can take place between the source and the drain known as **Drain-Induced Barrier Lowering (DIBL).** This leads to punch-through leakage or breakdown between the source and the drain, and loss of gate control. One should consider the surface potential along the channel to understand the punch-through phenomenon. As the drain bias increases, the conduction-band edge (which represents the electron energies) in the drain is pulled down, leading to an increase in the drain-channel depletion width.

In a long-channel device, the drain bias does not influence the source-to-channel potential barrier, and it depends on the increase of gate bias to cause the drain current to flow. However, in a short-channel device, as a result of increase in drain bias and pull-down of the conduction band edge, the source-channel potential barrier is lowered due to DIBL. This, in turn, causes drain current to flow regardless of the gate voltage (that is, even if it is below the threshold voltage V_T). More simply, the advent of DIBL may be explained by the expansion of drain depletion region and its eventual merging with source depletion region, causing punch-through breakdown between the source and the drain. The punch-through condition puts a natural constraint on the voltages across the internal circuit nodes.

2.2.3 Subthreshold Region Conduction

The cut-off region of operation is also referred to as the subthreshold region, which is mathematically expressed as $I_{DS} = 0$ and $V_{GS} < V_T$. In the subthreshold region, **subthreshold conduction** takes place in small-geometry transistors. Normally, the current flow in the channel depends on creating and maintaining an inversion layer on the surface. If the gate voltage is inadequate to invert the surface (i.e., $V_{GS} < V_T$), the electrons in the channel encounter a **potential barrier** that blocks the flow. However, in small-geometry MOSFETs, this potential barrier is controlled by both V_{GS} and V_{DS}. If the drain voltage is increased, the potential barrier in the channel decreases, leading to drain-induced barrier lowering (DIBL). The lowered potential barrier finally leads to flow of electrons between the source and the drain, even if $V_{GS} < V_T$ (i.e., even when the surface is not in strong inversion). The channel current flowing in this condition is called the **subthreshold current**. This current, due mainly to diffusion between the source and the drain, causes concern in deep submicron designs. The model implemented in SPICE brings in an exponential, semi-empirical dependence of the drain current on V_{GS} in the **weak inversion region**. Defining a voltage V_{on} as the boundary between the regions of weak and strong inversion, the drain current I_D can be written as

$$I_{D\,(\text{weak inversion})} = I_{on} \cdot e^{(V_{GS} - V_{on})\left(\frac{q}{nkT}\right)}$$

where I_{on} is the current in strong inversion for $V_{GS} = V_{on}$.

Channel-Length Modulation

So far, the variations in channel length is not considered due to the changes in drain-to-source voltage V_{DS}. For long-channel transistors, the effect of channel-length variation is not prominent. With the decrease in channel length, however, the variation matters. The inversion layer reduces to a point at the drain end when $V_{DS} = V_{DS\,(SAT)} = V_{GS} - V_T$. That is, the channel is **pinched off** at the drain end. The onset of saturation-mode operation is indicated by the pinch-off event. If the drain-to-source voltage

is increased beyond the saturation edge ($V_{DS} > V_{DSAT}$), a still larger portion of the channel becomes pinched off. Let the effective channel (i.e., the length of the inversion layer) be $L_{eff} = L - \Delta L$, where L = Original channel length (the device being in nonsaturated mode), and ΔL = Length of the channel segment where the inversion layer charge is zero. Thus, the pinch-off point moves from the drain end toward V_{DS}, the source with increasing drain-to-source voltage. The remaining portion of the channel between the pinch-off point and the drain end will be in depletion mode. For the shortened channel, with an effective channel voltage of $V_{DS(SAT)}$, the channel current is given by

$$I_{DS(SAT)} = \frac{\mu_n C_{ox}}{2} \cdot \frac{W}{L_{eff}} \cdot (V_{GS} - V_{T0})^2$$

The current expression pertains to a MOSFET with effective channel length L_{eff}, operating in saturation. The above equation depicts the condition known as **channel-length modulation**, where the channel is reduced in length. As the effective length decreases with increasing V_{DS}, the saturation current $I_{DS(SAT)}$ will consequently increase with increasing V_{DS}. The current $I_{DS(SAT)}$ can be rewritten as

$$I_{DS(SAT)} = \frac{\mu_n C_{ox}}{2} \left(\frac{1}{1 - \frac{\Delta L}{L}} \right) \frac{W}{L} \cdot (V_{GS} - V_{T0})^2$$

The second term on the right-hand side of Eq. (2.12) accounts for the channel modulation effect. It can be shown that the factor channel length ΔL is expressed as

$$\Delta L \; \alpha \; \sqrt{V_{DS} - V_{DS(SAT)}}$$

One can even use the empirical relation between ΔL and V_{DS} given as follows.

$$1 - \frac{\Delta L}{L} \approx 1 - \lambda V_{DS}$$

The parameter λ is called the **channel-length modulation coefficient**, having a value in the range 0.02 V to 0.005 V. Assuming that $\lambda V_{DS} >> 1$, the saturation current can be written as

$$I_{DS(SAT)} = \frac{\mu_n C_{ox}}{2} \cdot \frac{W}{L_{eff}} \; (V_{GS} - V_{T0})^2 \cdot (1 + \lambda V_{DS})$$

The above simplified equation shows a linear dependence of the saturation current on the drain-to-source voltage. The slope of the current-voltage characteristic in the saturation region is determined by the channel length modulation factor λ.

2.2.4 Impact Ionization

An electron traveling from the source to the drain along the channel gets kinetic energy at the cost of electrostatic potential energy in the pinch-off region and becomes a 'hot' electron. As the hot electrons travel towards the drain, they can generate secondary electron-hole pairs by impact ionization. The secondary electrons are collected at the drain and cause the drain current in saturation to increase with drain bias at high voltages, thus leading to a fall in the output impedance. The secondary holes are collected as substrate current. This effect is called impact ionization. The hot electrons can even penetrate the gate oxide, causing a gate current. This finally leads to degradation in MOSFET parameters like increase of threshold voltage and decrease of transconductance. Impact ionization can create problems

such as noise in mixed-signal systems, poor refresh times in dynamic memories, or latch-up in CMOS circuits. The remedy to this problem is to use a device with a lightly doped drain. By reducing the doping density in the source/drain, the depletion width at the reverse-biased drain-channel junction is increased and consequently, the electric field is reduced. Hot carrier effects do not normally present an acute problem for *p*-channel MOSFETs. This is because the channel mobility of holes is almost half that of the electrons. Thus, for the same field, there are fewer hot holes than hot electrons. However, lower hole mobility results in lower drive currents in *p*-channel devices than in *n*-channel devices.

2.3 □ DRAIN TO SOURCE CURRENT I_{ds} VS V_{ds} OF *p*MOS

The *p*MOS is constructed with *n*-substrate, *p*-diffusion source and drain, and $S_i O_2$ layer with polysilicon layer on oxide layer in between source and drain as shown in Fig. 2.3. The charges are induced to the channel in the *p*-substrate region in between the source and drain with application of V_{gs} voltage. With application of source-to-drain voltage V_{ds}, the induced charges from source to drain cause a source-to-drain current $I_{sd.}$ Like *n*-MOSFET, the source-to-drain current due to induced charge channel and V_{ds} for *p*MOS can be written as

$$I_{sd} = \beta_p \left[(V_{sg} - V_{THP}) V_D - \frac{V_{SD}^2}{2} \right] \tag{2.17}$$

where $\beta_p = \dfrac{K_p \cdot W}{L}$ K_P = *p*MOS transconductance parameter. $V_{sg} = -V_{gs}$ and V_{THP} = threshold *p*MOS voltage, $V_{sd} = -V_{ds}$ and $V_{sg} \geq V_{THP}$, and $V_{sd} \leq V_{sg} - V_{THP}$

For saturation region, $V_{sd} = V_{sg} - V_{THP}$

$$I_{sd} = \frac{\beta_p}{2} [(V_{sg} - V_{THP})^2] \tag{2.18}$$

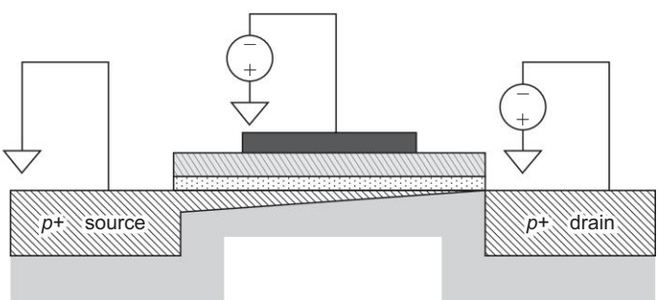


Fig. 2.3 Cross-sectional view of *p*MOS

2.4 □ THE *p*MOS TRANSISTOR'S THRESHOLD VOLTAGE, V_{THP}

The threshold voltage of a MOSFET depends on the gate structure of MOS transistors in which changes are stored in the dielectric oxide layer and in the substrate-oxide layer interface.

The threshold voltage may be expressed as

$$V_{TH} = \varphi_{ms} \frac{Q_B - Q_{SS}}{C_O} + 2\phi_{fn} \tag{2.19}$$

where Q_B = Change per unit area in the depletion layer below the oxide, Q_{SS} = Change density at substrate oxide-layer interface, C_O = Capacitance per unit gate area, ϕ_{ms} = Work function difference between gate and substrate, and ϕ_{fn} = Fermi level potential between invented surface and bulk si-substrate. The Q_B can be written as

$$Q_B = \sqrt{2\varepsilon_O \varepsilon_{si} QN(2\phi_{fn} + V_{SB})} \tag{2.20}$$

$$\phi_{fn} = \frac{KT}{Q} \ln\left(\frac{N}{n_i}\right) \tag{2.21}$$

where V_{SB} = Substrate bias voltage
Q = Charge of electron = 1.6×10^{-19} coulomb
N = Impurity concentration in the substrate
n_i = Intrinsic electron concentration
ε_{si} = Intrinsic electron concentration
K = Boltzman's constant
Q_S = (1.5 to 8) $\times 10^{-8}$ coulomb/m^2

2.5 □ SCALING OF MOS CIRCUITS

The scaling down of size of MOSFET leads to improved performance of VLSI design and higher packing density of circuit on a chip. VLSI fabrication technology should also be evaluated to increase packing density. VLSI fabrication technology may be characterized in terms of several figures of merit which are given below:

- Minimum feature size
- Number of gates on one chip
- Power dissipation
- Maximum operational frequency
- Die size
- Production cost

Many of these figures of merit can be improved by shrinking the dimensions of transistors, interconnections and the separation between features, and by adjusting a few doping levels and voltages. Over the past many years till date, much effort has been focused towards evolution of fabrication process technology and scaling down of the devices and feature size. So scaling is an important factor and it is essential for a VLSI designer to know scaling of MOS devices.

2.5.1 Scaling Factors

Figure 2.4 shows the device dimensions and substrate doping level which are associated with a scaling of the MOSFET transistors. There are two scaling factors $\frac{1}{\alpha}$ and $\frac{1}{\beta}$. The $\frac{1}{\beta}$ is used as a scaling factor for supply voltage V_{DD} and gate oxide thickness D whereas $\frac{1}{\alpha}$ is used for all other linear dimensions.

There are two models—constant-field model and constant-voltage model. In case of the **constant field model**, $\beta = \alpha$ whereas in case of **constant-voltage model**, $\beta = 1$. The following are scaling factors of device parameters which reveal the effects of scaling:

- **Gate Area A_g**

$$A_g = L.W$$

where L and W are channel length and width respectively. Both are scaled by $\frac{1}{\lambda}$. So A_g is scaled by $\frac{1}{\lambda^2}$

- **Gate Capacitance C_g**

$$C_{g=} C_{ox} L.W$$

where C_{ox} is oxide capacitance scaled by $\beta(=1/1/\beta)$ and C_g is scaled by β/α^2.

- **Parasitic Capacitance C_x**

C_x is proportional to $\frac{A_Y}{d}$

where d = Depletion width around source or drain which is scaled by $\frac{1}{\alpha}$ and A_x, area of depletion region around source or drain, scaled by $\frac{1}{\alpha^2}$. Thus, C_x is scaled by $\frac{1}{\alpha^2}.\frac{1}{1/\alpha} = \frac{1}{\alpha}$

- **Carrier Density in Channel Q_C**

$$Q_C = C_{OX} . V_{gs}$$

where Q_C = Average change unit area in channel. In the on state, C_O is scaled by β and V_{gs} is scaled by

$$\frac{1}{\beta}. \beta = 1$$

- **Channel Resistance R_C**

$$R_C = \frac{L}{W} \cdot \frac{1}{Q_C \cdot M}$$

where M = Carrier mobility is scaled by 1 and Q_c is scaled by 1. So R_C is scaled by

$$\frac{1}{\alpha} \cdot \frac{1}{1/\alpha} = 1$$

- **Gate Delay T_d**

$$T_d \, \alpha \, R_C . C_g$$

Thus, T_d is scaled by 1. $\frac{\beta}{\alpha^2} = \frac{\beta}{\alpha^2}$

- **Maximum Operating Frequency f_o**

$$f_o = \frac{W}{L} \cdot \frac{\mu C_{ox} V_{DD}}{C_g}$$

f_o is scaled by 1. $\frac{1}{\beta} .1 \frac{\beta}{\beta/\alpha^2} = \frac{\alpha^2}{\beta}$

- **Saturation current I_{dss}**

$$I_{dss} = \frac{C_{ox}^4}{2} \cdot \frac{W}{L} (V_{gs} - V_{THW})^2$$

where V_{gs} and V_{THN} are scaled by $\frac{1}{\beta}$ and I_{dss} is scaled by $\beta.1.1.1.\left(\frac{1}{\beta}\right)^2 = \frac{1}{\beta}$

- **Current Density J**

$$J = \frac{I_{dss}}{A}$$

where A = Cross-sectional area of channel scaled by $\frac{1}{\alpha^2}$. Thus, J is scaled by

$$\frac{1}{\beta} \cdot \Big/ \frac{1}{\alpha^2} = \frac{\alpha^2}{\beta}$$

- **Switching Energy E_g**

$$E_g = 1.\frac{C_g}{2} (V_{DD})^2$$

Thus, E_g is scaled by $\frac{\beta}{\alpha^2} \cdot \frac{1}{\beta^2} = \frac{1}{\alpha^2 \beta}$

- **Power Dissipation Per Gate, P_g**

$$P_g = P_{gs} + P_{gd}$$

where P_{gs} Power dissipation between source and gate and P_{gd} = Power between drain and gate.

$$P_{gd} = \frac{(V_{DD})^2}{R_C}$$

$$P_{gd} = E_g \cdot f_o$$

So P_{gs} and P_{gd} are scaled by $\frac{1}{\beta^2}$, and so P_g is scaled by $\frac{1}{\beta^2}$

- **Power Dissipation Per Unit Area P_a**

$$P_a = \frac{P_g}{A_g}$$

So P_a is scaled as $\frac{1/\beta^2}{1/\alpha^2} = \frac{\alpha^2}{\beta^2}$

• Power Speed Product P_T

$$P_T = P_g \,.\, T_d$$

So P_T is scaled as $\frac{1}{\beta^2} \cdot \frac{\beta}{\alpha^2} = \frac{1}{\alpha^2 \beta}$

Table 2.1 shows the scaling factors for constant electric tied model ($\beta = \alpha$), constant-voltage model, and constant V and D model.

• Substrate Doping Scaling Factors The built-in potential V_B depends on substrate doping. We can neglect V_B as V_B is smaller than V_{DD}. Due to substrate doping, V_B should be needed. As channel length of MOS transistors is reduced, the depletion-region widths must also be scaled down to prevent the source and drain depletion-region. Depletion-region width for the junctions are given by

$$d = \sqrt{\frac{2\varepsilon_{si}\varepsilon_o V}{qN_B}} \tag{2.22}$$

where ε_{si} = Relative permittivity of silica
ε_o = Permittivity of free space
V = Effective voltage across junction = $V_a + V_B$
Q = Electron charge
N_B = Doping concentration of p-substrate
V_a = Applied voltage
V_B = Built-in potential

And V_B is written as

$$V_B = \frac{KT}{q} C_n \left(\frac{N_B N_D}{n_i} \right) \tag{2.23}$$

where N_D is source or drain doping concentration and n_{ivi} intrinsic carrier concentration in silicon. If V_B is neglected and $V_a = V_{DD}$ then $d = \sqrt{\frac{2\varepsilon_{si}\varepsilon V_{DD}}{qN_B}}$.

As V_{DD} is scaled as $\frac{1}{\beta}$ and d by $\frac{1}{\alpha}$, hence, N_B can be scaled as $\frac{\alpha^2}{\beta}$. The carrier $V_a = mV_B$, where m = Real number.

$$V = mV_B + V_B = (1 + m)\, V_B$$

If V_a is scaled as $\frac{1}{\beta}$, then

$$V_S = \frac{mV_\beta}{\beta} + V_\beta$$

Scaling factor = $\frac{\beta + m}{\beta(m+1)}$

So N_B can be scaled as $\frac{\alpha^2(\beta + m)}{m+1}$

• **Depletion Width** When N_B is increased by α and if $V_a = 0$ then V_B is increased by α and d is decreased by

$$\sqrt{\frac{l_n(\alpha)}{\alpha}}$$

The depletion width is a function of substrate concentration N_B and supply voltage V_{DD}. The maximum depletion width is obtained at $E_{max} = \frac{2V}{d}$ where $V = \frac{E_{max} d}{\alpha}$. So we can write

$$d = \sqrt{\frac{2\varepsilon_{si}\varepsilon_o}{q_{NB}}} \cdot \frac{E_{max}}{2}.$$

Hence,

$$d = \frac{\varepsilon_{si}\varepsilon_o}{qN_B}.E_{max}$$

• **Limitations** Scaling down has many associated effects which cause problems or limitations in miniaturization of interconnects, contact resistance of logic level, and supply voltage due to noise.

2.5.2 Limits of Miniaturization

The minimum size of a device is determined mainly by process technology and theory of the device. The miniaturization of device size depends on alignment accuracy and resolution of photolithography technology with mask. Using photolithography technology, minimum size is obtained within 3 μm (submicron) but with availability of write E-beam technology, this limit can be further reduced to nano level. The size is usually estimated in terms of channel length L which must be minimum of 2 d (where d = Depletion width). The transit time is written as

$$\tau = \frac{L}{v_{drift}} = \frac{2d}{\mu E} \tag{2.24}$$

where V_{drift} = Drift velocity = μE, E = Electric field.

2.5.3 Limits of Interconnects and Contact Resistance

The width and spacing of interconnects are scaled by $\frac{1}{\alpha}$ and area must be scaled as $\frac{1}{\alpha^2}$. For short-distance interconnections, conductor length is scaled by $\frac{1}{\alpha}$. So the resistance is scaled as α. With reduction of device size, the integration scale or length of integration increases and thus results in lengthening of interconnections and increase in number of interconnections. Therefore, there is change in resistance and parasitic capacitance time constant, propagation delays, etc. The propagation delay is written as

$$T_P = R_{int}\, C_{int} + 2.3\,(R_{on}\, C_{int} + R_{on}\, C_L + R_{int}\, C_L)$$

where R_{int} = Resistance of the interconnect which is written as

$R_{int} = \frac{PL_C}{HW}$, where P = Resistivity, L_c = Length of interconnection, and C_{mi} = Capacitance of interconnect.

$$C_{mi} = \varepsilon_{ox}\left[1.15 \,{}^{w}\!/_{t_{ox}} + \frac{2.L8H^{0.222}}{t_{ox}}\right]L \tag{2.25}$$

where ε_{ox} = permittivity of oxide layer

2.5.4 Limit due to Subthreshold Current

One of the major problems in scaling of the devices is sub threshold current I_{rub} which is propotional to exp $\{[V_{gs} - V_{THN}]\, Q/KT\}$. When the transistor is in off state, then $V_{gs} - V_{THN}$ is – *ve* and should be as large as possible to minimize I_{sub} along with V_{DD} and to increase $V_{gs} - V_{th}$ magnitude. Limit is required to control breakdown voltage which is written as

$$V_{Break} = \frac{\varepsilon_{si}\varepsilon_{ox}}{2qN_B}(E_{cnH})^2 \tag{2.26}$$

where ε_{si} = permittivity of silicon layer

The breakdown voltage is scaled as $\beta(m + 1)/\alpha^2\,(\beta + m)$. As gate-to-source electric field is greater, breakdown voltage is greater.

2.5.5 Limits on Logic Levels and Supply Voltage due to Noise

Scaling of the devices depends on operating frequencies in which smaller the gate delay, higher the operation frequency. It remits to lower power dissipation. For smaller device size, greater switching speeds cause noise problems. Their mean current fluctuation in the channel due to noise is given by

$$(2^2) = 4\, KT\, R_n\, g_m\, \Delta t$$

where R_n = Equivalent noise resistance at the input and Δf = Bandwidth. The noise resistance

$$R_n = \frac{1}{g_m}\left[\frac{1}{2}\cdot\frac{V_g'}{V_P'} + \frac{1}{6}\right]^1 - 1.$$

where $V'_g = V_{gs} - V_{THN} + V_B$, V_B = Junction built-in potential $V'_p = V_p + V_B$ V_P = Pinch-off voltage so the current fluctuation (2^2) is written as

$(2^2) = 2^{KT.}\left[\dfrac{V_{gs} - V_{THN} + V_B}{V_P + V_B} + \dfrac{1}{6}\right]$ of the equivalent noise voltage which can be written as]

$$\Delta V = \sqrt{\frac{1}{2}\frac{qs(V_{gs} - V_{THN})}{C_g \cdot f}} \tag{2.27}$$

where f = Operating frequency, C_g = Gate capacitance, $S = \dfrac{dn_t}{d_n}$ = Surface-state efficiency, dn_t = Change of number of tapped carriers which depends on number of reduced free carriers d_n.

2.6 ☐ DESIGN PROCESS OF MOSFET-BASED DEVICES

Design process consists of design rules to present actual in put into layout diagram. It establishes a communication link between the designer specifying requirements and the fabricator who materializes them. The design rules are used to make a workable mask layout through which various layers on silicon are firmed as per device requirements. Design processes, stick diagram, and symbolic diagrams are key elements to form layout mask.

2.6.1 MOS Layers

MOS design process converts MOSFET based circuits into masks for fabrication of circuits in IC form to meet specifications. There are four basic layers of MOSFET—*n*-diffusion, *p*-diffusion, polysilicon, and metal—which are isolated from one another by thick or thin silicon dioxide to make insulation. The masks have *n*-diffusion, *P*-diffusion, polysilicon layer and oxide insulation layers. Polysilicon and thinox regions cross one another to form the transistors. In some processes, there may be a sound metal layer and a second polysilicon layer which are joined together to form the antennas. For depletion mode *n*-MOS transistors, the implant within form oxide layers are used. For CMOS, bipolar transistors are included in addition to CMOS process.

2.6.2 Stick Diagram

A stick diagram represents layer information and topology. These diagrams are evaluated from circuits, and mask layouts are easily turned from these diagrams. There are colour-code schemes and monochrome stick-diagram codes for the layers.

Symbolic diagrams are another convenient way to represent *n*MOS *p*-MOS, CMOS, or BiCMOS-based circuits. Table 2.1 shows a stick diagram and symbolic representation of different layers of *n*MOS devices. The transistor stick diagrams are given more stress for ready translation into mask-layout forms. All features and layers defined in tables 2.1, 2.2 and 2.3—with the exception of implant (yellow) and the buried contact (brown)—which are used in CMOS design. Yellow in CMOS design is now used to identify *p*-transistors and wires, as depletion-mode devices are not utilized. As a result,

no confusion results from the allocation of the same color to two different features. The two types of transistors used—'*n*' and '*p*' are separated in the stick layout by the demarcation line (representing the *p*-well boundary) above which all *p*-type devices are placed (transistors and wires (yellow)). The *n*-devices (green) are consequently placed below the demarcation line and are thus, are located in the *p*-well.

Diffusion paths must not cross the demarcation line and *n*-diffusion and *p*-diffusion wires must not join. The '*n*' and '*p*' features are normally joined by metal where a connection is needed. Apart from the demarcation line, there is no indication of the actual *p*-well topology at this (stick diagram) level of abstraction; neither does the *p*+ mask appear. Their geometry will appear when the stick diagram is translated to a mask layout. However, we must not forget to place crosses on V_{DD} and V_{SS} rails to represent the substrate and *p*-well connection respectively. The design style is illustrated simply by taking as an example the design of a single bit of a shift register. The design begins with the drawing of the V_{DD} and V_{SS} rails in parallel and in metal, and the creation of an (imaginary) demarcation line.

Table 2.1 Encoding of stick diagram for *n*-MOS processing

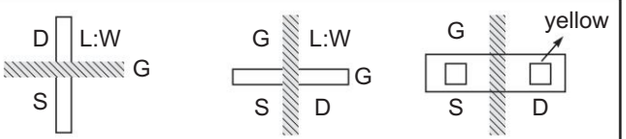
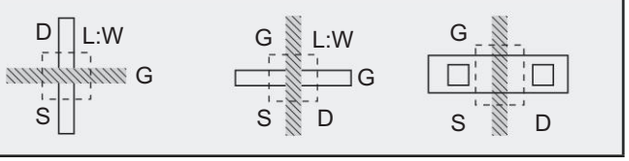
Layers	Color	Stick diagram	Symbolic diagram
n+ diffusion	Green		
Polysilicon	Red		
Metal 1	Blue		
Contact cut	Black		
n-type enhancement-mode MOS	----		
n-type depletion-mode MOS			

Table 2.2 shows stick diagram and symbolic diagram for *p*-MOS processing.

Table 2.2 Encoding of stick diagram for *p*-MOS processing

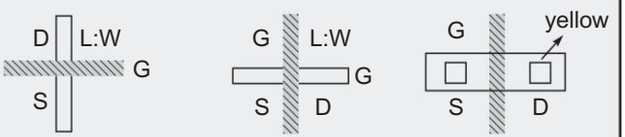
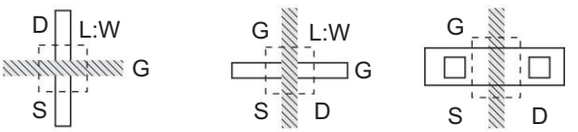
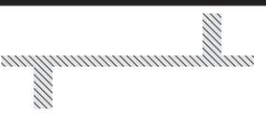
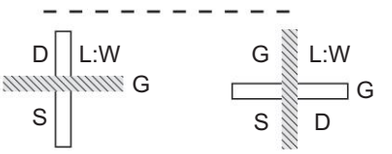
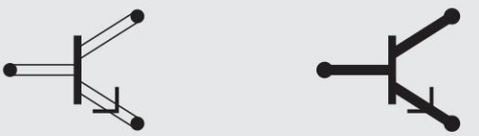
Layers	Color	Stick diagram	Symbolic diagram
p+ diffusion	Yellow		
Polysilicon	Same as in *n*-processing		
Metal 2	Dark blue		
Via	Black		
V_{DD} or V_{SS} contact	Black		
p-type enhancement-mode MOS	----		
p-type depletion-mode MOS			

Table 2.3 shows additional encoding of stick and symbolic representation.

Table 2.3 Addition encoding of stick diagram for CMOS and BiCMOS processing

Layers	Color	Stick diagram	Symbolic diagram
Polysilicon-2	Orange		
*n*MOS is below the line			
Demarcation line in which	Brown		

*n*MOS is above the line		
Demarcation line in which	Brown	
Bipolar *npn*	----	
Bipolar *pnp*	----	

2.6.3 Examples of Stick and Symbolic Diagrams

Stick and corresponding symbolic diagrams of *n*MOS inverter are shown in Fig. 2.4.

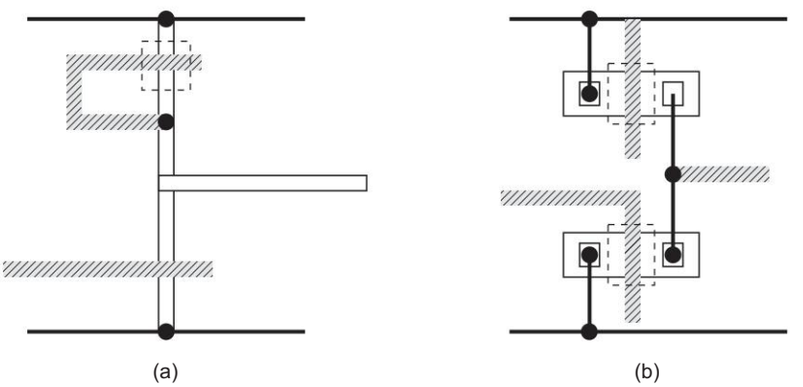


Fig. 2.4 *n*MOS inverter: (a) Stick diagram (b) Symbolic diagram

Stick and symbolic diagram of a CMOS inverter are shown in Fig. 2.5 and circuit diagram and description of CMOS inverter is given in Chapter 3 (Fig. 3.3).

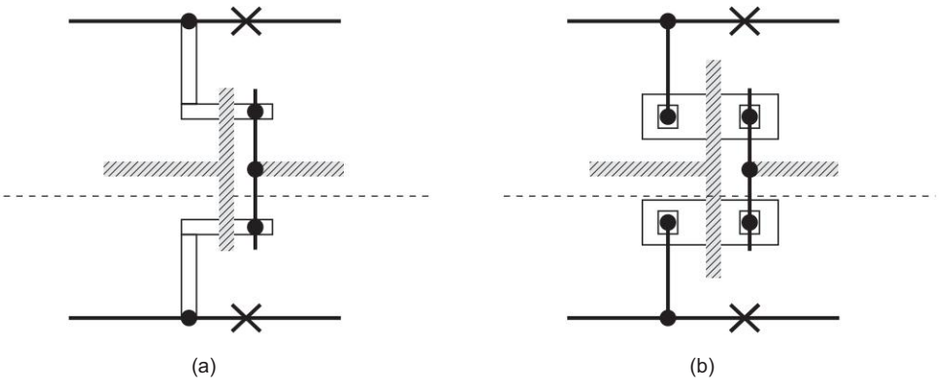


Fig. 2.5 CMOS inverter: (a) Stick diagram (b) Symbolic diagram

Stick and symbolic diagram of a BiCMOS inverter are shown in Fig. 2.6.

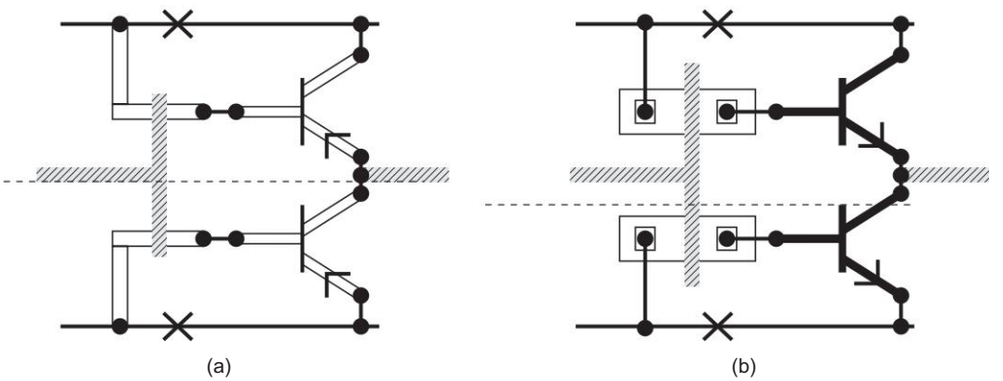


Fig. 2.6 BiCMOS: (a) Stick diagram (b) Symbolic diagram

2.6.4 *n*MOS Design Style

A normal approach to stick-diagram layout is used mostly for MOS-based circuits because both are easy to use and to turn into a mask layout.

The layout of *n*MOS involves

- *n*-diffusion [*n*-diff] and other thinoxide regions [thinox](green)
- Polysilicon 1 [poly]—since there is only one polysilicon layer here (red)
- Metal 1 [metal]—since there is only one metal layer here (blue)
- Implant (yellow)
- Contacts (black or brown [buried])

A transistor is formed wherever poly crosses *n*-diff (red over green) and all diffusion wires (interconnections) are *n*-type (green).

When starting a layout, the first step normally taken is to draw the metal rail (blue) for V_{DD} and GND in parallel allowing enough space between them for the other circuit elements. Next, thinox (green) paths may be drawn between the rails for inverters and inverter based logic as shown in Fig. 2.7(a), and also contacts are made. Inverter and inverter based logic comprise a pull-up structure, usually a depletion

mode transistor, connected from the output point to VDD and the pull-down structure of enhancement mode transistors suitably interconnected between the output point and GND. This step in the process is illustrated in Fig. 2.7. The polysilicon lines (red) crosses thinox (green) wherever transistors are required. The implants (yellow) for the depletion-mode transistor and the length-to-width (L:W) ratio is also written for each transistor. Ratios are required particularly in *n*MOS and *p*MOS circuits.

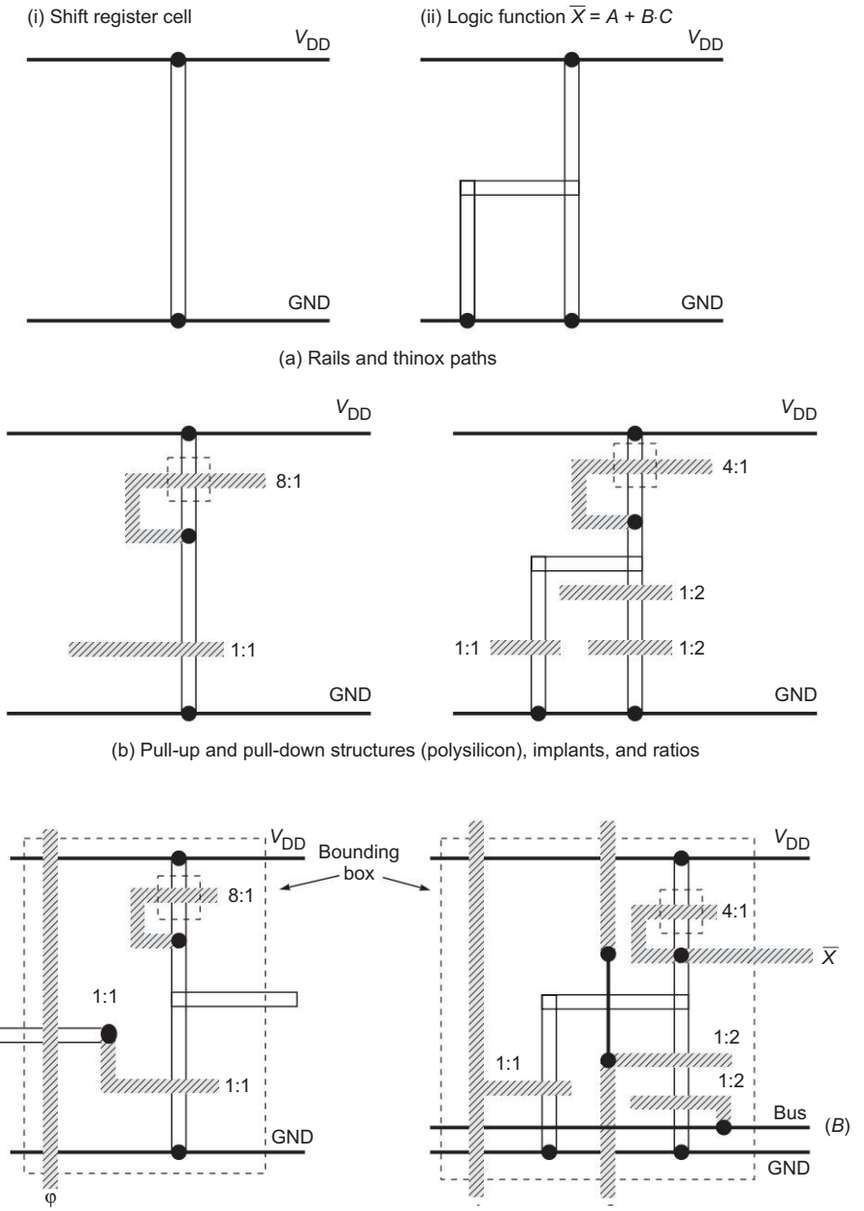


Fig. 2.7 *n*MOS stick-diagram steps

Signal path-bypass transistors and long signal paths require metal buses (blue). A convenient strategy is to run power rails and buses in parallel in metal (blue) and then propagate control signals at right angles on poly as shown in Fig. 2.7(c).

2.6.5 CMOS Design Style

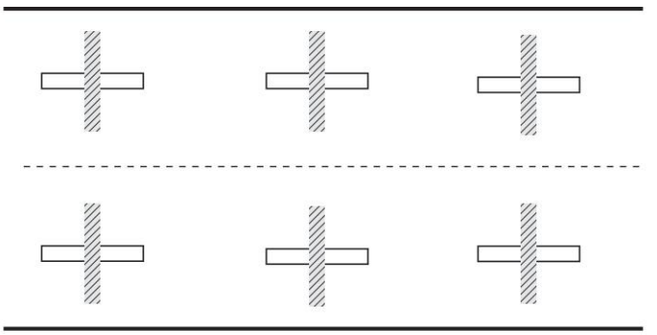
The stick and layout representation for CMOS is an extension of the *n*MOS approach and style already outlined. All features and layers defined in Table 2.1–2.3 with the exception of implant (yellow) and the buried contact (brown)—are used in CMOS design. Yellow in CMOS design is now used to identify *p*-transistors and wires, as depletion-mode devices are not utilized. As a result, no confusion results from the allocation of the same color to two different features. As mentioned earlier, *n*MOS and *p*MOS are separated in the stick layout by the demarcation line (representing the *p*-well boundary) above which all *p*-type devices are placed (transistors and wires (yellow)). The *n*-devices (green) are consequently placed below the demarcation line and are thus located in the *p*-well as shown in Table 2.3. Figure 2.8 shows the steps used for making stick diagrams of a single-bit CMOS shift register. The demarcation line for *p*MOS and *n*MOS used in the circuit is shown in Fig. 2.8(a). In the figure, diffusion paths do not cross the demarcation line and *n*-diffusion and *p*-diffusion wires must not join. The '*n*' and '*p*' features are normally joined by metal where a connection is needed. Apart from the demarcation line, there is no indication of the actual *p*-well topology at this (stick diagram) level of abstraction; neither does the *p*+ mask appear. Their geometry will appear when the stick diagram is translated to a mask layout. The design begins with the drawing of the V_{DD} and V_{SS} rails in parallel and in metal and the creation of an (imaginary) demarcation line in between, as in Fig. 2.8(a). The *n*-transistors are then placed below this line and thus close to V_{SS}, while *p*-transistors are placed above the line and below V_{DD}. A similar approach can be followed with transistors in symbolic form.

The interconnection of *p*MOS and *n*MOS as required, using metal and contact to the rails are shown in Fig. 2.8(b). In the figure, only metal and polysilicon can cross the demarcation line with the above restriction. Finally, the remaining interconnections are made as appropriate and the control signals and data inputs are added as illustrated in Fig. 2.8(d). The indications of V_{DD} and V_{SS} have to be given in a stick diagram. These stick diagrams are converted into mask layouts where all green features belong to *n*MOS and yellow features belong to *p*MOS.

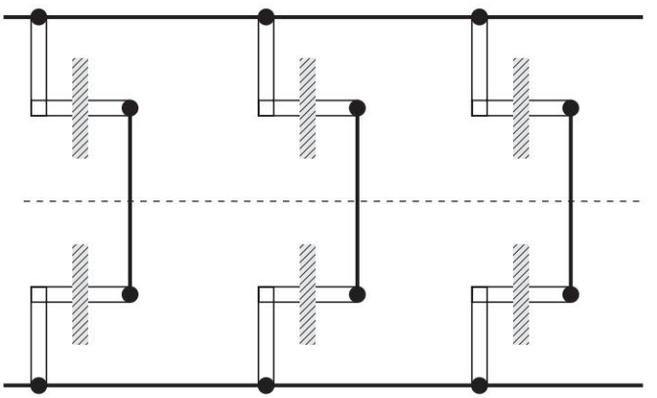
An even simpler representation, which nevertheless carries much of the information present in a stick diagram, is to draw a symbolic diagram as in Fig. 2.9. This diagram represents the same circuit as Fig. 2.8(c). This form of diagram facilitates transistor merging, as shown, and is also readily translated to mask layouts. Demarcation line may be shown but is not essential since transistor symbols are already encoded.

2.7 □ DESIGN RULES FOR LAYOUT

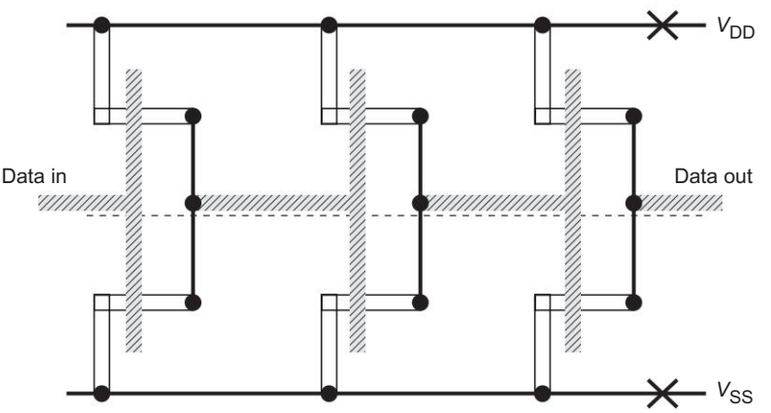
The design rules are required for a ready translation of circuit concepts, usually in stick diagram or symbolic form, into actual mask layout in silicon. The design rules usually provide workable and reliable layouts. Circuit designers have tighter requirements, smaller layouts for improved performance and decreased silicon area, whereas the process engineer wants design rules that result in a controllable and reproducible process. So there has to be a compromise for a competitive circuit to be produced at a reasonable cost. Design-rule definitions are determined by process-line equipment and process design.



(a) *n*MOS and *p*MOS with demarcation line



(b) Metal and diffusion connections



(c) Remaining connections

Fig. 2.8 Steps for making stick diagram for CMOS-based circuits

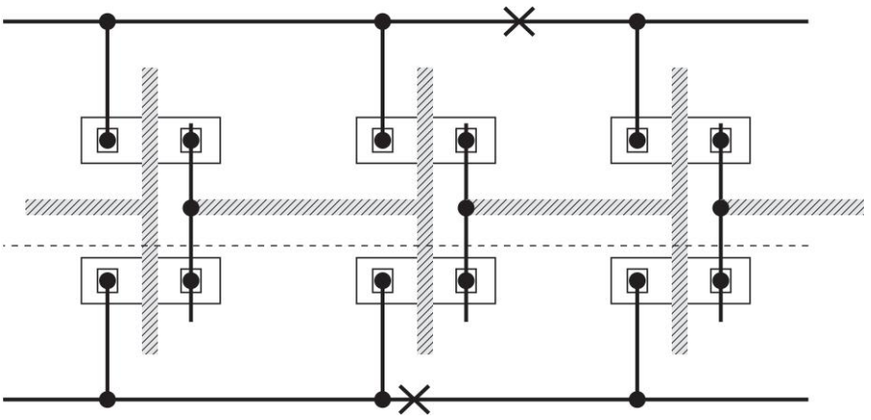


Fig. 2.9 Symbolic representation of Fig. 2.8(c)

For example, if a 10:1 wafer stepper is used instead of a 1:1 projection mark aligner, the level-to-level registration will be closer. Design rules can be affected by the maturity of the process line.

The simpler 'lambda (λ) based design rules have been widely used, particularly in the educational context and in the design of multiproject chips. The design rules are based on a single parameter λ which leads to a simple set of rules for the designer. The simplicity of lambda-based rules also provides a simple mask-layout design in general for the 'micron-based' rule sets which follow.

2.7.1 Lambda-based Design Rules

The design rules and layout methodology based on the concept of λ provide a process and feature-size-independent way of setting out mask dimensions to scale. All paths in all layers will be dimensioned in λ units and subsequently, λ can be allocated an appropriate value compatible with the feature size of the fabrication process. The actual mask-layout design takes little account of the value subsequently allocated to the feature size. For example, λ can be allocated a value of 1.0 μm so that minimum feature size on chip will be 2 μm (2 λ). Design rules also specify line-width separations, and extensions in terms of λ. Design rules can be conveniently set out in diagrammatic form as in Fig. 2.8 for the widths and separation of conducting paths, and in Fig. 2.10 for extensions and separations associated with *n*MOS and *p*MOS transistor layouts.

The design rules associated with contacts between layers are set out in Fig. 2.11 and it will be noted that connection can be made between two or, in the case of *n*MOS designers, three layers. When making contacts between polysilicon and diffusion in *n*MOS circuits, it should be recognized that there are three possible approaches—polysilicon to metal, metal to diffusion, and buried contact polysilicon to diffusion or butting contacts which are widely used. In CMOS designs, polysilicon to diffusion contacts are made via metal. When making connections between metal and either of the other two layers, the process is quite simple. The $2\ \lambda \times 2\ \lambda$ contact cut indicates an area in which the oxide is to be removed down to the underlying polysilicon or diffusion surface. When connecting diffusion to polysilicon using the butting-contact approach (Fig. 2.11), the process is rather more complex. In effect, a $2\ \lambda \times 2\ \lambda$ contact cut is made down to each of the layers to be joined. Since the polysilicon and diffusion outlines overlap and the thin oxide under polysilicon acts as a mask in the diffusion process, the polysilicon

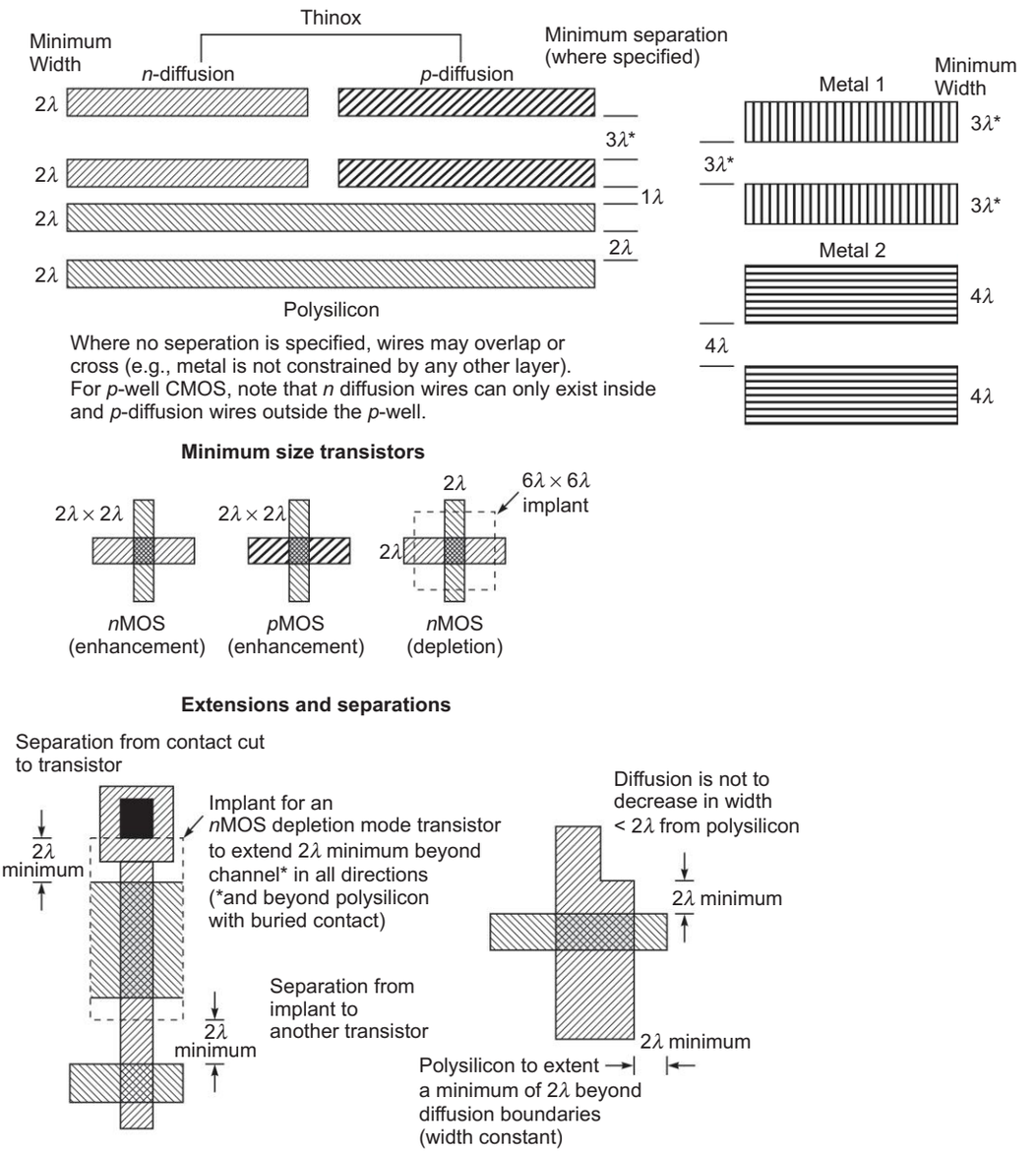


Fig. 2.10 Representation of different layers and MOS layout by using lambda-based design rules

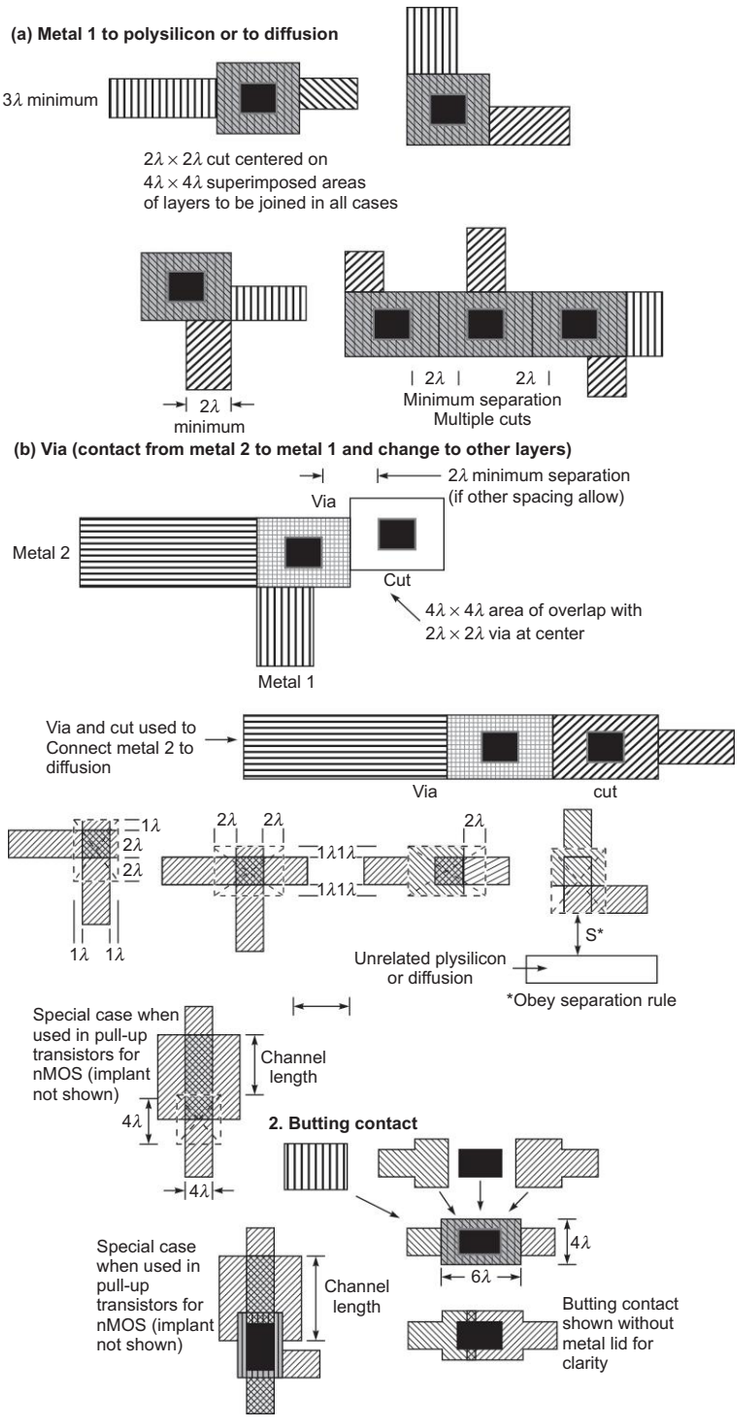


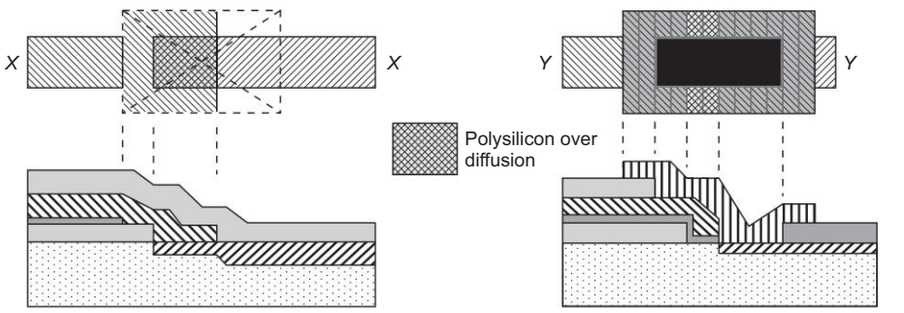
Fig. 2.11 Representation of contacts used in layout of MOS and CMOS-based circuits

and diffusion layers are also butted together. The contact between the two butting layers is then made by a metal overlay as shown in Fig 2.11(b). In buried contact, basically, layers are joined over a 2 $\lambda \times$ 2 λ area with the buried cut extending by 1 λ in all directions around the contact area except that the contact-cut extension is increased to 2 λ in diffusion paths leaving the contact area. This is to avoid forming unwanted transistors (see following examples). The buried-contact approach shown in Fig. 2.9 and 2.10 is simpler, the contact cut (broken line) in this case indicates where the thin oxide is to be removed to reveal the surface of the silicon wafer before polysilicon is deposited. Thus, the polysilicon is deposited directly on the underlying crystalline wafer. When diffusion takes place, impurities will diffuse into the polysilicon as well as into the diffusion region within the contact area. Thus, a satisfactory connection between polysilicon and diffusion is ensured. Buried contacts can be smaller in area than their butting-contact counterparts and since they use no metal layer, they are subject to fewer design-rule restrictions in a layout.

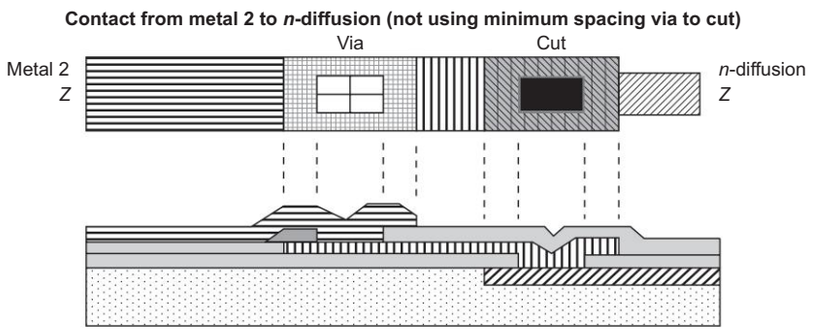
The design rules ensure that no transistor is formed unintentionally in series with the contact, need to avoid the formation of unwanted diffusion to polysilicon contacts, and protect the gate oxide of any transistor in the vicinity of the buried contact-cut area.

2.7.2 Double Metal MOS Process Rules

From the overall chip-interconnection aspect, the second metal layer, in particular, is important because of connection to other layers using metal 1 to metal 2 contact called **via** which can be established as shown in Fig. 2.12(c).



(a) Buried contact...section through *XX* (b) Butting contact...section through *YY*



(c) Metal 2–via-metal 1–cut-*n*-diffusion connection...section through *ZZ*

Fig. 2.12 Cross sections of contacts for double-metal process

Usually, second-level metal layers are coarser than the first (conventional) layer and the isolation layer between the layers may also be of relatively greater thickness, to distinguish contacts between first and second metal layers. They are known as vias rather than contact cuts. The second-metal-layer representation is color-coded dark blue (or purple). The oxide below the first metal layer is deposited by Chemical Vapor Deposition (CVD) and the oxide layer between the metal layers is applied in a similar manner. Depending on the process, removal of the selected areas of the oxide is accomplished by plasma etching, which is designed to have a high level of vertical ion bombardment to allow for high uniform etch rates. A second thin oxide layer is grown after depending and patterning the first polysilicon layer (poly 1) to isolate it from the now-to-be-deposited second poly layer (poly 2). The presence of the second poly layer gives greater flexibility in interconnections and also allows poly 2 transistors to be formed by intersecting poly 2 and diffusion. For the double-metal process, the following steps are used shortly:

1. Use the second-level metal for the global distribution of power buses, i.e., V_{DD} and GND (V_{SS}), and for clock lines.
2. Use the first-level metal for local distribution of power and for signal lines.
3. Layout the two metal layers so that the conductors are mutually orthogonal whenever possible.

2.7.3 CMOS Lambda-based Design Rules

The CMOS fabrication process is much more complex than *n*MOS fabrication. Figure 2.13 shows CMOS design rules. However, the Mead and Conway concepts for *n*MOS design rules are extended for CMOS design rules with the exclusion of butting and buried contacts. The additional rules are concerned with those features unique to *p*-well CMOS, such as the *p*-well and *p*+ mask and the special 'substrate' contacts. The rules given are also readily translated to an *n*-well process.

2.7.4 Special Lambda Rules of Bi-CMOS

Apart from CMOS lambda rules, additional rules are included for representation of Bipolar Junction Transistor (BJT). Figure 2.14 shows BJT layout in which BCCD underlines the entire area and the *p*-base underlines all within its boundary.

1. Comments on Lambda-Based Design Rules

For the lambda-based rules discussed initially, the design rules are formulated in terms of a length unit λ which is related to the resolution of the process. λ may be viewed as a bound on the width deviation of a feature from its ideal 'as drawn' size also as a bound on the maximum mis-alignment of any one mask. In the worst case, these effects may combine to cause the relative position of feature edges on different mask levels to deviate by as much as 2 λ in their interrelationship. Inevitably, a consequence of using the lambda-based concept is that every dimension must be rounded up to whole λ values and this leads to layouts which do not fully exploit the capabilities of the process.

Similar concepts underline the establishment of 'micron-based' rule sets, but actual dimensions are given so that full advantage can be taken of the fabrication-line capabilities and tighter layouts result.

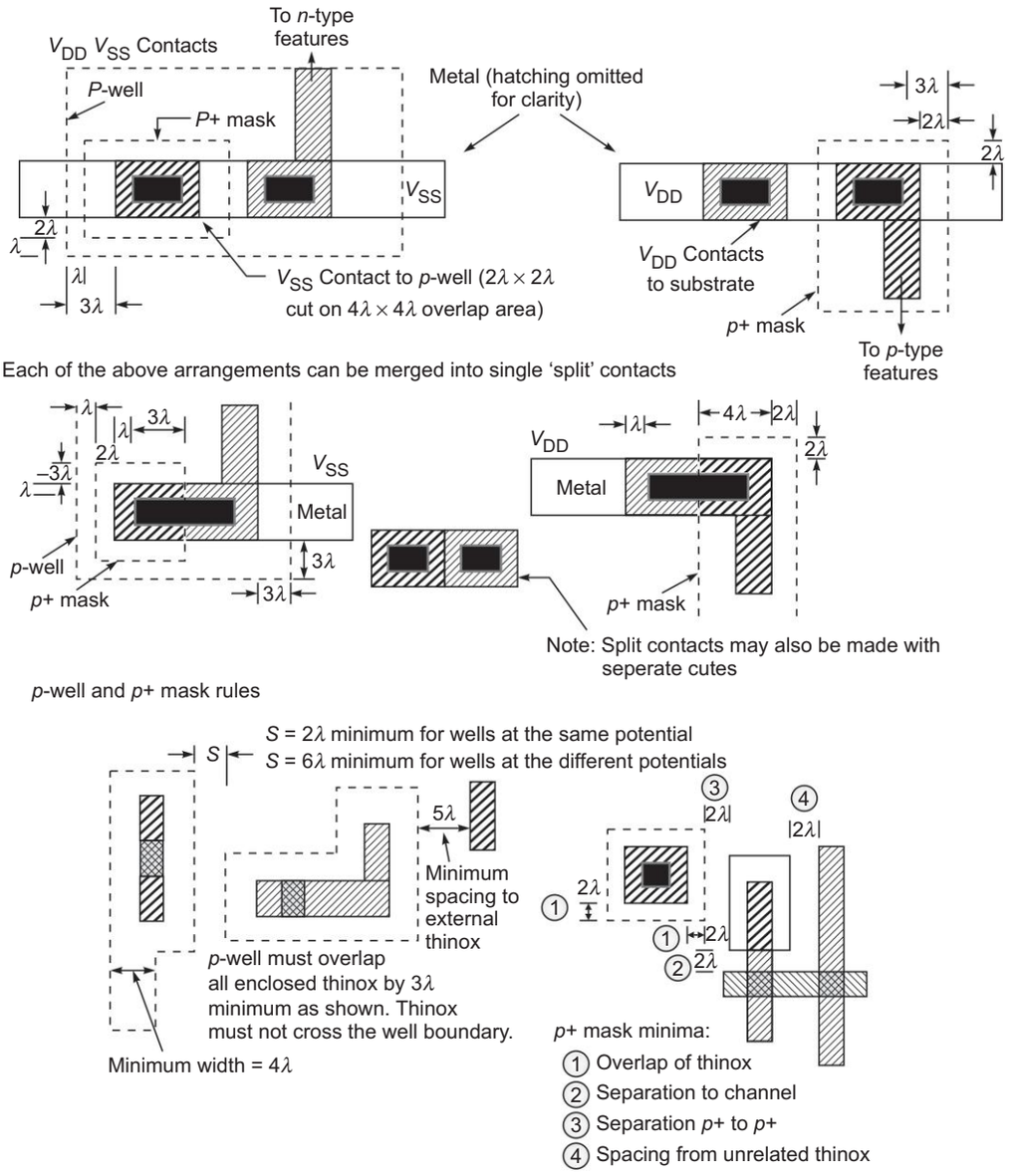


Fig. 2.13 *p*-well CMOS design rules

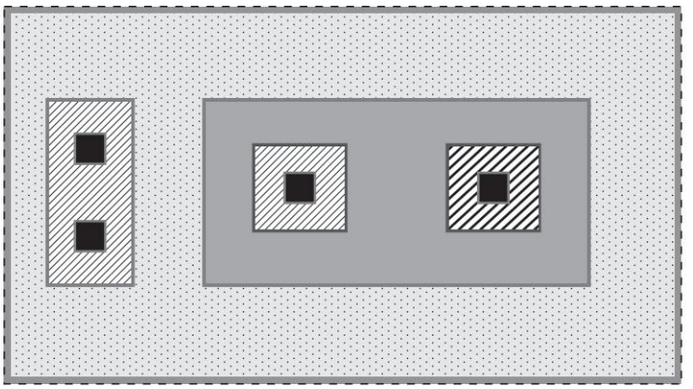


Fig. 2.14 Lambda-rule-based BJT layout

2.8 □ TRANSLATION OF STICK DIAGRAM TO LAMBDA-BASED LAYOUT

As discussed earlier, the stick diagram is the middle step to make a lambda-based mask layout. Any CMOS or MOS-based circuits first can be converted into stick diagrams and then the stick diagrams can be easily converted into a mask layout. Figure 2.15 shows conversion of a stick diagram of a MOS shift-register cell into a layout mask.

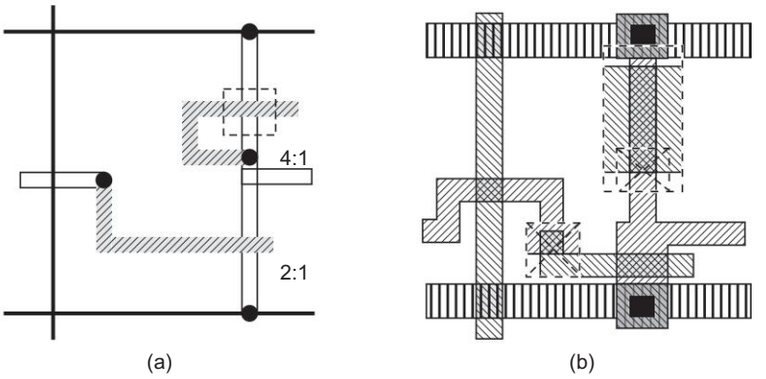


Fig. 2.15 (a) Stick diagram (b) Corresponding layout mask

2.9 □ TRANSLATION OF SYMBOLIC DIAGRAM INTO LAMBDA-BASED LAYOUT

Like the stick diagram, the symbolic diagram is used to make lambda-based mask layout. Any CMOS or MOS-based circuits first can be converted into symbolic diagrams and then the symbolic diagrams can be easily converted into a mask layout. Figure 2.16 shows conversion of a symbolic diagram of a 1-bit CMOS shift-register cell into a layout mask.

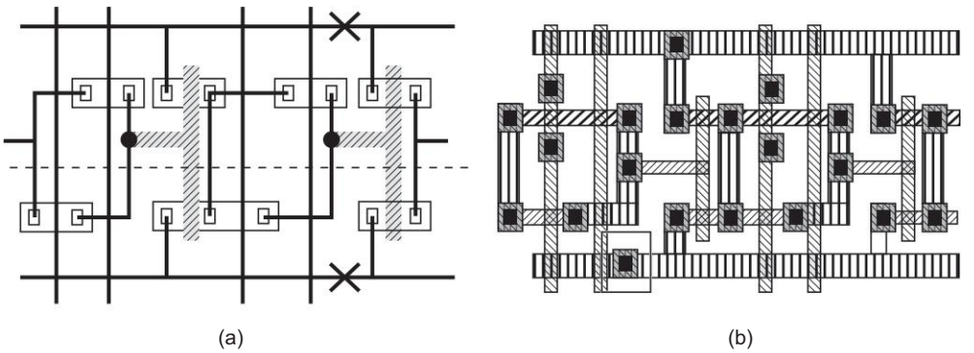


Fig. 2.16 (a) Symbolic diagram (b) Corresponding layout mask

Figure 2.17 shows translation of a symbolic diagram of BiCMOS-based two-input NAND gates into a mask layout. The circuit diagram and description of BiCMOS-based NAND gates are given in Chapter 6.

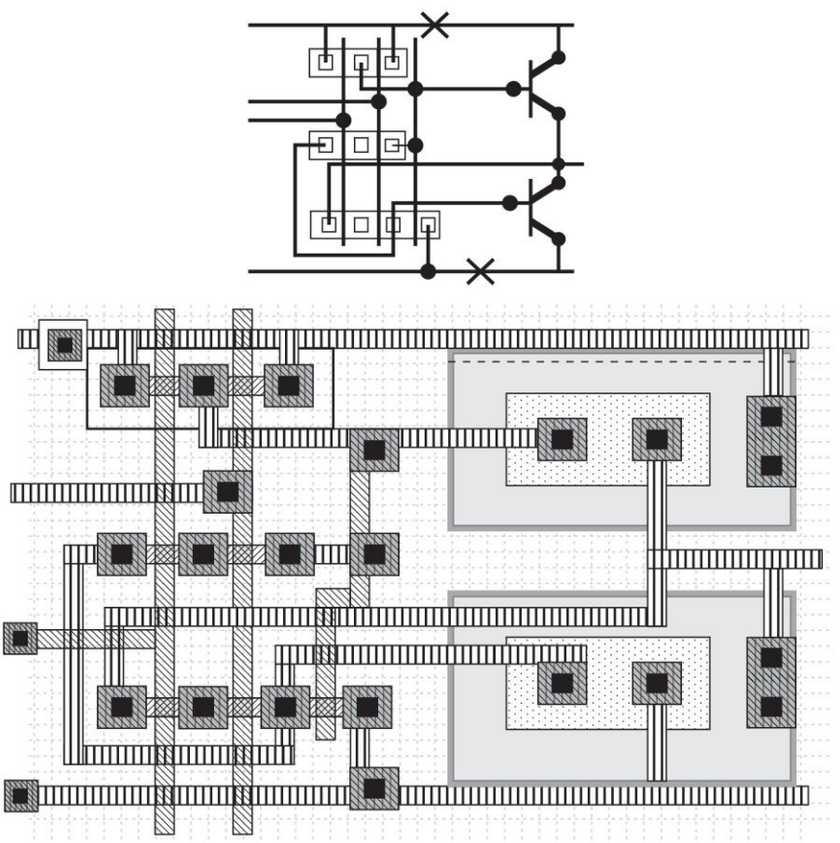


Fig. 2.17 Translation of symbolic diagram of BiCMOS-based two-input NAND gate into layout mask

2.10 ☐ LAYOUT OF RESISTANCE AND CAPACITANCE

Sheet resistance (discussed in Chapter 4) concept is applied to MOS transistors. Figure 2.18 shows thinox mask layout which is the union of diffusion and channel regions. This thinox acts as sheet resistance which is written as

$$R = Z \,.\, R_S$$

where $Z = L/W$ and R_S = Sheet resistance in ohms/square

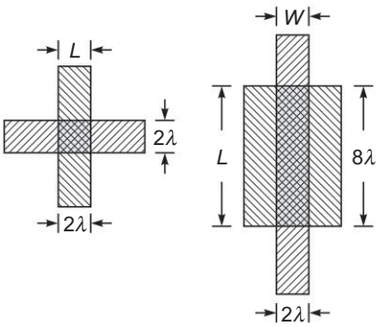


Fig. 2.18 Layout of transistor channels as resistance

The capacitance is also formed by using a multilayer concept. The capacitance is determined from the following formula

$$C = \text{Relative area} \times \text{relative } C \text{ value} = \text{Relative area} \times C_r$$

From Fig. 2.19, metal capacitance is estimated as

$$\text{Relative metal area} = \frac{100\lambda \times 3\lambda}{4\lambda^2} = 75$$

$$\text{Metal capacitance} = C_m = 75\ C_r$$

$$\text{Relative polysilicon area} = \frac{4\lambda \times 4\lambda + 3\lambda \times 2\lambda}{4\lambda^2} = 5.5$$

$$\text{Polysilicon capacitance} = C_p = 5.5\ C_r$$

$$\text{Gate capacitance} = C_r$$

$$\text{Total capacitance} = C = C_m + C_p + C_r = 75\ C_r + 5.5\ C_r + C_r = 81.5\ C_r$$

For 2 μ MOS technology, relative capacitance $C_r = 0.00024$ pF. Total capacitance C is determined as $C = 0.01956$ pF.

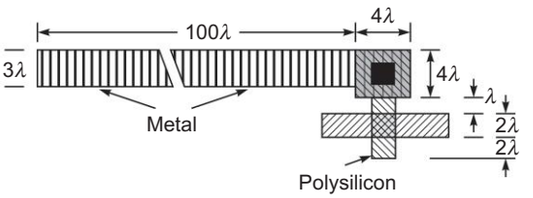


Fig. 2.19 Layout of multilayer-based capacitance

2.11 ¤ MORE EXAMPLES OF MASK LAYOUT

Figure 2.20 shows layout of an *n*MOS based three-inputs NOR gate having implant and buried contact and its corresponding stick diagram.

Figure 2.21 shows translation of a stick diagram into layout for a CMOS-based 4:1 multiplexer.

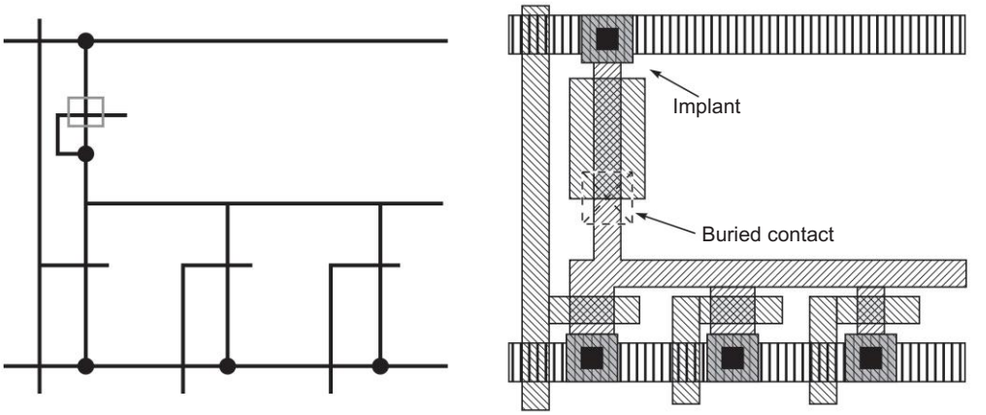


Fig. 2.20 *n*MOS based three-input NOR gate layout and stick diagram

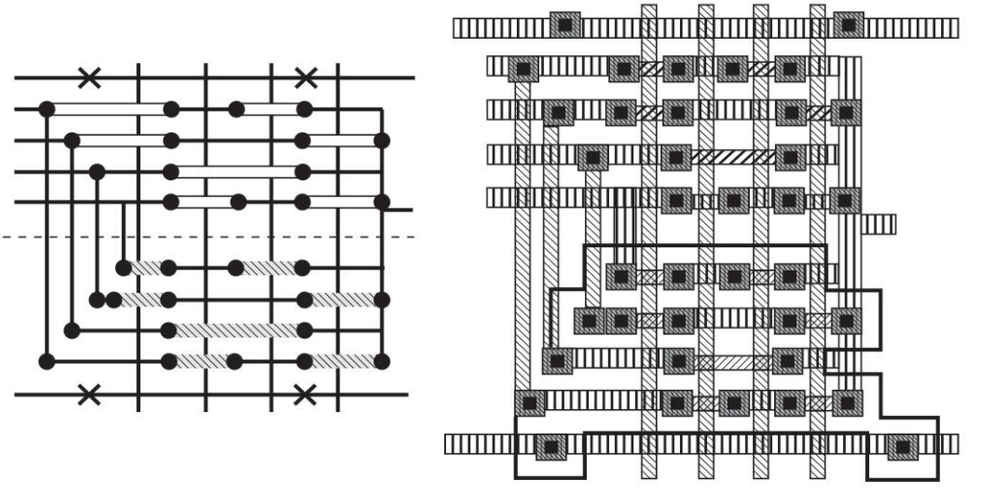


Fig. 2.21 Layout of CMOS based 4:1 multiplexer and its corresponding stick diagram

¤ REFERENCES

2.1 D.A. Hodges and H.G. Jackson, *Analysis and Design of Digital Integrated Circuits*, McGraw-Hill Publishing Company, 2nd ed., 1988, ISBN 0-07-029158-6.

2.2 N.H.E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design,* Addison-Wesley, 2nd ed., 1993, ISBN 0-201-53376-6.
2.3 W. Tanner, *MOSIS User Manual, Release 4.0*, August 1994.
2.4 H.W. Johnson and M. Graham, *High Speed Digital Design: A Handbook of Black Magic,* Prentice-Hall Publishing Company, 1993, ISBN 0-13-395724-1.
2.5 R.S. Muller and T.I. Kamins, *Device Electronics for Integrated Circuits,* 2nd ed., John Wiley and Sons, 1986, ISBN 0-471-88758-7.
2.6 Y.P. Tsividis, *Operation and Modeling of the MOS Transistor,* McGraw-Hill, 1987, ISBN0-07-065381-X.
2.7 M. Bohr, "MOS Transistors: Scaling and Performance Trends," *Semiconductor International,* pp. 75–79, June 1995.
2.8 K.Y. Toh, P.K. Ko, and R.G. Meyer, "An Engineering Model for Short-Channel MOS Devices," *IEEE Journal of Solid State Circuits,* Vol. 23, No. 4, August 1988.
2.9 R.A. Pease, J.D. Bruce, H.W. Li, and R.J. Baker, "Comments on Analog Layout Using ALAS!" *IEEE Journal of Solid-State Circuits,* Vol. 31, No. 9, September 1996, pp. 1364–1365.
2.10 F. Maloberti, "Layout of Analog and Mixed Analog-Digital Circuits," in J.E. Franca and Y. Tsividis, eds., *Design of Analog-Digital VLSI Circuits for Telecommunications and Signal Processing,* 2nd ed., Prentice-Hall, 1994, ISBN 0-13-203639-8.
2.11 C.D. Motchenbacher and F.C. Fitchen, *Low-Noise Electronic Design,* John Wiley and Sons, 1973, ISBN 0-471-61950-7.

EXERCISES

2.1 An *n*-channel MOSFET is known to have $2|\phi_F| = 0.57$ V, $\gamma = 0.45$ V$^{1/2}$, $\mu_n = 550$ cm^2/V-s, and $V_{THN0} = 0.8$ V. Assuming $\lambda = 0$, $n_i = 1.45 \times 10^{10}$ atoms/cm^3 and $\frac{kT}{q} = 26$ mV, find the value of *KP*. Suppose *W/L* –10/2. Find I_D when $V_{GS} = 2$ V, $V_{SB} = 1$ V and $V_{fIS} = 1.1$ V.
2.2 If a MOSFET is used as a capacitor in the strong inversion region where the gate is one electrode and the source/drain is the other electrode, does the gate overlap of the source/drain change the capacitance? Why? What is the capacitance?
2.3 If the oxide thickness of a MOSFET is 400 A^o, what is C'_{ox}?
2.4 Show that the parallel connection of MOSFETs shown in Fig. P2.1 behave as a single MOSFET with a width equal to the sum of each individual MOSFET's width.

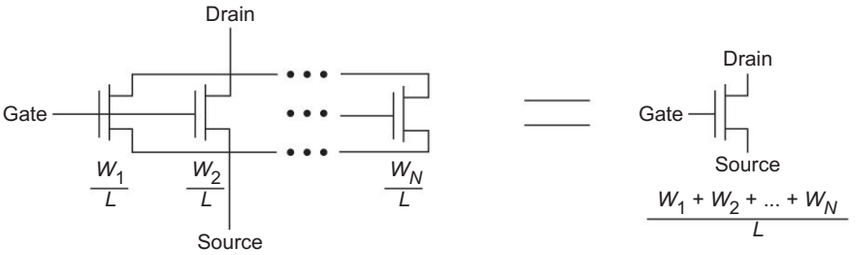


Fig. P2.1

2.5 Show that the bottom MOSFET, Fig. P2.2, in a series connection of two MOSFETs cannot operate in the saturation region. Neglect the body effect. *[Hint:* Show that Ml is always in either cut-off ($V_{GS1} < V_{THN}$) or triode ($V_{M1} < V_{GS1}$)

2.6 Show that the series connection of MOSFETs shown in Fig. P2.2 behaves as a single MOSFET with twice the length of the individual MOSFETs. Again neglect the body effect.

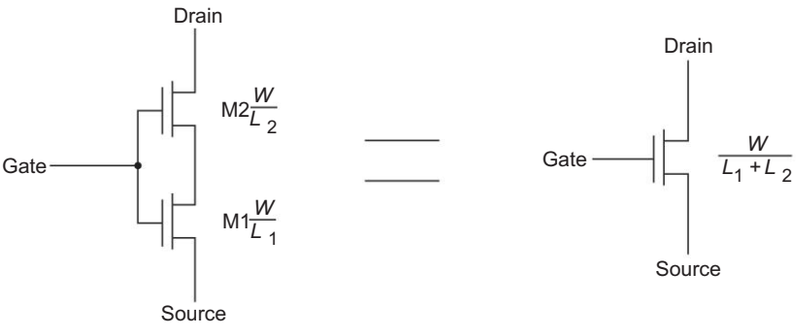


Fig. P2.2

2.7 Draw the circuits from the layouts in Fig. 2.15 to Fig. 2.17 and Fig. 2.20 to Fig. 2.21.

2.8 Draw the stick diagram, symbolic diagram, and layout of the circuits in Fig. P2.1 and Fig. P2.2.

2.9 Draw the stick diagram, symbolic diagram, and layout of the BiCMOS-based circuit in Fig. P2.3.

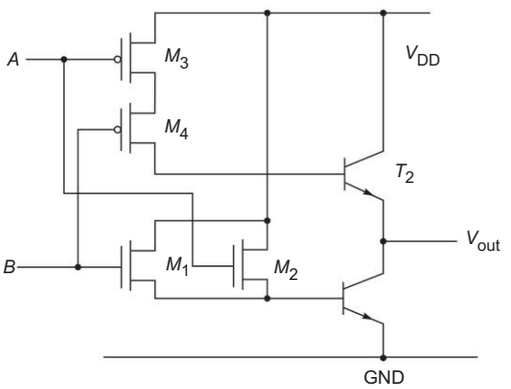


Fig. P2.3

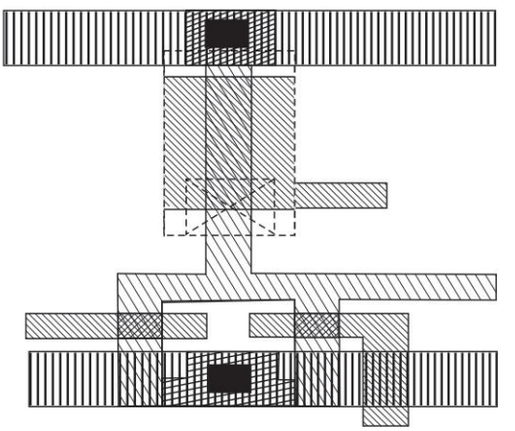


Fig. P2.4

2.10 Make the circuit diagram of the layout shown in Fig. P2.4.

2.11 Draw the stick diagram of the layout shown in Fig. P2.5.

2.12 Make the circuit diagram from the layout shown in Fig. P2.5.

2.13 Make the stick diagram from the layout shown in Fig. P2.5.

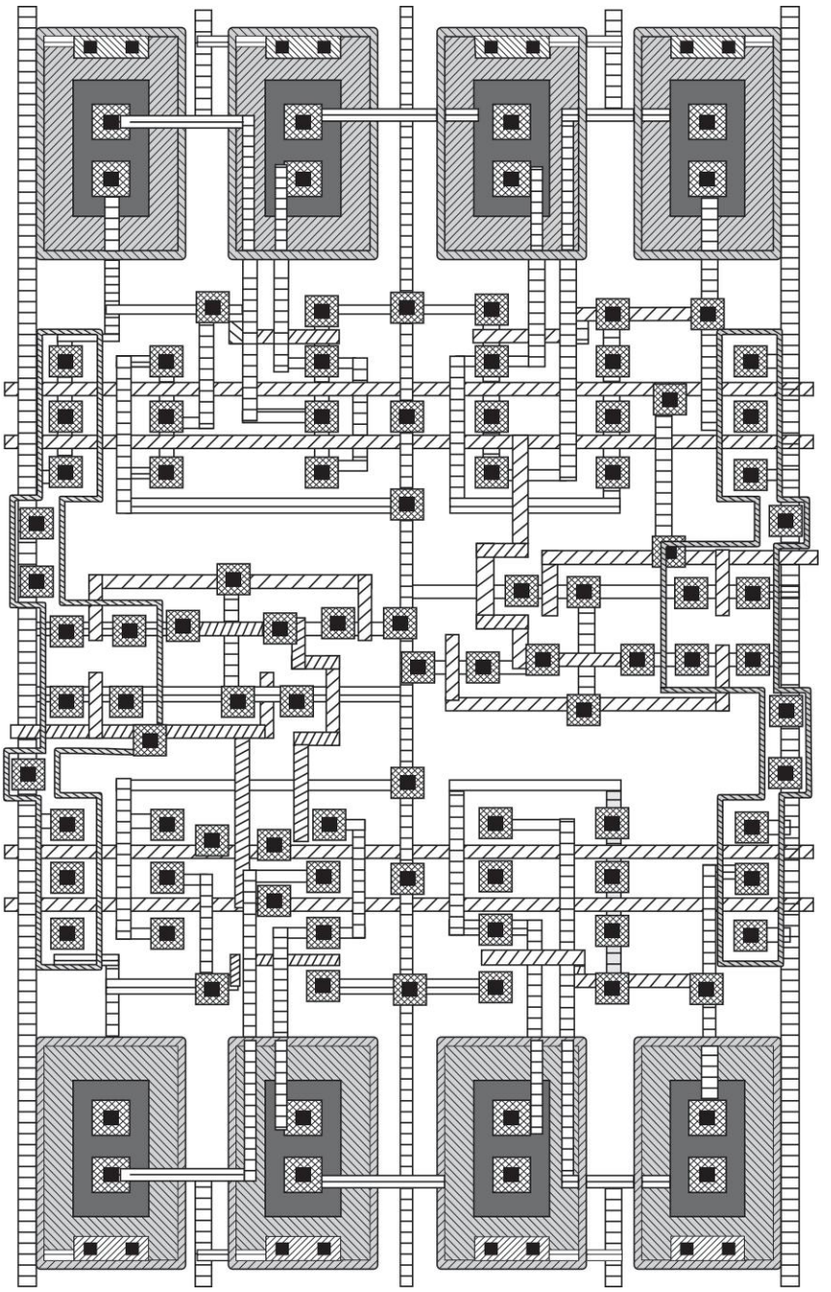


Fig. P2.5

2.14 Estimate the value of multilayer capacitance shown in Fig. 2.19, where relative capacitance is 0.045 pF.

2.15 Estimate the value of the transistor-channel resistance shown in Fig. 2.18, where sheet resistance per square meter is 0.1 ohm/m^2.

3

CMOS-Based Digital Design

In this chapter, we present CMOS/MOS based digital circuits. At this point, the student should be aware of simulation and design of CMOS-based digital circuits. The transition into digital circuit design should be relatively straightforward.

3.1 ◻ DIGITAL MOSFET MODEL

Consider the MOSFET circuit shown in Fig. 3.1. Initially, the MOSFET is off, $V_{GS} = 0$, and the drain of the MOSFET is at V_{DD}. If the gate of the MOSFET is taken instantaneously from 0 to V_{DD}, a current is given by

$$I_{ds} = \frac{\beta}{2}(V_{gs} - V_{THN})^2 \tag{3.1}$$

where β = Device parameter = $K_n W/L$, W = Width of channel, and L = Length of channel. The I_{ds} flows through the MOSFET.

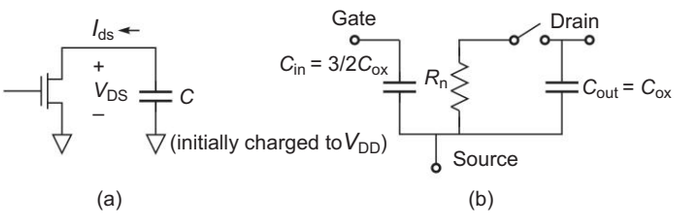


Fig. 3.1 (a) MOSFET switching circuit (b) Its simple digital model

Figure 3.1(b) shows a simple digital MOSFET model consisting of resistance R_n, and input capacitance $C_{in} = 3/2\ C_{ox}$ and output capacitance $C_{out} = C_{ox}$. An estimate for the resistance between the drain and source of the MOSFET is given by

$$R_n = \frac{V_{DD}}{\frac{\beta}{2}(V_{DD} - V_{THN})^2} \tag{3.2a}$$

In this model, when $V_{GS} > V_{DD}/2$, the switch is closed and when $V_{GS} < V_{DD}/2$, the switch is opened. When the input switches from 0 to V_{DD}, the output voltage will decay with a time constant of $R_n . C_{ox}$.

3.1.1 Pass Transistors

The isolated nature of the gate allows MOS transistors to be used as switches in series with lines carrying logic levels similar to the use of relay contacts as shown in Fig. 3.2(a). The application of the MOS device is called pass transistors. The output is given by

$$Y = A.B.C.X$$

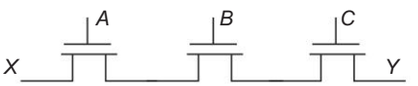


Fig. 3.2 (a) Pass transistors in series

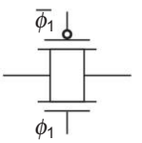


Fig. 3.2 (b) CMOS-based pass transistor

Since the *n*-channel passes logic lows well and the *p*-channel passes logic highs well, putting the two complementary MOSFETs in parallel, as shown in Fig. 3.2(b), results in a TG that passes both logic levels well. The CMOS TG requires two control signals, and ϕ_1 [see Fig. 3.2(b)]. The propagation-delay times of the CMOS TG are

$$t_{PHL} = t_{PLH} = (R_n \,||\, R_p)\, C_{load} \tag{3.2b}$$

The capacitance on the *S* input of the TG is the input capacitance of the *n*-channel MOSFET, or $C_{jnn} = 1.5C_{ox}J$. The capacitance on the *S* input of the TG is the input capacitance of the *p*-channel MOSFET, or C_{inp}. Making the widths of the MOSFETs, used in the TG, large reduces the propagation-delay times from the input to the output of the TG when driving a specific load capacitance. However, the delay times in turning the TG on, the select lines going high, increase because of the increase in input capacitance. This should be remembered when simulating.

3.1.2 Delay Through Series-Connected MOSFETs

Delay can be achieved through series-connected MOSFETS shown in Fig. 3.3(a). The equivalent delay model of the circuit is shown in Fig. 3.3(b). The capacitance of each internal node (MOSFET) is approximately given by

$$C_n = C_{in} + C_{out} = 1.5C_{ox} + C_{ox} = 2.5\, C_{ox} \tag{3.3}$$

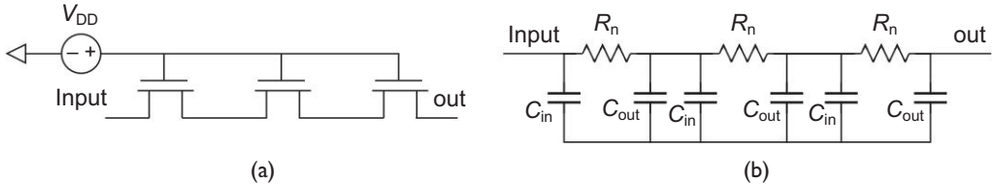


Fig. 3.3 (a) Series connected MOSFETs (b) Delay model

The circuit behaves as an *RC* transmission line with time delay which is given as

$$t_d = 0.35 C_n R_n l^2 \tag{3.4}$$

where *l* is the number of MOSFETs in the series connection. Making the appropriate substitutions into this equation, we get

$$t_d = 0.35 \bullet 2.5 \bullet C_{ox} R_n l^2 = 0.875 C_{ox} R_n \tag{3.5}$$

Example 3.1 □ *Estimate and simulate the delay through ten n-channel MOSFETs. Assume minimum size (L = 2 μm and W = 3 μm) devices. Use the CN20 parameters.*

Solution: The digital model resistances of the *n*- and *p*-channel MOSFETs are

$$R_n = 12k \frac{2\ \mu m}{3\ \mu m} = 8k$$

The oxide resistance $C_{ox} = 4.8$ mf
The delay of 10 series-connected MOSFETs = 4.8. 8. 10^2 = 3.8 ns

3.2 □ CMOS INVERTER

The CMOS inverter is a basic building block for digital circuit design. Figure 3.4 shows the inverter performing the logic operation of A to $\overline{A}$. When the input to the inverter is connected to zero level, the output is pulled to 5 V through the *p*-channel transistor. When the input terminal is connected to V_{DD} the output is pulled to ground through the *n*-channel MOSFET. Its output voltage swings from V_{DD} to zero. The power dissipation of the CMOS inverter is very small. The inverter can be sized to give equal sourcing and sinking capabilities; and the logic switching threshold can be set by changing the size of the device.

From the equivalent circuit (shown in Fig. 3.4), V_{out} is written as $V_{out} = \dfrac{V_{DD}.R_{in}}{R_{in} + R_L}$

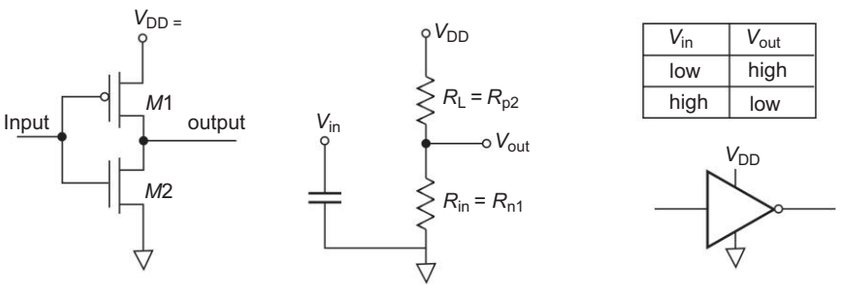


Fig. 3.4 The CMOS inverter, schematic, and logic symbol

"When V_{in} = Low, then $R_{in} > R_L$ and $V_{out} \approx V_{DD}$ = High and when V_{in} = High, then $R_{in} << R_L$ then $V_{out} \approx$ Low"

3.2.1 dc Characteristics

Figure 3.5 shows the transfer characteristic plot (V_{out} versus V_{in}) of an inverter (Fig. 3.3). In the region 1 of the transfer characteristics, the input voltage is sufficiently low (typically not much greater than the threshold voltage of *M*l), so that *M*l is off and *M*2 is on. As V_{in} increases, both *M*2 and *M*l turn on (Region 2). Increasing V_{in} further causes *M*2 to turn off and *M*l to turn on fully, as shown in Region 3.

The maximum output "high" voltage is labeled V_{0H} and the minimum output "low" voltage is labeled, V_{0L}. Points A and B on this curve are defined by the slope of the transfer curves. Input voltages less than or equal to the voltage V_{IL}, defined by the point A, are considered a logic low on the input of the inverter. Input voltages greater than or equal to the voltage V_m, defined by the point B, are considered a logic high on the input of the inverter. Input voltages between V_{IL} and V_{IH} do not specify a valid logic voltage level. The difference in V_{IL} and V_{IH} should be very small for a very good inverter.

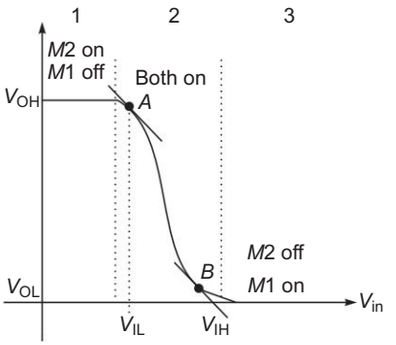


Fig. 3.5 Transfer characteristics of CMOS inverter

3.2.2 Noise Margins

The noise margins of a digital gate or circuit indicate how well the gate will perform under noisy conditions. The noise margin for the high logic levels is written as

$$NM_H = V_{OH} - V_{IH} \tag{3.6}$$

The noise margin for the low logic levels is written as

$$NM_L = V_{IL} - V_{OL} \tag{3.7}$$

Ideally, $NM_L = NM_H = V_{DD}/2$.

3.2.3 Inverter Switching Point

In Fig. 3.6, the point C corresponds to the inverter switching point V_{SP} on the curve when the input voltage is equal to the output voltage. At this point, both MOSFETs in the inverter are in the saturation

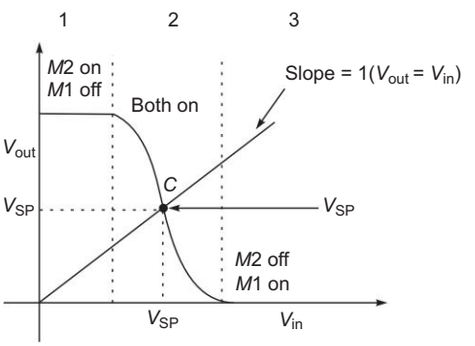


Fig. 3.6 Transfer characteristics of the inverter showing the switching point

region. The inverter switching input voltage V_{SP} can be derived by equaling the drain current in each MOSFET as follows:

$$I_{ds} = \frac{\beta_n}{2}(V_{SP} - V_{THN})^2 = \frac{\beta_p}{2}(V_{DD} - V_{SP} - V_{THP})^2 \tag{3.8}$$

V_{SP} is obtained for the equation as

$$V_{SP} = \frac{\sqrt{\frac{\beta_n}{\beta_p}}.V_{THN} + (V_{DD} - V_{THP})}{1 + \sqrt{\frac{\beta_n}{\beta_p}}} \tag{3.9}$$

3.2.4 Ring Oscillator using Inverters

The odd number of inverters of the circuit shown in Fig. 3.7 forms a closed loop with positive feedback and is called a ring oscillator. We assume that the inverters are identical and minimum-sized. n is the number (odd) of inverters in the ring oscillator. Since the ring oscillator is self-starting, it is often added to a test portion of a wafer to give an indication of the speed of a particular run. For a minimum-size inverter, output capacitance $C_{out} = 2C_{ox}$ and input capacitance $C_{in} = 3C_{ox}$. The total capacitance C_{tot} can be written as

$$C_{lol} = C_{out} + C_{in} = 2C_{ox} + 3C_{ox} = 5C_{0X} \tag{3.10}$$

The propagation-time delay of each inverter corresponding to pMOSFET and nMOSFET transistors [from Fig. 3.3(b)] can be written as

$$t_{HL} + t_{LH} = (R_{nl} + R_{p2})\, C_{tot} \tag{3.11}$$

The oscillation frequency of a ring oscillator can be written as

$$f_c = \frac{1}{n(t_{HL} + t_{LH})} \tag{3.12}$$

where n = Number (odd) of inverters in a ring oscillator

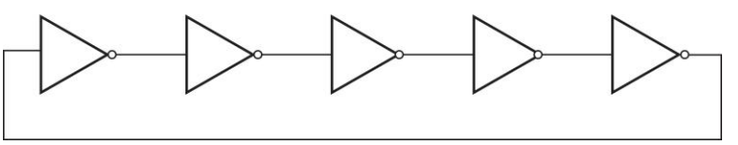


Fig. 3.7 A five-stage ring oscillator

3.2.5 Dynamic Power Dissipation of Inverter

Each time the inverter changes states, it must either supply a charge to C_m or sink the charge stored on C_m to ground. If a square pulse is applied to the input of the inverter with a period T and frequency f_{clk}, the average amount of current that the inverter must pull from V_{DD} is given by

$$I_g = \frac{V_{DD}.C_{tot}}{T} \tag{3.13}$$

The average dynamic power dissipated by the inverter is

$$P_g = V_{DD} \cdot I_g = \frac{C_{tot}.V_{DD}^2}{T} = C_{tot}.V_{DD}^2 \cdot f_{clk} \tag{3.14}$$

So the power dissipation is a function of the clock frequency.

3.2.6 Driving Long Lines using Inverter

In large system design such as Dynamic Random-Access Memory (DRAM), the signal should be transmitted over a line that has a large parasitic resistance R and capacitance C. It is required to drive an RC transmission line by an inverter with resistance/unit length (r) capacitance/unit length (c), and unit length (l) as shown in Fig. 3.8. The time delay from the input to voltage across the capacitor by adding the delays, is given by

$$t_{HL} + t_{LH} = (R_n + R_p)(C_{out} + c - l + C_{load}) + 0.35rcl^2 + (rl)(C_{load}) \tag{3.15}$$

where the first term in this equation is the delay associated with the inverter driving the total capacitance at its output to ground. The second term is the delay through the line, while the last term is estimated as a delay associated with driving a capacitive load through the line's resistance. The most common method of reducing the delay through the transmission line is to place buffer stages with the inverter at different locations along the line. This effectively breaks up the line and can lower the overall delay. If C_{load} contributes more to the delay, a buffer can be inserted between the RC line and C_{load} to reduce the delay.

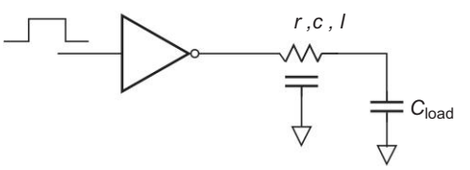


Fig. 3.8 Driving *RC* transmission line by inverter

3.2.7 Other Inverter Configurations

Three other inverter configurations are shown in Fig. 3.9. The inverter shown in Fig. 3.9(a) is an nMOS-only inverter, useful in avoiding latch-up. The inverters shown in Fig. 3.9(b) and (c) use a p-channel load, which is useful in logic gates with a large number of inputs. The output logic low will never reach 0 V for these inverters, and thus the noise margins are poorer than the basic CMOS inverter of Fig. 3.3. The power dissipation of the inverters shown in Fig. 3.9 is greater than the basic CMOS inverter. However, since the input capacitance of these inverters is less than the basic CMOS inverter, and the output voltage swing is reduced, the inverter with the greatest power dissipation is determined by the operating frequency. At high operating frequencies, the basic CMOS inverter dissipates the most power.

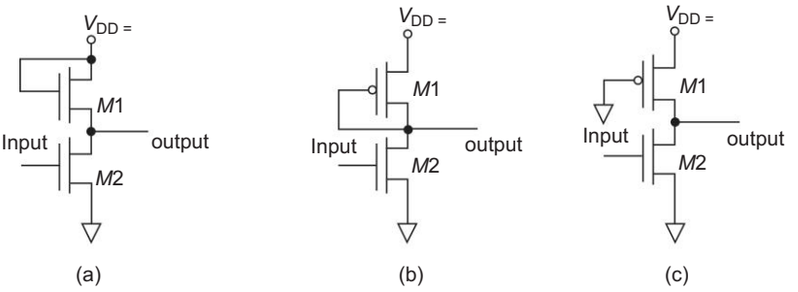


Fig. 3.9 Inverter configurations

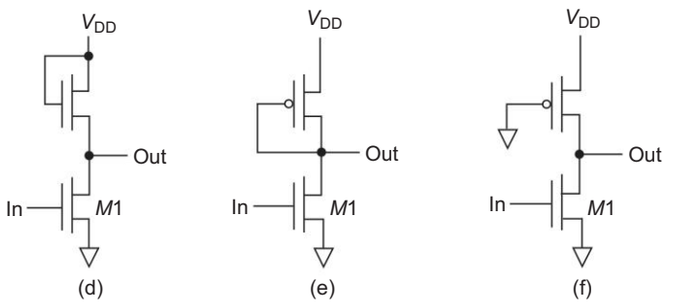


Fig. 3.9 Some other MOS-based inverter configurations

3.2.8 *n*MOS Super Buffer

Figure 3.10 (a) shows the basic "*n*MOS super buffer." When the input signal is low, *M*l and *M*4 are off while *M*2 and *M*3 are on. The output is pulled to ground through *M*2. When the input is high, the buffer causes *M*l and *M*4 to turn on pulling the output to $V_{DD} - V_{THN}$, assuming the input high-signal amplitude is V_{DD}.

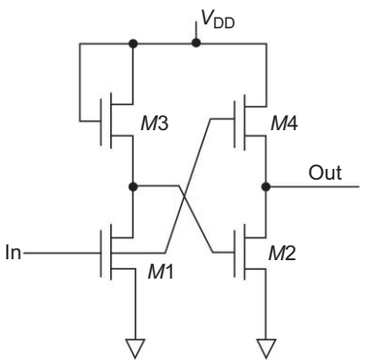


Fig. 3.10(a) NMOS super buffer

The reduced output voltage of the output buffer can be improved using the circuit of Fig. 3.10(b). The inverter driving the gate of *M*2 uses an on-chip generated dc voltage of nominally V_{DD} + 2 V. This allows the output signal to reach V_{DD}.

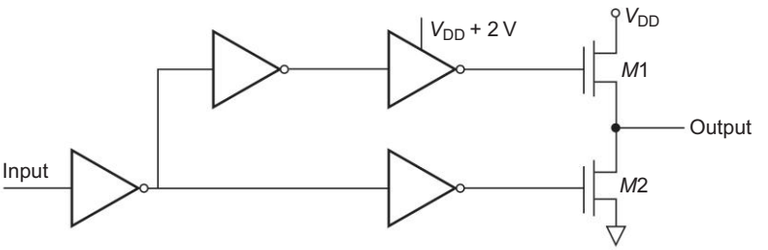


Fig. 3.10 (b) Improved version of output buffer

3.2.9 Inverters with Tri-State Outputs

Figure 3.11 shows an inverter with the tri-state outputs. A high on the *S* input makes the circuit operate normally, i.e., as an inverter. A low on the *S* input forces the output into the Hi-Z, or high-impedance state. These circuits are useful when data are shared on a communication bus. The logic symbol of the tri-state inverter is also shown in Fig. 3.11.

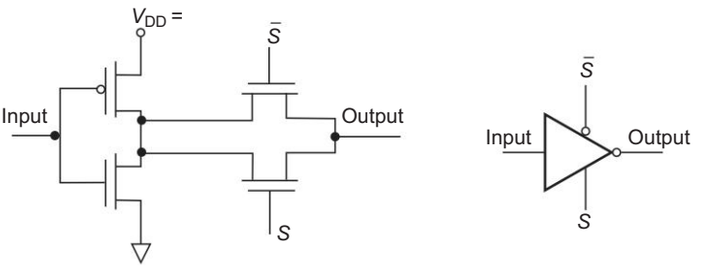
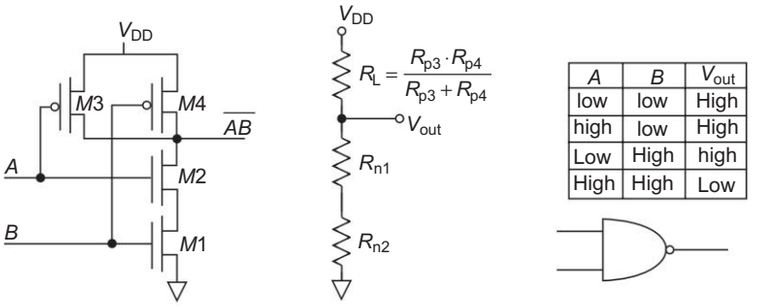


Fig. 3.11 Circuits and logic symbol for the tri-state inverter

3.3 □ CMOS NAND GATE

The CMOS NAND is a universal logic gate for digital logic circuit design. Figure 3.12 shows two-input NAND gates performing the logic operation of AB to $\overline{AB}$ where A and B are two inputs.

From the equivalent circuit (shown in Fig. 3.12), V_{out} is written as $V_{out} = \dfrac{V_{DD}.R_{in}}{R_{in} + R_L}$ where, $R_{in} = R_{n1} + R_{n2}$, and $R_L = \dfrac{R_{p3}.R_{p4}}{R_{p3} + R_{p4}}$,



A	B	V_{out}
low	low	High
high	low	High
Low	High	high
High	High	Low

Fig. 3.12 CMOS NAND gate, equivalent circuit and symbol

When A and B are low then $M1$ and $M2$ are off, $R_{in} >> R_L$, and $V_{out} \sim \frac{V_{DD}.R_{in}}{R_{in}} = V_{DD} = \text{high}.$

When A is high and B is low then $M1$ is on and $M2$ is off, $R_{in} >> R_L$ and $V_{out} \sim \frac{V_{DD}.R_{in}}{R_{in}} = V_{DD} = \text{high}.$

When B is high and A is low then $M1$ is off and $M2$ is on, $R_{in} >> R_L$ and $V_{out} \sim \frac{V_{DD}.R_{in}}{R_{in}} = V_{DD} = \text{high}.$

When A and B are high then $M1$ is on and $M2$ is on, $R_{in} << R_L$ and $V_{out} \sim \frac{V_{DD}.R_{in}}{R_L} = \text{low}.$

3.3.1 dc Characteristics

The dc characteristics are determined by the voltage transfer curve of a gate with p-channel MOSFETs, with $W = W_p$, $L = L_p$, and n-channel MOSFETs with $W = W_n$, $L = L_n$. If both inputs of the gate are shorted, the NAND gate behaves as an inverter.

To determine the gate switching point voltage, V_{SP}, two MOSFETs in parallel behave like a single MOSFET with a width equal to the sum of the individual widths. For the two parallel p-channel MOSFETs M_3 and M_4 in Fig. 3.12, we can write

$$W_3 + W_4 = 2W_p \tag{3.16}$$

The transconductance parameters can also be combined into the transconductance parameter of a single MOSFET, or

$$\beta_3 + \beta_4 = 2\,\beta_p \tag{3.17}$$

Neglecting body effect, two MOSFETs in series (with their gates tied together) are considered to be a single MOSFET with a channel length equal to the sum of the individual MOSFET lengths.

$$L_1 + L_2 = 2L_n \tag{3.18}$$

The transconductance of the single MOSFET is given by

$$\beta_1 + \beta_2 = 2\beta_n/2 \tag{3.19}$$

If we model the NAND gate with both inputs shorted together as an inverter with an n-channel transistor having a width of W_n and length $2L_n$ and a p-channel MOSFET with a width of $2W_p$ and length L_p then we can write the transconductance ratio

$$\text{Transconductance ratio} = \frac{\beta_n}{4\beta_p} \tag{3.20}$$

The switching point voltage of the two-input NAND gate is then given by

$$V_{SP} = \frac{\sqrt{\frac{\beta_n}{4\beta_p}}.V_{THN} + (V_{DD} - V_{THP})}{1 + \sqrt{\frac{\beta_n}{4\beta_p}}} \tag{3.21}$$

3.3.2 Multi-input NAND Gate

Figure 3.13 shows *N*-input NAND gate consisting of *N* number of CMOS. The switching point voltage of the *N*-input NAND gate is then given by

$$V_{SP} = \frac{\sqrt{\frac{\beta_n}{N^2\beta_p}}.V_{THN} + (V_{DD} - V_{THP})}{1 + \sqrt{\frac{\beta_n}{N^2\beta_p}}} \tag{3.22}$$

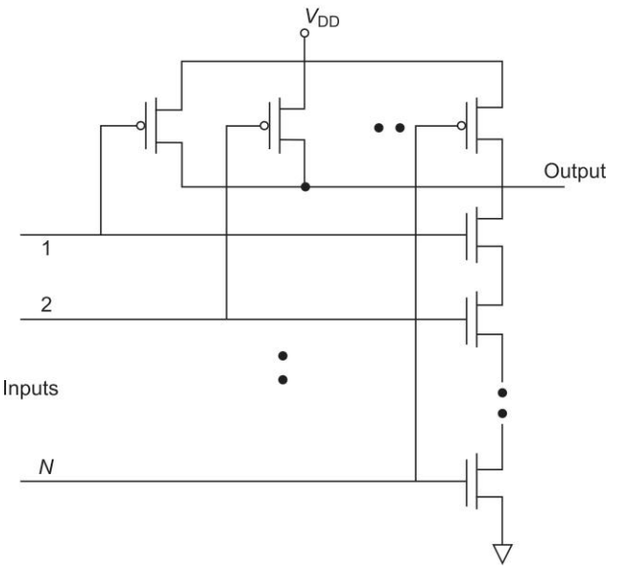


Fig. 3.13 *N*-input NAND gate using CMOS

3.3.3 Propagation Time Delay

We consider the *N*-input NAND gate of Fig. 3.14 driving a capacitive load C_{load}. The low-to-high propagation time is

$$t_{PLH} = R_p/N\ (N.C_{outp} + C_{outp}/N + C_{load}) \tag{3.23}$$

where C_{load} represents the capacitance external to the gate and $R_p.C_{outp}$ represents time constant of *n* MOSFET. If the load capacitance is much greater than the output capacitance of the gate, the low-to-high propagation time can be estimated by

$$t_{PLH} = R_p/N.\ C_{load} \tag{3.24}$$

The high-to-low propagation time can be given by the following equation

$$t_{PLH} = N.R_n\ [C_{outn}/N + N.C_{outp} + C_{load}] + 0.35.\ R_n\ C_{inn}\ (N\text{-}1)^2 \tag{3.25}$$

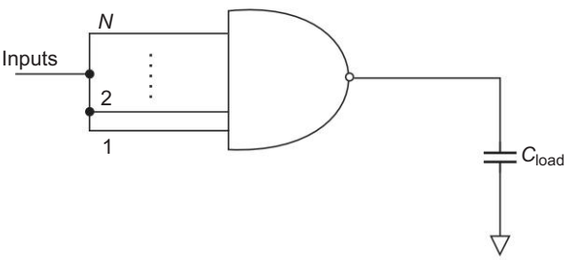
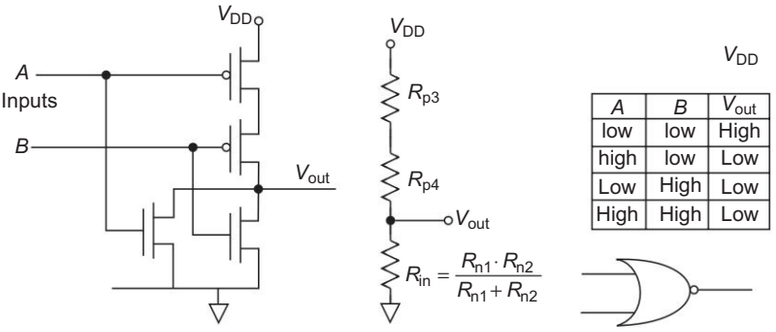


Fig. 3.14 An *N*-input NAND gate-driving capacitive load

3.4 □ CMOS NOR GATE

Figure 3.15 shows two-input NAND gates performing the logic operation of $A + B$ to $\overline{A + B}$ where *A* and *B* are two inputs. From the equivalent circuit (shown in Fig. 3.3), V_{out} is written as $V_{out} = \dfrac{V_{DD}.R_{in}}{R_{in} + R_L}$ where, $R_L = R_{p3} + R_{p4}$ and $R_{in} = \dfrac{R_{n1}.R_{n2}}{R_{n1} + R_{n2}}$.



A	B	Vout
low	low	High
high	low	Low
Low	High	Low
High	High	Low

Fig. 3.15 CMOS NOR gate equivalent circuit and symbol

When A and B are low then $M1$ and $M2$ are off and $R_{in} >> R_L$ and $V_{out} \sim \frac{V_{DD}.R_{in}}{R_{in}} = V_{DD}$ = high.

When A is high and B is low then $M1$ is on and $M2$ is off, $R_{in} << R_L$ and $V_{out} \sim \frac{V_{DD}.R_{in}}{R_L} = V_{DD}$ = low. When B is high and A is low then $M1$ is off and $M2$ is on, $R_{in} << R_L$ and $V_{out} \sim \frac{V_{DD}.R_{in}}{R_L} = V_{DD}$ = low. When A and B are high then $M1$ is on and $M2$ is on, $R_{in} << R_L$ and $V_{out} \sim \frac{V_{DD}.R_{in}}{R_L}$ = low.

3.4.1 dc Characteristics of the NOR Gate

The dc characteristics are determined by the voltage-transfer curve of a gate with p-channel MOSFETs, with $W = W_p$, $L = L_p$, and n-channel MOSFETs with $W = W_n$, $L = L_n$. If both inputs of the gate are shorted, the NOR gate behaves as an inverter.

If we model the NOR gate with both inputs shorted together as an inverter with an n-channel transistor having a width of W_n and length $2L_n$ and a p-channel MOSFET with a width of $2W_p$ and length L_p then we can write the transconductance ratio.

$$\text{Transconductance ratio} = \frac{4\beta_n}{\beta_p} \tag{3.26}$$

The switching point voltage of the two-input NAND gates is then given by

$$V_{SP} = \frac{\sqrt{\frac{4\beta_n}{\beta_p}}.V_{THN} + (V_{DD} - V_{THP})}{1 + \sqrt{\frac{4\beta_n}{\beta_p}}} \tag{3.27}$$

3.4.2 Multi-input NAND Gate

Figure 3.16 shows an N-input NOR gate consisting of N number of CMOS. The switching-point voltage of the N-input NOR gate is then given by

$$V_{SP} = \frac{\sqrt{\frac{N^2\beta_n}{\beta_p}}.V_{THN} + (V_{DD} - V_{THP})}{1 + \sqrt{\frac{N^2\beta_n}{\beta_p}}} \tag{3.28}$$

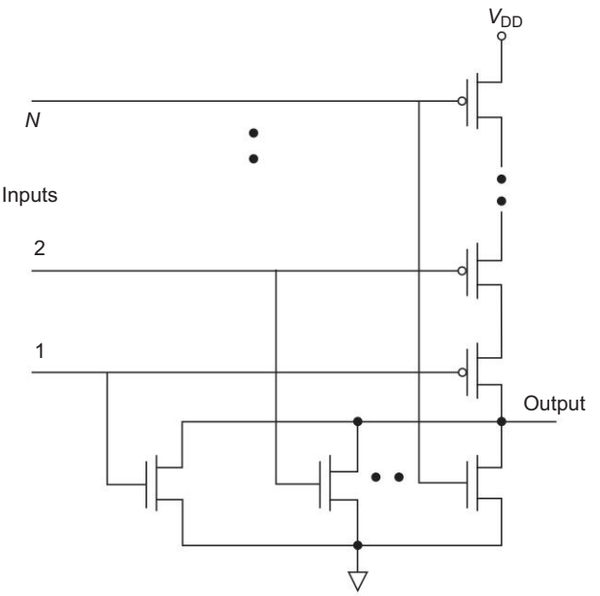


Fig. 3.16 *N*-input NOR gate using CMOS

3.5 ▫ OTHER LOGIC GATES USING NAND GATE

NAND gate is an universal gate which can be used for constructing other gates such as AND, OR, XOR and XNOR gates.

3.5.1 AND Gate

The AND gate can be written in terms of NAND gate as

$$AB = \overline{\overline{AB}} \tag{3.29}$$

Figure 3.17 shows a CMOS-based AND gate.

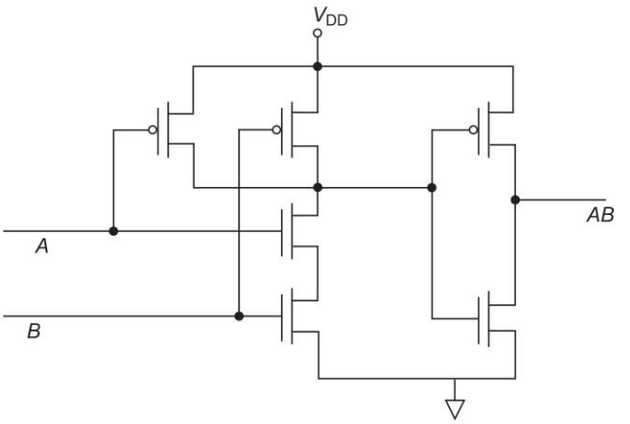


Fig. 3.17 CMOS-based OR gate

3.5.2 OR Gate

The OR gate can be written in terms of NAND gate as

$$A + B = \overline{\overline{A}.\overline{B}} \tag{3.30}$$

Figure 3.18 shows a CMOS-based OR gate.

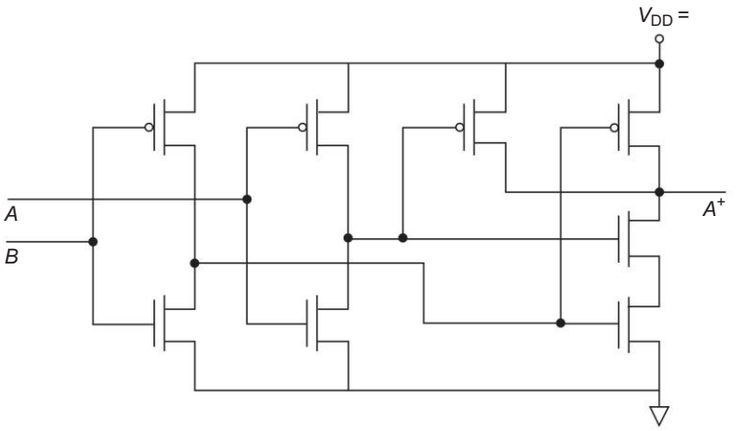


Fig. 3.18 CMOS-based OR gate

3.5.3 XOR Gate

The XOR gate can be written in terms of NAND gate as

$$A \oplus B = \overline{\overline{A\overline{B}}.\overline{\overline{A}B}} \tag{3.31}$$

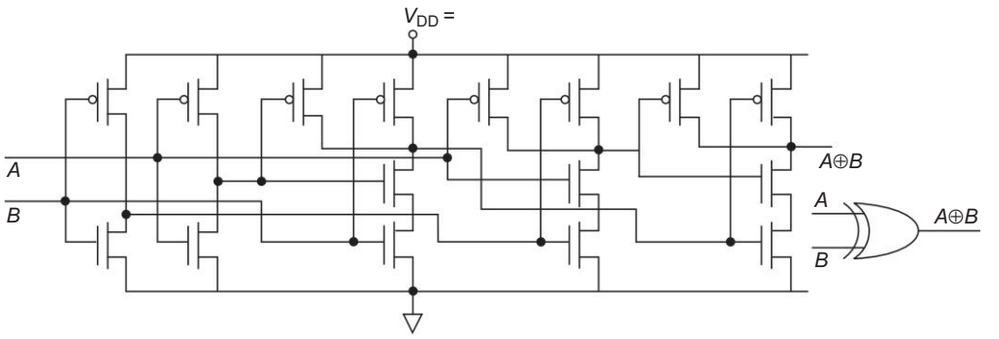


Fig. 3.19 CMOS-based XOR gate and its symbol

3.5.4 XNOR Gate

The XNOR gate can be written in terms of NAND gate as

$$\overline{A \oplus B} = \overline{A\overline{B}}.\overline{\overline{A}B} \tag{3.32}$$

Figure 3.20 shows a CMOS-based XNOR gate.

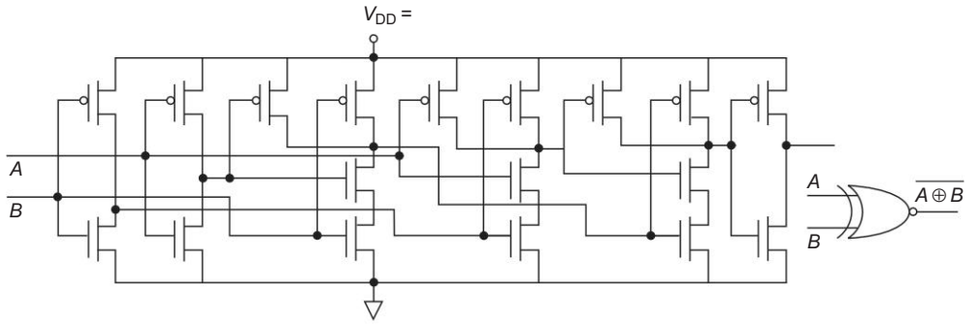


Fig. 3.20 CMOS-based XNOR gate

3.6 ☐ COMBINATIONAL DIGITAL CIRCUIT

Combinational digital circuits are the circuits where the outputs are a function of the current inputs alone. These circuits are based on Boolean equations to build larger and more complex functional forms and expressions.

3.6.1 Binary-to-Decimal Coder/Decoder

The binary-to-decimal converter consists of 10 inputs $A_0, A_1 \ldots\ldots A_9$ and four outputs Y_0, Y_1, Y_2, and Y_3. The outputs written in terms of NAND gate are given below:

$$Y_0 = A_1 + A_3 + A_5 + A_7 + A_9 = \overline{\overline{A_1}.\overline{A_3}.\overline{A_5}.\overline{A_7}.\overline{A_9}} \tag{3.33}$$

$$Y_1 = A_2 + A_3 + A_6 + A_7 = \overline{\overline{A_2}.\overline{A_3}.\overline{A_6}.\overline{A_7}} \tag{3.34}$$

$$Y_2 = A_4 + A_5 + A_6 + A_7 = \overline{\overline{A_4}.\overline{A_5}.\overline{A_6}.\overline{A_7}} \tag{3.35}$$

$$Y_3 = A_8 + A_9 = \overline{\overline{A_8}.\overline{A_9}} \tag{3.36}$$

The decimal-to-binary coder outputs $A_0, A_1 \ldots\ldots A_9$ can be written as functions of Y_0, Y_1, Y_2 and Y_3 as given below:

$$\left.\begin{aligned}
A_0 &= \overline{Y_3}.\overline{Y_2}.\overline{Y_1}.\overline{Y_0} = \overline{\overline{Y_3}.\overline{Y_2}.\overline{Y_1}.\overline{Y_0}} \\
A_1 &= \overline{Y_3}.\overline{Y_2}.\overline{Y_1}.Y_0 = \overline{\overline{Y_3}.\overline{Y_2}.\overline{Y_1}.Y_0} \\
A_2 &= \overline{Y_3}.\overline{Y_2}.Y_1.\overline{Y_0} = \overline{\overline{Y_3}.\overline{Y_2}.Y_1.\overline{Y_0}} \\
A_3 &= \overline{Y_3}.\overline{Y_2}.Y_1.Y_0 = \overline{\overline{Y_3}.\overline{Y_2}.Y_1.Y_0} \\
A_4 &= \overline{Y_3}.Y_2.\overline{Y_1}.\overline{Y_0} = \overline{\overline{Y_3}.Y_2.\overline{Y_1}.\overline{Y_0}} \\
A_5 &= \overline{Y_3}.Y_2.\overline{Y_1}.Y_0 = \overline{\overline{Y_3}.Y_2.\overline{Y_1}.Y_0} \\
A_6 &= \overline{Y_3}.Y_2.Y_1.\overline{Y_0} = \overline{\overline{Y_3}.Y_2.Y_1.\overline{Y_0}} \\
A_7 &= \overline{Y_3}.Y_2.Y_1.Y_0 = \overline{\overline{Y_3}.Y_2.Y_1.Y_0} \\
A_8 &= Y_3.\overline{Y_2}.\overline{Y_1}.\overline{Y_0} = \overline{Y_3}.\overline{Y_2}.\overline{Y_1}.\overline{Y_0} \\
A_9 &= Y_3.\overline{Y_2}.\overline{Y_1}.Y_0 = \overline{Y_3}.\overline{Y_2}.\overline{Y_1}.Y_0
\end{aligned}\right\} \tag{3.37}$$

3.6.2 Binary to Hexadecimal Coder/Decoder

The binary to hexadecimal converter consists of 16 inputs A_0, A_1…….A_{15} and four outputs Y_0, Y_1, Y_2 and Y_3. The outputs written in terms of NAND gate are given below:

$$\left.\begin{aligned} Y_0 &= A_1 + A_3 + A_5 + A_7 + A_9 + A_{11} + A_{13} + A_{15} = \overline{\overline{A_1}.\overline{A_3}.\overline{A_5}.\overline{A_7}.\overline{A_9}.\overline{A_{11}}.\overline{A_{13}}.\overline{A_{15}}} \\ Y_1 &= A_2 + A_3 + A_6 + A_7 + A_{10} + A_{11} + A_{14} + A_{15} = \overline{\overline{A_2}.\overline{A_3}.\overline{A_6}.\overline{A_7}.\overline{A_{10}}.\overline{A_{11}}.\overline{A_{14}}.\overline{A_{15}}} \\ Y_3 &= A_4 + A_5 + A_6 + A_7 + A_{12} + A_{13} + A_{14} + A_{15} = \overline{\overline{A_2}.\overline{A_3}.\overline{A_6}.\overline{A_7}.\overline{A_{10}}.\overline{A_{11}}.\overline{A_{14}}.\overline{A_{15}}} \\ Y_4 &= A_8 + A_9 + A_{10} + A_{11} + A_{12} + A_{13} + A_{14} + A_{15} = \overline{\overline{A_8}.\overline{A_9}.\overline{A_{10}}.\overline{A_{11}}.\overline{A_{12}}.\overline{A_{13}}.\overline{A_{14}}.\overline{A_{15}}} \end{aligned}\right\} \quad (3.38)$$

The binary to hexadecimal decoder outputs A_0, A_1…….A_{15} can be written as function of Y_0, Y_1, Y_2, and Y_3 as given below.

$$\left.\begin{aligned} A_0 &= \overline{Y_3}.\overline{Y_2}.\overline{Y_1}.\overline{Y_0} = \overline{\overline{Y_3}.\overline{Y_2}.\overline{Y_1}.\overline{Y_0}} \\ A_1 &= \overline{Y_3}.\overline{Y_2}.\overline{Y_1}.Y_0 = \overline{\overline{Y_3}.\overline{Y_2}.\overline{Y_1}.Y_0} \\ A_2 &= \overline{Y_3}.\overline{Y_2}.Y_1.\overline{Y_0} = \overline{\overline{Y_3}.\overline{Y_2}.Y_1.\overline{Y_0}} \\ A_3 &= \overline{Y_3}.\overline{Y_2}.Y_1.Y_0 = \overline{\overline{Y_3}.\overline{Y_2}.Y_1.Y_0} \\ A_4 &= \overline{Y_3}.Y_2.\overline{Y_1}.\overline{Y_0} = \overline{\overline{Y_3}.Y_2.\overline{Y_1}.\overline{Y_0}} \\ A_5 &= \overline{Y_3}.Y_2.\overline{Y_1}.Y_0 = \overline{\overline{Y_3}.Y_2.\overline{Y_1}.Y_0} \\ A_6 &= \overline{Y_3}.Y_2.Y_1.\overline{Y_0} = \overline{\overline{Y_3}.Y_2.Y_1.\overline{Y_0}} \\ A_7 &= \overline{Y_3}.Y_2.Y_1.Y_0 = \overline{\overline{Y_3}.Y_2.Y_1.Y_0} \\ A_8 &= Y_3.\overline{Y_2}.\overline{Y_1}.\overline{Y_0} = \overline{Y_3.\overline{Y_2}.\overline{Y_1}.\overline{Y_0}} \\ A_9 &= Y_3.\overline{Y_2}.\overline{Y_1}.Y_0 = \overline{Y_3.\overline{Y_2}.\overline{Y_1}.Y_0} \\ A_{10} &= Y_3.\overline{Y_2}.Y_1.\overline{Y_0} = \overline{Y_3.\overline{Y_2}.Y_1.\overline{Y_0}} \\ A_{11} &= Y_3.\overline{Y_2}.Y_1.Y_0 = \overline{Y_3.\overline{Y_2}.Y_1Y_0} \\ A_{12} &= Y_3.Y_2.\overline{Y_1}.\overline{Y_0} = \overline{Y_3.Y_2.\overline{Y_1}.\overline{Y_0}} \\ A_{14} &= Y_3.Y_2.Y_1.\overline{Y_0} = \overline{Y_3.Y_2.Y_1.\overline{Y_0}} \\ A_{15} &= Y_3.Y_2.Y_1.Y_0 = \overline{Y_3.Y_2.Y_1.Y_0} \end{aligned}\right\} \quad (3.39)$$

3.6.3 Binary-to-Gray Coder/Decoder

The five-bit binary-to-Gray coder consisting of six inputs A_0, A_1…….A_5 and five outputs Y_0, Y_1...Y_4 is written in terms of NAND gate as given below:

$$\left.\begin{aligned} Y_0 &= A_0 \oplus A_1 = \overline{\overline{A_0\,\overline{A_1}}.\,\overline{\overline{A_0}\,A_1}} \\ Y_1 &= A_1 \oplus A_2 = \overline{\overline{A_1\,\overline{A_2}}.\,\overline{\overline{A_1}\,A_2}} \\ Y_2 &= A_2 \oplus A_3 = \overline{\overline{A_2\,\overline{A_3}}.\,\overline{\overline{A_2}\,A_3}} \end{aligned}\right\} \qquad (3.40)$$

$$Y_3 = A_3 \oplus A_4 = \overline{\overline{A_3\,\overline{A_4}}.\,\overline{\overline{A_3}\,A_4}}$$

$$Y_4 = A_5$$

The five-bit Gray-to-binary decoder consisting six inputs Y_0, Y_1…….Y_5 and five outputs A_0, A_1...A_4 is written in terms of NAND gate given below:

$$\left.\begin{aligned} A_0 &= Y_0 \oplus Y_1 = \overline{\overline{Y_0\,\overline{Y_1}}.\,\overline{\overline{Y_0}\,Y_1}} \\ A_1 &= Y_1 \oplus Y_2 = \overline{\overline{Y_1\,\overline{Y_2}}.\,\overline{\overline{Y_1}\,Y_2}} \\ A_2 &= Y_2 \oplus Y_3 = \overline{\overline{Y_2\,\overline{Y_3}}.\,\overline{\overline{Y_2}\,Y_3}} \end{aligned}\right\}$$

$$A_3 = Y_3 \oplus Y_4 = \overline{\overline{Y_3\,\overline{Y_4}}.\,\overline{\overline{Y_3}\,Y_4}} \qquad (3.41)$$

$$A_4 = Y_5$$

3.6.4 Multiplexer/Data Selector

Multiplex means many-to-one circuit in which there are many inputs and only one output. By applying control signal, one can select any input to the output. For a 4-to-1 multiplexer or data selector, two control signals S_0 and S_1 are required. The output Y written in terms of input data D_0….D_3 is

$$\begin{aligned} Y &= D_0.\overline{S_1}.\overline{S_0} + D_1.\overline{S_1}.S_0 + D_2.S_1.\overline{S_0} + D_3.S_1.S_0 \\ &= \overline{\overline{D_0.\overline{S_1}.\overline{S_0}}.\overline{D_1.\overline{S_1}.S_0}.\overline{D_2.S_1.\overline{S_0}}.\overline{D_3.S_1.S_0}} \end{aligned} \qquad (3.42)$$

The output Y of a 16-to-1 multiplexer consisting of 16 inputs D_0, D_1…….D_{15} and four control signals S_0, S_1…S_3 can be written as

$$\begin{aligned} Y &= D_0.\overline{S_3}\,\overline{S_2}\,\overline{S_1}.\overline{S_0} + D_1.\overline{S_3}.\overline{S_2}\,\overline{S_1}.S_0 + \ldots\ldots + D_3.S_3.S_2.S_1.S_0 \\ &= \overline{\overline{D_0.\overline{S_3}\,\overline{S_2}\,\overline{S_1}.\overline{S_0}}.\overline{D_1.\overline{S_3}.\overline{S_2}\,\overline{S_1}.S_0}\ldots\ldots\ldots\ldots\overline{D_3.S_3.S_2.S_1.S_0}} \end{aligned} \qquad (3.43)$$

3.6.5 Nibble Multiplexer/Word Multiplexer

A nibble multiplexer is used for multiplexing digital words or selecting one digital word from many digital words in the output on the basis of control signals. Figure 3.21 shows a four-bit 4-to-1 word multiplexer having digital words $A_0 A_1 A_2$, $B_0 B_1 B_2$, $C_1 C_2 C_3$, and $D_1 D_2 D_3$ and requiring control signals S_0 and S_1, and outputs Y_0, Y_1 and Y_2 which is written as

$$\left.\begin{aligned} Y_0 &= A_0.\overline{S_1}.\overline{S_0} + B_0.\overline{S_1}.S_0 + C_0.S_1.\overline{S_0} + D_0.S_1.S_0 \\ &= \overline{\overline{A_0.\overline{S_1}.\overline{S_0}}.\overline{B_0.\overline{S_1}.S_0}.\overline{C_0.S_1.\overline{S_0}}.\overline{D_0.S_1.S_0}} \\ Y_1 &= A_1.\overline{S_1}.\overline{S_0} + B_1.\overline{S_1}.S_0 + C_1.S_1.\overline{S_0} + D_1.S_1.S_0 \\ &= \overline{\overline{A_1.\overline{S_1}.\overline{S_0}}.\overline{B_1.\overline{S_1}.S_0}.\overline{C_1.S_1.\overline{S_0}}.\overline{D_1.S_1.S_0}} \end{aligned}\right\}$$

$$\begin{aligned} Y_2 &= A_3.\overline{S_1}.\overline{S_0} + B_3.\overline{S_1}.S_0 + C_3.S_1.\overline{S_0} + D_3.S_1.S_0 \\ &= \overline{\overline{A_3.\overline{S_1}.\overline{S_0}}.\overline{B_3.\overline{S_1}.S_0}.\overline{C_3.S_1.\overline{S_0}}.\overline{D_3.S_1.S_0}} \end{aligned} \tag{3.44}$$

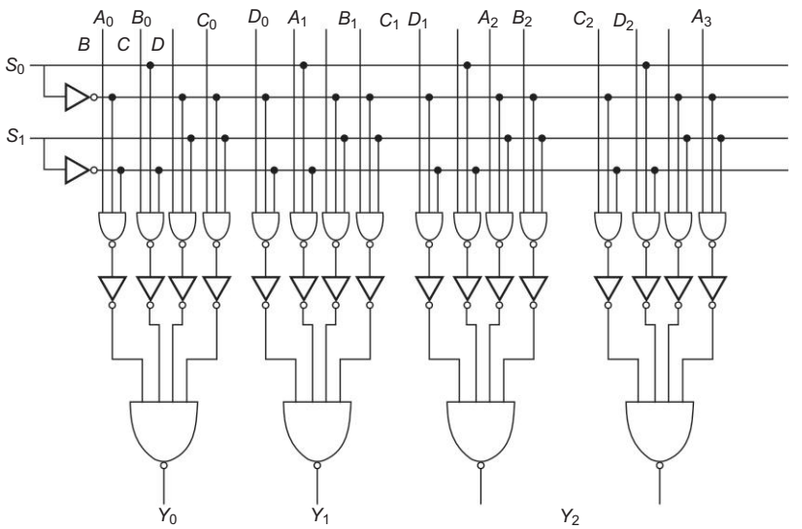


Fig. 3.21 A four-bit 4 to 1 word multiplexer

3.6.6 Word Comparator

A word comparator is a digital circuit that recognizes two identical words, We consider comparison between two digital words $A = A_4 A_3 A_2 A_1 A_0$ and $B = B_4 B_3 B_2 B_1 B_0$. The output Y of a word comparator is written as

$$\begin{aligned} Y &= \overline{A_5 \oplus B_5}.\overline{A_4 \oplus B_4}.\overline{A_3 \oplus B_3}.\overline{A_2 \oplus B_2}.\overline{A_1 \oplus B_1}.\overline{A_0 \oplus B_0} \\ &= \overline{A_4\overline{B_4}}.\overline{\overline{A_4}B_4}.\overline{A_3\overline{B_3}}.\overline{\overline{A_3}B_3}.\overline{A_2\overline{B_2}}.\overline{\overline{A_2}B_2}.\overline{A_1\overline{B_1}}.\overline{\overline{A_1}B_1}.\overline{A_0\overline{B_0}}.\overline{\overline{A_0}B_0} \\ &= \overline{\overline{A_4\overline{B_4}}.\overline{\overline{A_4}B_4}.\overline{A_3\overline{B_3}}.\overline{\overline{A_3}B_3}.\overline{A_2\overline{B_2}}.\overline{\overline{A_2}B_2}.\overline{A_1\overline{B_1}}.\overline{\overline{A_1}B_1}.\overline{A_0\overline{B_0}}.\overline{\overline{A_0}B_0}} \end{aligned} \tag{3.45}$$

If two words A and B are identical, all XNOR have high inputs and as a result, the output Y is high. When two words A and B are not identical, the output Y becomes low.

3.6.7 Arithmetic Digital Circuit

Arithmetic digital circuit is a circuit that does arithmetic operation of binary numbers such as summation and subtraction. In this section, we discuss the circuit for summation and subtraction.

1. Half Adder

A half adder is a logic circuit that adds two bits A, B and has two outputs carry C and sum S which are expressed as

$$\left.\begin{aligned} S = A \oplus B = \overline{\overline{AB}.\,\overline{AB}} \\ C = AB = \overline{\overline{A.B}} \end{aligned}\right\} \tag{3.46}$$

Figure 3.22 shows a CMOS based half-adder circuit.

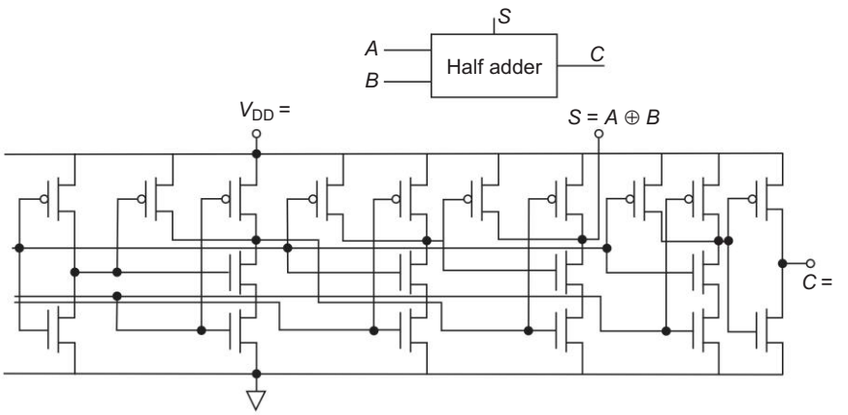


Fig. 3.22 CMOS-based half adder and its symbol

2. Full Adder

A full adder is a logic circuit consisting of three inputs A_n, B_n, and C_n, and has two outputs carry C_{n+1} and sum S_n which are expressed as

$$\left.\begin{aligned} S_n = A_n \oplus B_n \oplus C_n \\ C_{n+1} = A_nB_n + B_nC_n + A_nC_n = \overline{\overline{A_n.B_n}.\overline{B_n.C_n}.\overline{A_n.C_n}} \end{aligned}\right\} \tag{3.47}$$

Figure 3.23 shows a CMOS-based full-adder circuit.

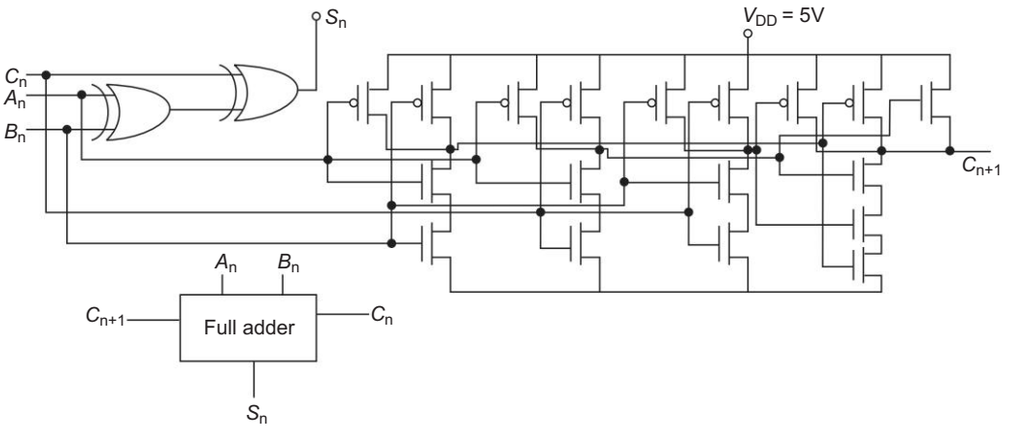


Fig. 3.23 CMOS-based full adder and its symbol

3. 2's Complement Subtractor-adder Circuit

Figure 3.24 shows a 2's complement adder-subtractor circuit that can add or subtract binary numbers. There are two digital words/binary numbers $A = A_3 A_2 A_1 A_0$ and $B = B_3 B_2 B_1 B_0$. When Sub is low, the B bits pass through the controlled inverter without inversion and as a result, the sum of full adders $S = A + B$, where $S = S_3 S_2 S_1 S_0$.

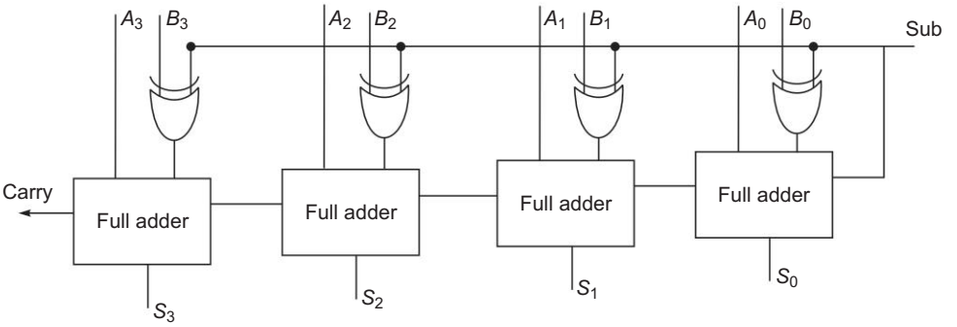


Fig. 3.24 2's complement adder/subtracter circuit

When Sub is high, the controlled inverter produces a 1's complement and the high Sub adds high to the first full adder and the addition of high provides conversion of 1's complement to 2's complement. As a result, the sum of full adders

$$S = A + B' = A - B \tag{3.48}$$

4. Multiplication

High-speed multiplication is another arithmetic operation in VLSI applications. The Braun array multiplier is a high-speed multiplication method which is discussed in this section. We consider finding the

multiplication of two binary numbers $A = a_m \ldots a_1 a_0$ and $B = b_n \ldots b_1 b_0$ by using this multiplier. The decimal values of these numbers are

$$\left.\begin{aligned} A_d &= \sum_{i=0}^{m} a_i 2^i \\ B_d &= \sum_{j=0}^{n} b_j 2^j \end{aligned}\right\} \tag{3.49}$$

The product P of A and B has $(m + n)$ bits. Its decimal value is

$$P_d = \left(\sum_{i=0}^{m} a_i 2^i\right) \times \left(\sum_{j=0}^{n} b_i 2^j\right) = \sum_{i=0}^{m} \sum_{j=0}^{n} (a_i \times b_j) 2^{i+j} \tag{3.50}$$

In the binary system, the multiplication of a multicant bit a_i by a multiplier bit b_j can be implemented by a logic AND function to produce a partial product bit $a_i b_j$. The above expression can be written as

$$P_d = \sum_{i=0}^{m} \sum_{j=0}^{n} (a_i b_j) 2^{i+j} = \sum_{k=0}^{m+n+1} p_k 2^k \tag{3.51}$$

Realizing that the multiplication is simply the addition of $a_i b_j$'s $(0 \le i \le m, 0 \le j \le n)$, a full-adder tree can be built to estimate the product of the bits p_k's $(0 \le k \le (m + n + 1))$.

For example, we take the Braun array multiplier to multiply between $A = a_3 a_2 a_1 a_0$ and $B = b_3 b_2 b_1 b_0$. The expression for the product bits are

$$\left.\begin{aligned} p_0 &= a_0 b_0 \\ p_1 &= a_1 b_0 + a_0 b_1 + c \\ p_2 &= a_2 b_1 + a_1 b_1 + a_0 b_2 + c \\ p_3 &= a_3 b_0 + a_2 b_1 + a_1 b_2 + a_0 b_3 + c \\ p_4 &= a_3 b_1 + a_1 b_3 + a_2 b_2 + c \\ p_5 &= a_3 b_2 + a_2 b_3 + c \\ p_6 &= a_3 b_3 + c \\ p_7 &= c \end{aligned}\right\} \tag{3.52}$$

where c is the carry from a previous column.

Figure 3.25 shows a full adder tree created for multiplication of $A = a_3 a_2 a_1 a_0$ and $B = b_3 b_2 b_1 b_0$ using Braun array multiplier. In the figure, except the distribution of all input signals, all interconnections are limited to the nearest neighbors. Here, instead of having AND gate outside the array, the AND functions are included into the Full Adder (FA) cells. The area A_b and delay time τ_b of 4-bit by 4-bit Braun array multiplier in the figure is given by

$$\left.\begin{aligned} A_b &= 16 A_{\text{AND}} + 12 A_{\text{FA}} \\ \tau_b &= \tau_{\text{AND}} + 6 \tau_{\text{FA}} \end{aligned}\right\} \tag{3.53}$$

where A_{AND} and τ_{AND} are area and delay time of a two-input AND gate respectively. A_{FA} and τ_{FA} are area and delay time of a full adder respectively.

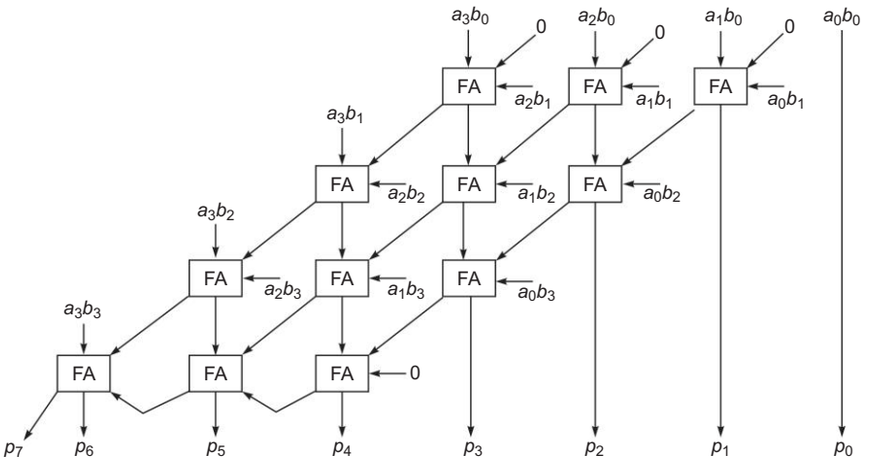


Fig. 3.25 Braun array multiplier

3.6.8 Complex CMOS Logic Gates

Figure 3.26 shows implementation of complex logic functions in CMOS which uses the basic building blocks. In general, And-Or-Invert (AOI) logic function can be implemented using these techniques. A major benefit of AOI logic is that for a relatively complex logic function, the delay can be significantly lower than a logic-gate implementation. The figure shows NAND and NOR gates implementation using AOI logic.

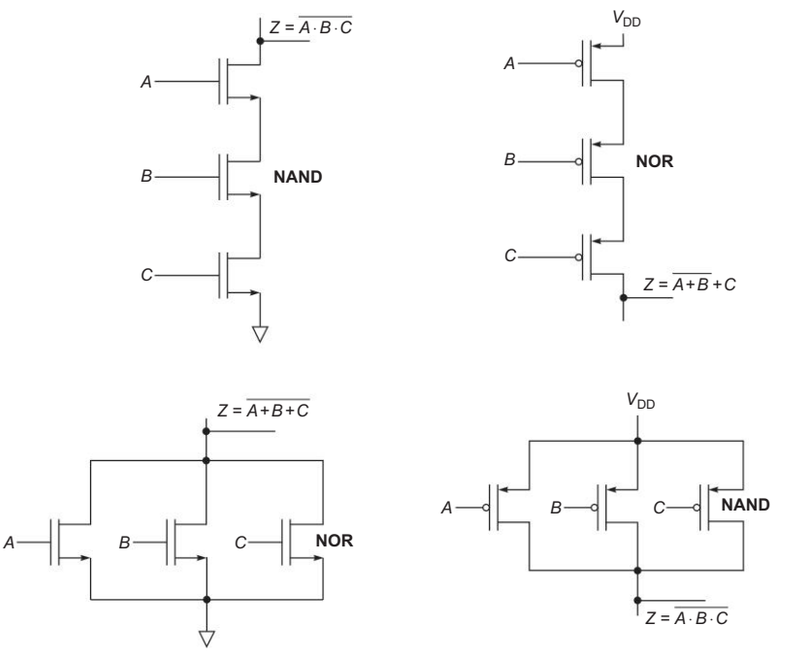


Fig. 3.26 Logic implementiion in CMOS

1. Exclusive OR gate (XOR) using AOI Logic

From basic truth table of XOR, its logic function for the XOR gate is given by

$$Z = A \oplus B = (A + B).(\overline{A} + \overline{B}) = \overline{\overline{A}.\overline{B} + A.B} \tag{3.54}$$

Figure 3.27 shows AOI-based CMOS XOR.

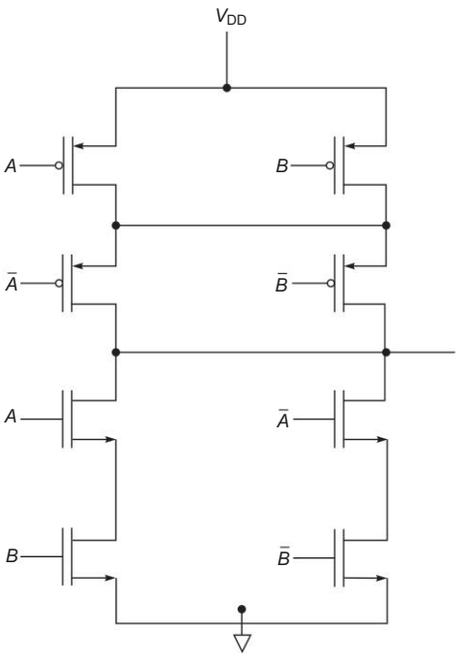


Fig. 3.27 CMOS AOI XOR gate

2. Full Adder using AOI

The logic function of sum and carry outputs for full adder can be written as

$$\left.\begin{aligned} S_n = A_n \oplus B_n \oplus C_n = \overline{A_n}.\overline{B_n}.C_n + \overline{A_n}.B_n.\overline{C_n} + A_n.\overline{B_n}.\overline{C_n} + A_n.B_n.C_n \\ C_{n+1} = A_n.B_n + C_n(A_n + B_n) \end{aligned}\right\}$$

We can write

$$\overline{C_{n+1}} = (\overline{A_n} + \overline{B_n}).(\overline{C_n} + \overline{A_n}.\overline{B_n}) \tag{3.55}$$

Sum S_n can be rewritten in terms of C_{n+1} as

$$S_n = (A_n + B_n + C_n).\overline{C_{n+1}} + A_n.B_n.C_n \tag{3.56}$$

Figure 3.28 shows AOI based CMOS full-adder circuit.

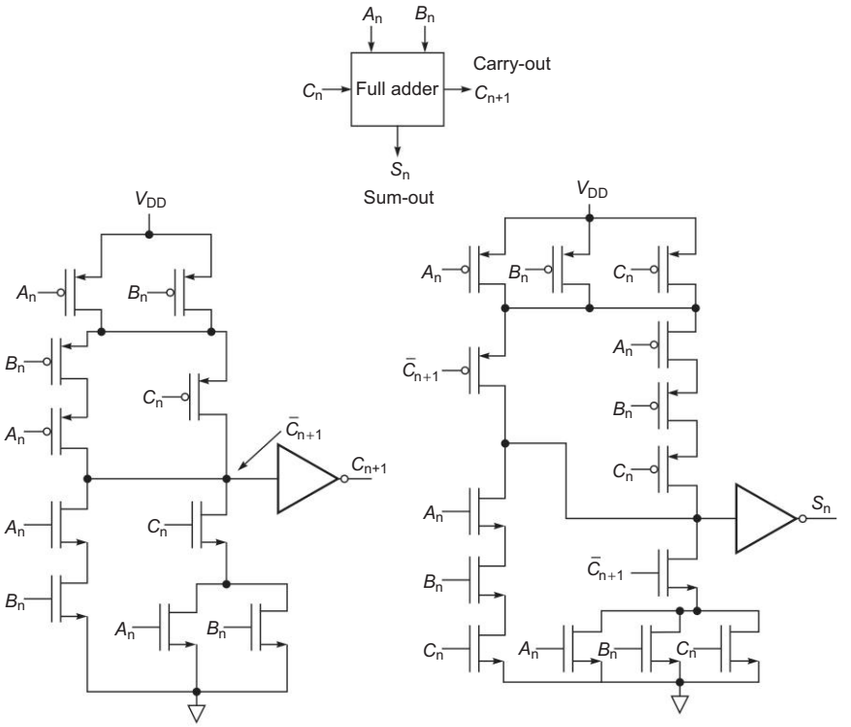


Fig. 3.28 A OI-based CMOS full adder

Example 3.2 ◻ *Implement the following logic function by using AOI.*

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

Solution:

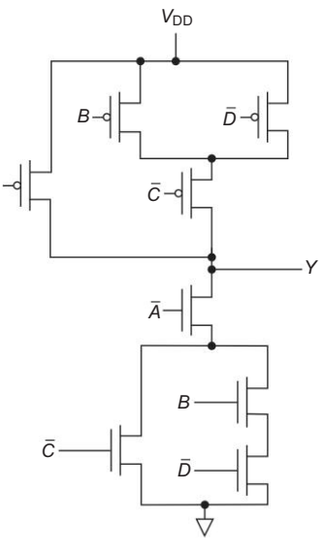


Fig. 3.29 Implementation of Example 3.2

3.7 ☐ SEQUENTIAL DIGITAL CIRCUIT

The sequential circuits are circuits whose outputs not only depend on the current inputs but also on the past ones. The present state of such a circuit is required to be stored in memory elements and determined by the operation history of the circuit. Whereas the output of a combinational circuit does not require memory and is thus completely determined by its present input. This limits applications of a combinational circuit.

A sequential circuit can operate with and without the control of clock signal which is a periodic square wave that can be used to trigger the transition states. Figure 3.30 shows general model of a synchronous circuit with clock signal in which x and y are inputs and the outputs are Z and Y. The output Y is stored in the memory as a state variable. The number of bits in the state variable decides the number of available states. The Z and Y can be written as

$$Z = f(x, y)$$
$$Y = g(x, y)$$

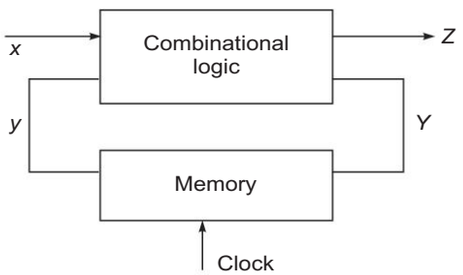


Fig. 3.30 General model of a sequential circuit

3.7.1 Asynchronous Sequential Circuit

Figure 3.31(a) shows a general model of asynchronous sequential circuit in which the next state Y directly feeds back to y without going through memory elements. So, in this circuit, the state transition is determined solely by combinational logic delay. Figure 3.31(b) shows an inverter-based asynchronous sequential circuit. The ring oscillator circuit discussed earlier in Section 3.2.4 is also an example of this type of circuit.

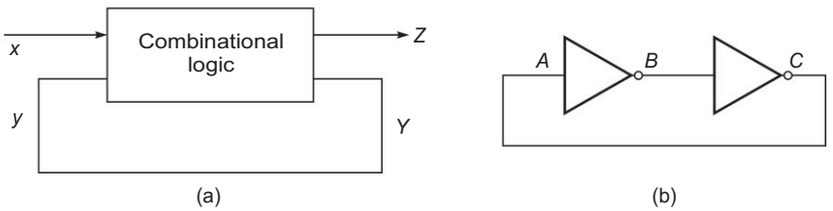
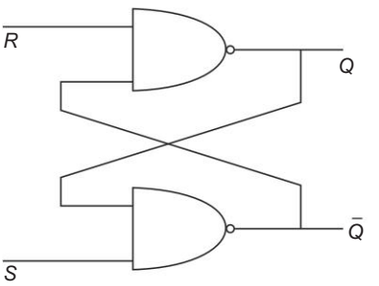


Fig. 3.31 Asynchronous sequential circuit (a) General model (b) Inverter based

3.7.2 *RS* Flip-Flop

A flip-flop is a device with two stable states where it remains in one of these states until triggered into other. Figure 3.32 shows NAND latch *RS* flip-flop with a truth table consisting of two inputs *R* and *S,* and two outputs *Q* and $\overline{Q}$. When both *R* and *S* are low, there is no change in output. When *R* is low and *S* is high, $Q = 1$ and $\overline{Q} = 0$ and output becomes SET state. When *R* is high and *S* is low, $Q = 0$ and $\overline{Q} = 1$ and output becomes RESET state.

When both *R* and *S* are high, output shows unpredictable state called RACE condition of flip-flop. In this transition, since both *R* and *S* are high, both NAND gates try to latch the output and the comparatively faster NAND gate wins the race and latches the output.



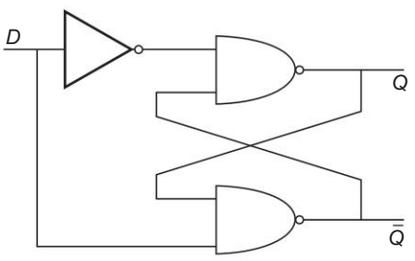
R	*S*	*Q*	Comments
Low	Low	NC	No change
Low	High	High	SET
High	low	low	RESET
High	High	Low	RACE

Fig. 3.32 *D* flip-flop and its truth table

3.7.3 *D* Flip-Flop

To eliminate race condition of *RS* flip-flop, it is required to design a new type of flip-flop having one input. This type of flip-flop is called *D* flip-flop. There are three types of *D* flip-flops—unclocked *D* flip-flop, clocked *D* flip-flop, and edge-triggered flip-flop. We can also make three *D* flip-flops with preset and clear.

Figure 3.33(a) shows an unclocked *D* flip-flop consisting of one input *D* and two outputs, *Q* and $\overline{Q}$. When *D* is high, *Q* = high, and output becomes SET state. When *D* is low, $Q = 0$ and $\overline{Q} = 1$, and output becomes RESET state. In this case, there is no race condition.

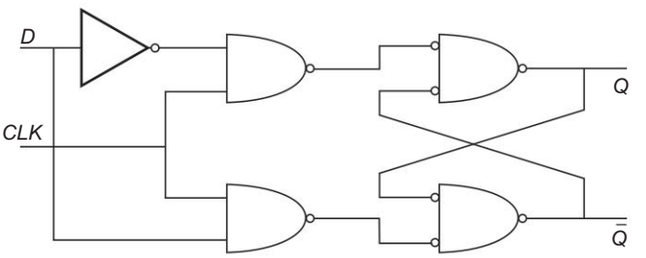


D	*Q*	Comments
High	High	SET
Low	Low	RESET

Fig. 3.33(a) *D* flip-flop and its truth table

Figure 3.33(b) shows a clocked *D* flip-flop consisting of one input *D* and two outputs, *Q* and $\overline{Q}$. A low clock *CLK* disables the input gate and prevents latch from changing states. For changing states, *CLK* should be high. As *CLK* is high, when *D* is high, *Q* becomes high and output becomes SET state,

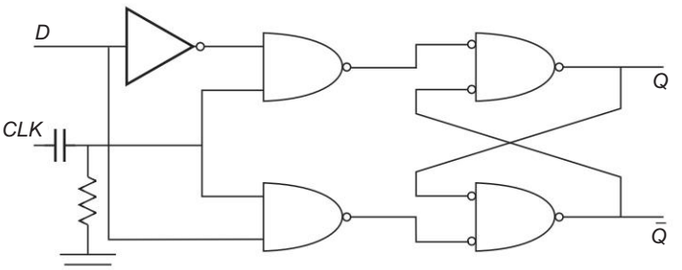
and when D is low, Q becomes low and $\overline{Q} = 1$ and output becomes RESET state. It means that as *CLK* is high, Q becomes D.



CLK	*D*	*Q*
Low	*X*	NC
High	Low	Low
High	High	High

Fig. 3.33 (b) Clocked *D* flip-flop

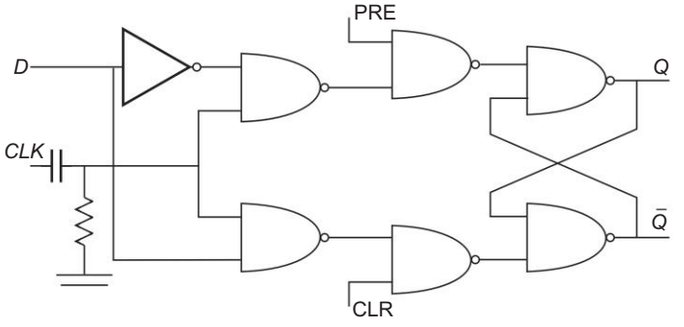
Figure 3.33(c) shows an edge-triggered clocked D flip-flop consisting of one input D and two outputs, Q and $\overline{Q}$. Both low and high clock *CLK* disables the input gate and prevents latch from changing states and also, negative-triggered clock *CLK* disables the input gate For changing states, *CLK* should be positive triggered. As *CLK* is positive triggered, when D is high, Q becomes high and output becomes SET state and when D is low, Q becomes low and $\overline{Q} = 1$, and output becomes RESET state.



CLK	*D*	*Q*
Low	*X*	NC
High	*X*	NC
↓	*X*	NC
↑	0	0
↑	1	1

Fig. 3.33 (c) Edge-triggered flip-flop

Figure 3.33(d) shows edge-triggered clocked D flip-flop with preset PRE and clear CLR consisting of one input D and two outputs, Q and $\overline{Q}$. In the figure, the truth table is also shown.

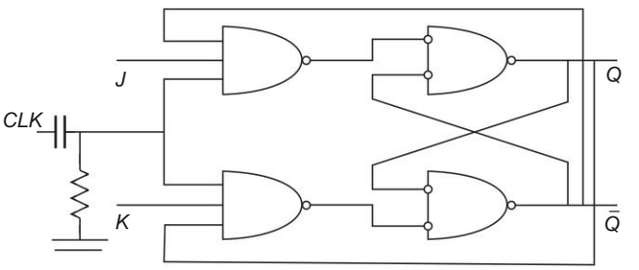


PR	CLR	*CLK*	*D*	*Q*
0	0	*X*	*X*	RACE
0	1	*X*	*X*	1
1	0	*X*	*X*	0
1	1	0	*X*	NC
1	1	1	*X*	NC
1	1	↓	*X*	NC
1	1	↑	0	0
1	1	↑	1	1

Fig. 3.33 (d) Edge-triggered flip-flop with preset PRE and clear CLR

3.7.4 *JK* Flip-Flop

For design of counter, a specially designed flip-flop is required as memory element having toggle state apart from other states set and reset of *D* flip-flop. Figure 3.34(a) shows an edge-triggered *JK* flip-flop which is triggered on the rising edge of a short pulse. In the figure, *JK* flip-flop consists of two inputs *J* and *K* and two outputs (Q and $\overline{Q}$). Both low high *CLK* disables the input gate and prevents latch from changing states. Negative edge-triggered pulse disables the input gate. For changing states, *CLK* should be positive triggered. For positive triggered pulse while *J* is low and *K* is high, Q becomes high and output becomes SET state, and while *J* is high and *K* is low, Q becomes low and output becomes RESET state. For same positive triggered pulse, while both *J* and *K* are high, depending on current state of the output Q, it becomes high or low.



CLK	*J*	*K*	*Q*
Low	*X*	*X*	NC
High	*X*	*X*	NC
↓	*X*	*X*	NC
X	0	0	NC
↑	0	1	1
↑	1	0	0
↑	1	1	Toggle

Fig. 3.34 (a) Edge-triggered *JK* flip-flop

Figure 3.33(b) shows edge-triggered clocked *JK* flip-flop with preset PRE and clear CLR consisting of one input *D* and two outputs, Q and $\overline{Q}$.

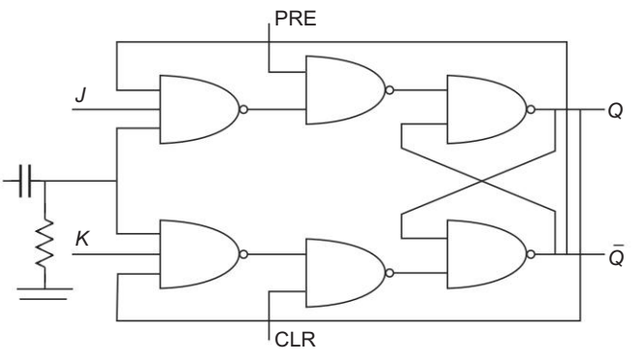


Fig. 3.34(b) Edge-triggered *JK* flip-flop with preset PRE and clear CLR

Figure 3.34(c) shows a master-slave *JK* flip-flop which is triggered on the rising edge of a short pulse. In the figure, *JK* flip-flop consists of two inputs (*J* and *K*) and two outputs (Q and $\overline{Q}$).

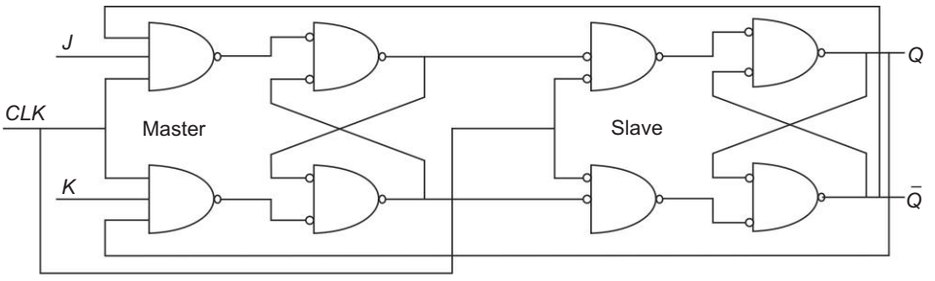


Fig. 3.34 (c) Master-slave *JK* flip-flop

3.7.5 Register

A register is a group of memory elements that simply store a binary word. The simplest register is a buffer register which stores a digital word. Figure 3.35(a) shows a four-bit buffer register consisting of four positive-triggered *D* flip-flops. When the positive-triggered pulse is applied in *CLK* signal then the store word becomes $Q_3\,Q_2\,Q_1\,Q_0 = X_3\,X_2\,X_1\,X_0$.

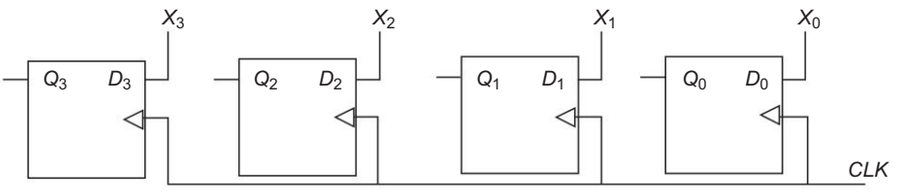


Fig. 3.35 (a) Four-bit buffer register

The register can be modified with stored word by shifting its bits left or right. The bit shifting is essential for certain arithmetic logic operations used in the processor. Figure 3.35(b) shows the left shift register in which D_{in} sets up the left flip-flop, Q_0 sets up the second flip-flop, Q_1 sets up the third, and so on.

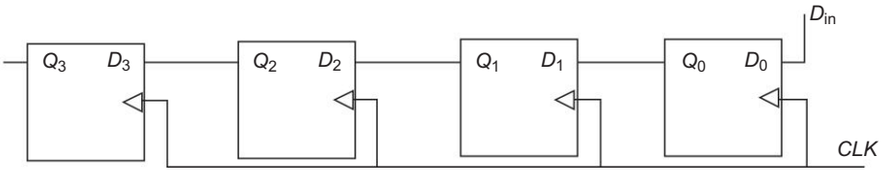


Fig. 3.35 (b) Four-bit left-shift register

Figure 3.35(c) shows the right shift register in which D_{in} sets up its right flip-flop, Q_3 sets up the third flip-flop, Q_2 sets up the second, and Q_1 sets up the first flip-flop.

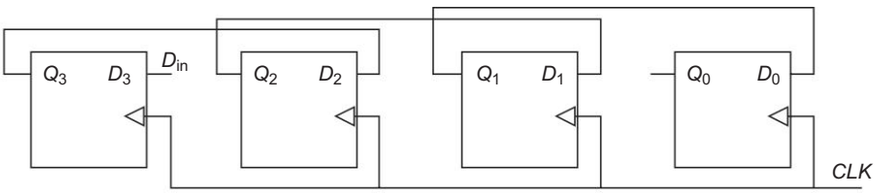


Fig. 3.35 (c) Four-bit right-shift register

Figure 3.35(d) shows a shift register with parallel loading in which it loads *X* bits directly into the flip-flops like a buffer register. If LOAD and SHL are low, the output of the NOR gate is high and the outputs of the flip-flops return to their data inputs. This forces the data to be retained in each flip-flop as positive clock edges arrive. So the contents are stored indefinitely. If LOAD is low and SHL is high, the circuit acts as a left-shift register. If LOAD is high and SHL is low, the circuit acts as a buffer register in which all *X* bits are loaded in flip-flops. The LOAD and SHL are forbidden to be high.

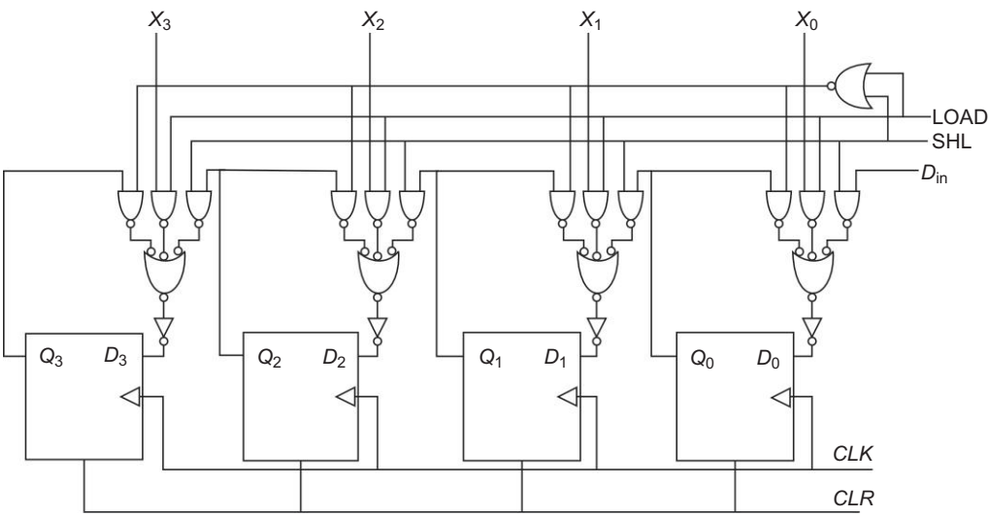


Fig. 3.35(d) Shift register with parallel loading

3.7.6 Counter

A counter is a special kind of register designed for counting the number of clock pulses arriving at its input. It is an electronic equivalent of a binary odometer. Figure 3.36(a) shows a ripple counter built with *JK* flip-flops in which when its clock input receives a negative edge, each flip-flop will toggle with returning of high values of *J* and *K*.

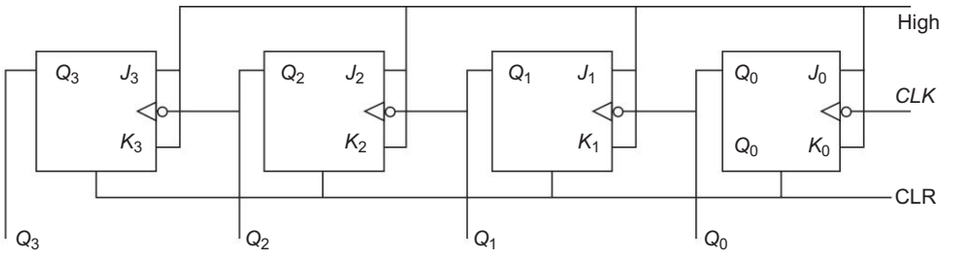


Fig. 3.36 (a) Ripple counter

When the CLR goes high, all the flip-flops reset and $Q = 0000$, where $Q = Q_3\, Q_2\, Q_1\, Q_0$. When CLR returns to low, the counter is ready to go. Since the LSB flip-flop receives each negative clock pulse, Q_0 toggles as per negative clock edge. The remaining flip-flops toggle less often because they receive the negative edges from the proceeding flip-flops. When Q_0 goes from 1 back to 0, Q_1 receives a negative edge and toggles. When Q_1 changes from 1 to 0, Q_1 goes from 1 to 0, the Q_3 flip-flop toggles, and so on. When the first clock pulse is reached, Q_0 becomes 1 and as a result, the output word is $Q = 0001$. When the second clock pulse is reached, Q_0 resets and carries and the next output word is $Q = 0010$. After the third the clock pulse, Q becomes 0011 and finally after the fifteenth clock pulse, Q becomes 1111. Figure 3.36(b) shows a controlled synchronous counter built with *JK* flip-flops in which when its clock input receives positive edge, each flip-flop will toggle with returning of high values of J and K. In the figure, it works with driving all the flip-flops in parallel after getting a high count signal. Because of the simultaneous clocking, the correct binary word appears after one propagation delay rather than four propagation delays in a ripple counter. In the figure, Q_1 flip-flop toggles on positive clock edge and Q_0 is a 1. The Q_2 toggles when Q_1 and Q_0 are 1s. Q_3 flip-flop toggles when Q_2, Q_1 and Q_0 are 1s. So the first positive clock pulse makes $Q = 0001$, the second makes $Q = 0010$ and finally Q becomes 1111.

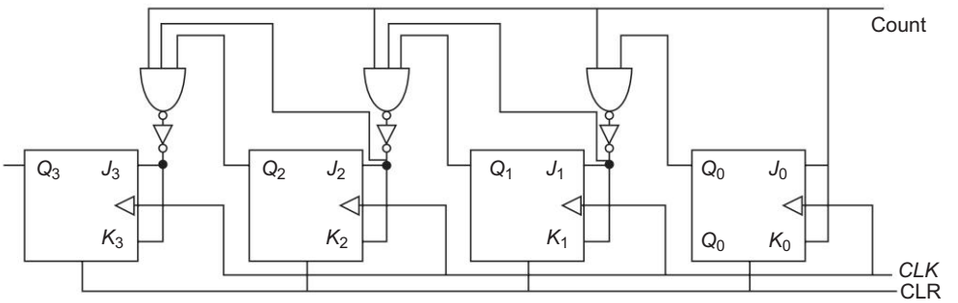


Fig. 3.36 (b) Controlled synchronous counter

Figure 3.36(b) shows a controlled up-down counter built with *JK* flip-flops. When the up signal is low, $\overline{Q_2}$, $\overline{Q_1}$, and $\overline{Q_0}$ are transmitted to the clock signal and as a result, it works as a down counter. While the up signal is high, Q_2, Q_1, and Q_3 are transmitted with clock signal, and it becomes an up counter.

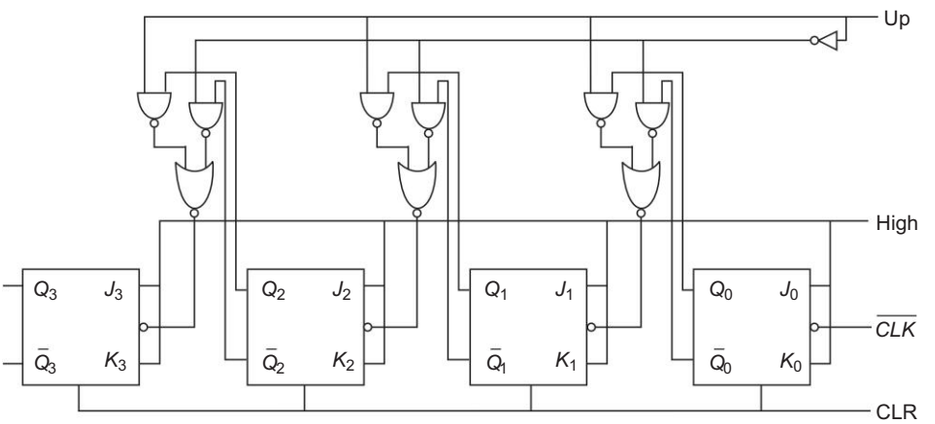


Fig. 3.36(c) Up-down counter

Figure 3.36(d) shows a ring counter built with D flip-flops in which the Q_0 sets up the D_1 input, Q_0 sets up the D_2 input, and so on. When $\overline{CLR}$ goes low then back to high, the initial output word is Q = 0001. The first positive clock edge shifts MSB into LSB position. The other bits shift left one position. As a result, Q becomes 0010. The second positive clock pulse causes Q to be 0100. For the third positive clock pulse, it becomes 1000 and for the fourth positive clock pulse, it becomes 0001.

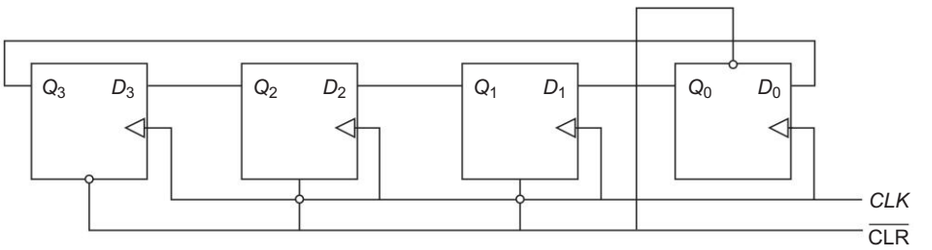


Fig. 3.36(d) Ring counter

3.8 □ CMOS TRANSMISSION GATE

The Transmission Gate (TG) is used in digital CMOS circuit design to pass or not pass a signal. The schematic and logic symbol of the transmission gate (TG) are shown in Fig. 3.37. The gate is made up of the parallel connection of a p- and an n-channel MOSFET. The resistance between the input and the output can be estimated as $R_n \,||\, R_p$.

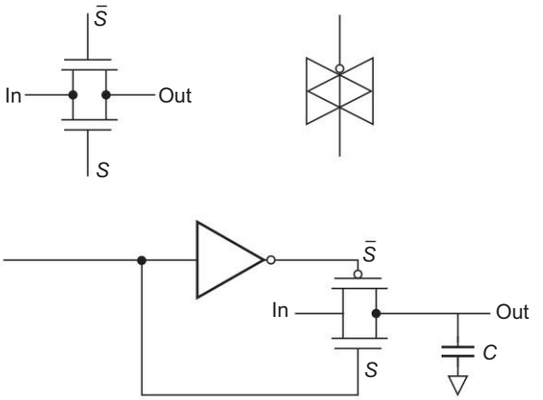


Fig. 3.37 CMOS transmission gate with control signal

Path Selector using Transmission Gate

The circuit shown in Fig. 3.38(a) is a two-input path selector. Logically, the output of the circuit can be written as

$$Z = AS + A\bar{S} \tag{3.57}$$

When the selector signal S is high, A is passed to the output while a low on S passes B to the output. This same idea can be used to implement multiplexers/demultiplexers (MUX/DEMUX). The number of control lines is related to n number of input lines by

$$2^m = n \tag{3.58}$$

where n is the number of inputs (outputs) to the MUX (DEMUX) and m is the number of control lines. A 4-to-1 MUX/DEMUX is shown in Fig. 3.38.

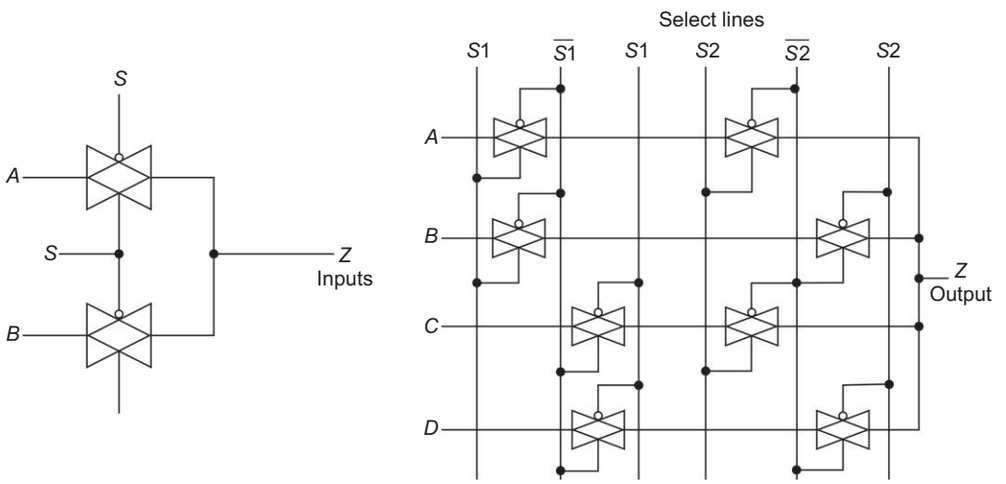


Fig. 3.38 Path selector with 4-to-1 multiplexer

3.9 ☐ DYNAMIC LOGIC GATES

Dynamic, or clocked, logic gates are used to decrease complexity, increase speed, and lower power dissipation. The basic concept used in dynamic logic is the capacitive input of the MOSFET to store a charge and thus remember a logic level for use later. Figure 3.39 shows the *n*-channel pass transistor driving an inverter. If we clock the gate of the pass transistor high, the logic level on the input will be passed to the input of the inverter. If this logic level is a "0," the input of the inverter will be forced to ground while a logic "1" will force the input of the inverter to $V_{DD} - V_{THN}$. When the pass transistor turns on, the input capacitance of the inverter is charged to $V_{DD} - V_{THN}$, or ground, through the pass transistor. As long as this charge is present, the logic value is remembered.

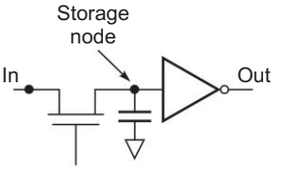


Fig. 3.39 Dynamic circuit with storage capacitance

3.9.1 Non-overlapping Clock Generation

Figure 3.40 shows the schematic of the non-overlapping clock generator. This circuit takes a clock signal and generates a two-phase non-overlapping clock which has logically $\phi_1.\phi_2 = 0$. The amount of overlap is set by the delay through the NAND gate and the two inverters on the NAND gate output. When input clock is going high, ϕ_1 becomes high whereas ϕ_2 becomes low and while the input clock goes low, ϕ_1 and ϕ_2 go low and high respectively. While driving long transmission lines, where the rise time of the signals can be significant, a large number of inverters may be needed to be used.

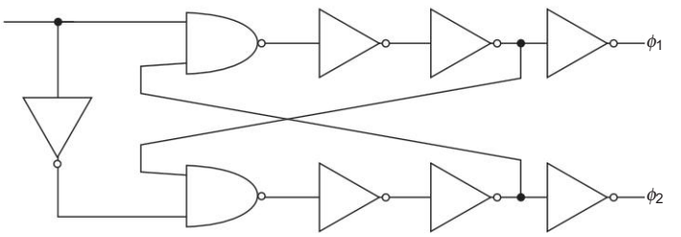


Fig. 3.40 Non-overlapping clock generation circuit

3.9.2 Clocked CMOS Logic

Clocked CMOS (C^2MOS) is used to reduce power dissipation and layout size and to increase speed. Figure 3.41 shows a clocked CMOS latch which performs the dynamic latch operation. When the clock input ϕ_1 is high, the input is inverted and available on the output of the gate. For low-input clock signals, the output is in a high-impedance node, very susceptible to signal feed-through. When the latch is enabled and ϕ_2 is high, the capacitance on the output node is charged. The same leakage in capacitance limits the minimum clock frequency to about 100 Hz. Implementing a shift register requires non-overlapping clocks for adjacent stages.

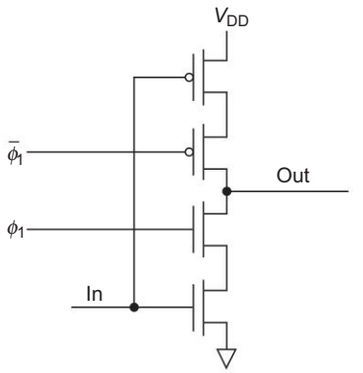


Fig. 3.41 Clocked CMOS logic circuit

1. PE Logic

Figure 3.42 shows the three-input NAND gate with precharge-evaluate logic, or PE logic. The operation of this gate relies on a single clock input. When ϕ_1 is low, the output node capacitance is charged to V_{DD} through M_5. During the evaluate phase, ϕ_1 is high, M_1 is on, and if A_0, A_1, and A_2 are high, the output is pulled low. The logic output is available only when ϕ_1 is high. One disadvantage of PE logic is that the gate logic output is available part of the time and not all of the time as in the static gates. The input capacitance of the PE gate is less than that of the static gate. Each input is connected to a single MOSFET where as the static gate inputs are tied to two MOSFETs. Potentially, the PE gate is then faster and dissipates less power. The output Z is given by

$$Z = \overline{A_0 . A_1 . A_2}$$

$$Z = \overline{A_0 . A_1 + A_2 . A_3 + A_4}$$

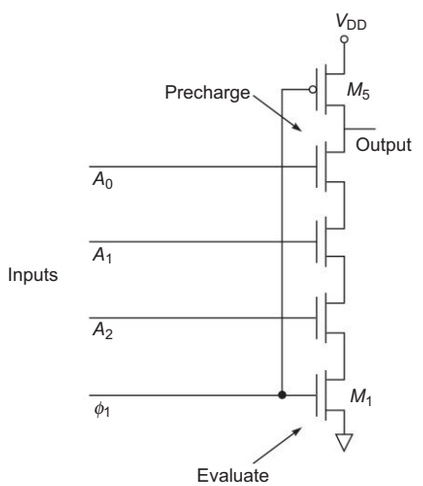


Fig. 3.42 3-input NAND gate using pre-charge and evaluate

The more complicated PE logic function in Fig. 3.43 is written as

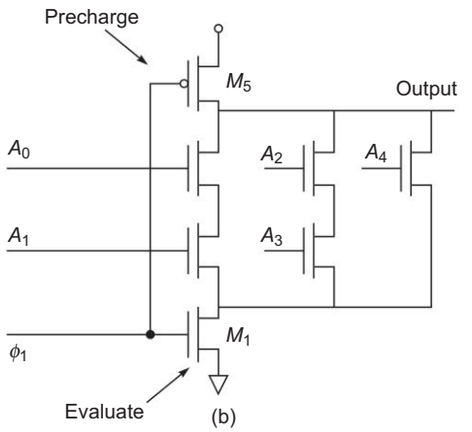


Fig. 3.43 Complicated PE gate using pre-charge and evaluate

2. Domino Logic

Figure 3.44 shows the cascade of PE gates in which during the precharge phase of the clock, the output of each PE gate is a logic high. This high-level output is connected to the input of the next PE gate.

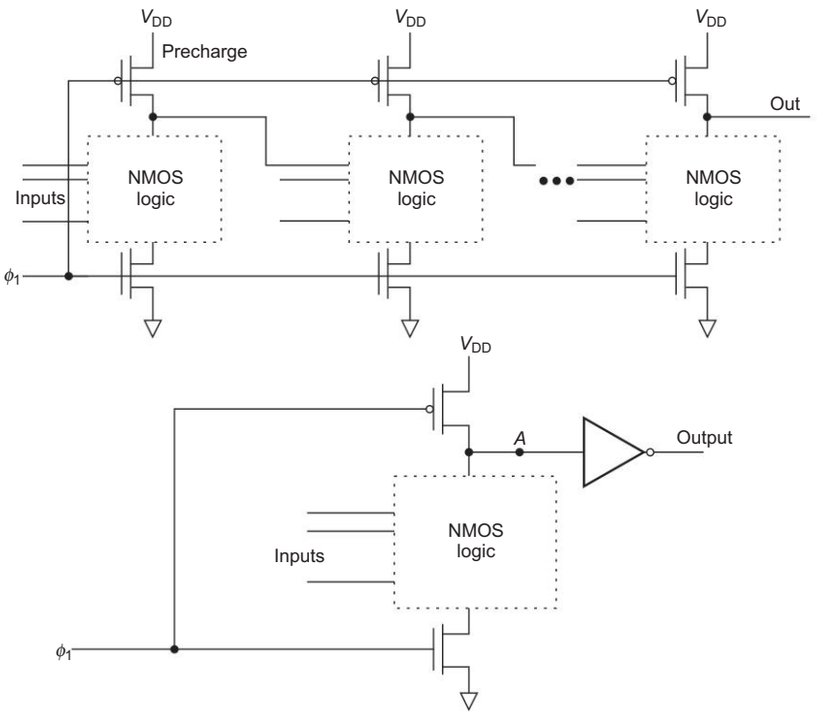


Fig. 3.44 Cascade of PE logic and Domino logic

Since the logic output of the first PE gate during the evaluate phase is a low, this output will turn off any MOSFETs in the second PE gate. However, during the precharge phase, those same MOSFETs in the second PE gate will be turned on. If we have the output voltage of the PE gate low, instead of high, the race condition can be eliminated. The PE gate with the addition of an inverter called Domino logic is shown in Fig. 3.44. The name *Domino* comes from the fact that a gate in a series of Domino logic gates cannot change output states until the previous gate changes states. The change in output of the gates occurs similar to a series of falling dominoes. The inverter used in the Domino gate has the added advantage that it can be sized to drive large capacitive loads.

3. NP Logic (Zipper Logic)

Figure 3.45 shows the logic function using NMOS and PMOS (NP) based logic. Staggering NMOS and PMOS stages eliminate the need for and delay associated with the inverter used in Domino logic, making higher-speed operation possible.

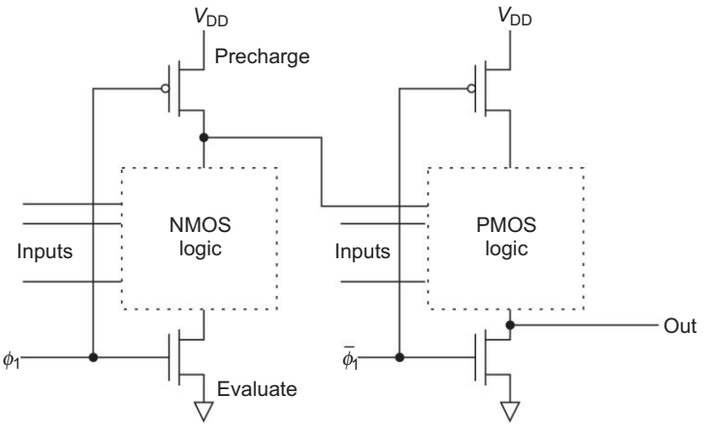


Fig. 3.45 NP logic

3.10 □ MEMORY CIRCUITS

Memory is an important part of digital devices where digital words consisting of bits are stored. There are two types of memory—Read Only Memory (ROM) and Random Access Memory (RAM).

3.10.1 ROM

ROM is the simplest kind of memory where we can read words by applying control signals. *Read* means to make the contents of the memory location appear at the output terminals of the ROM. There are address and contents of the memory location. With address lines, the registers of ROM store the contents. For n address lines, there are 2^n memory locations. Figure 3.46 shows ROM with three address lines. So, the number of memory locations is eight.

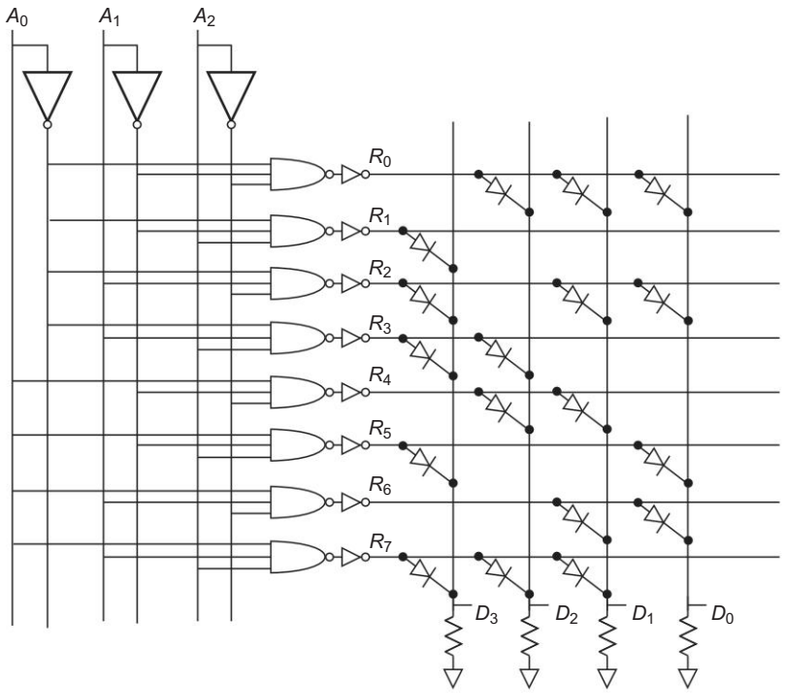


Fig. 3.46 ROM with three address lines

There are programmable ROM (PROM), erasable programmed ROM (EPROM) and electrically erasable programmed ROM (EEPROM). In a PROM, the user is allowed to store data via programming. Once it is stored, the programming becomes permanent. It makes the stored data not erasable.

The EPROM uses MOSFETS to store the data through programming but it can be erasable. The data is erased because it releases the stored charges via ultraviolet passed through window of IC package. Figure 3.47 shows the EPROM in which the modified *n*MOS is used at the interconnection of the

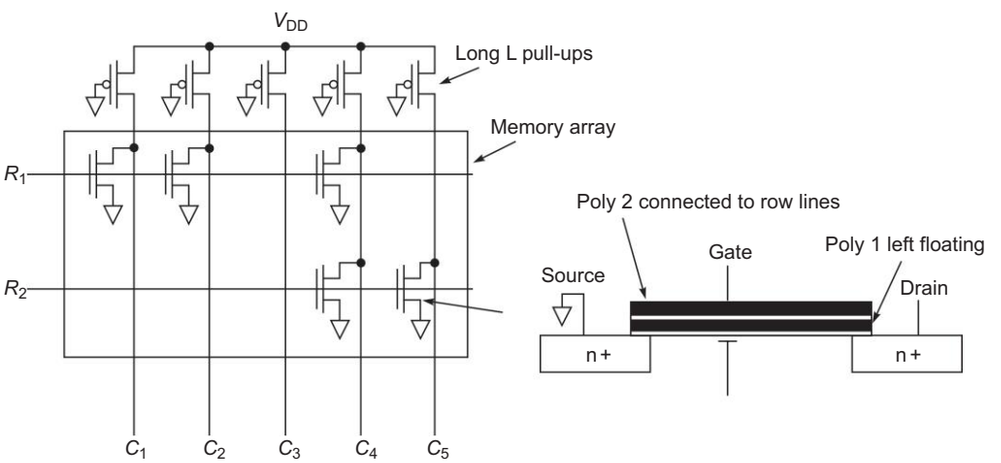


Fig. 3.47 EPROM with nMOSFET EPROM memory cell

column and row lines. In the figure, for row R_1 high results, the column lines C_1, C_2 and C_4 are to be pulled low. Column lines C_3 and C_4 are pulled high through the long L MOSFET loads at the top of the array. If the information that is to be stored in the ROM memory is not known prior to the fabrication, the memory array is fabricated with n channel MOSFETs at every intersection of row and column lines. The modified nMOS shown in Fig. 3.47 has two polylayers—one polylayer is floating and the other is connected through row lines.

The main problem of EPROM is its inability to reprogram. In some applications requiring nonvolatile memory, Electrically Erasable Programmable ROM (EEPROM) is used. In EEPROM, voltage generator is used on chip to generate the large voltage needed to program the EEPROM memory cell where gate oxide is thinner than that of EPROM. The application of high voltage (~ 10 V) to the top polylayer in nMOS memory cell results in conduction between substrate and poly1 layer (called Fowler-N Nordheim tunneling). Logic high in EEPROM is programmed by raising the voltage on poly2 to 10 V while a logic low is programmed by lowering the voltage to –10 V. Flash memory is an example of EEPROM.

3.10.2 RAM

A Random Access Memory (RAM) is a read-write memory equivalent to addressable registers. After applying supply address, one can either read the stored contents of the memory location or write new contents into the memory locations. There are two types of RAM—core RAM and semiconductor RAM memory. Since core RAM is expensive, most applications use semiconductor memory. As per the design of semiconductor memory circuits, there are two types—Static Random Access Memory (SRAM) and Dynamic Random Access Memory (DRAM). These types of memories are termed random access because any bit of data can be accessed at any time.

1. SRAM

In SRAM, as long as power is applied, the data is retained indefinitely. Figure 3.48 shows the block diagram of a SRAM consisting of row lines called **word lines** and the column lines called **bit or digit lines**. In the figure, the row address lines are latched, decoded, and then buffered. The high of a row line selects

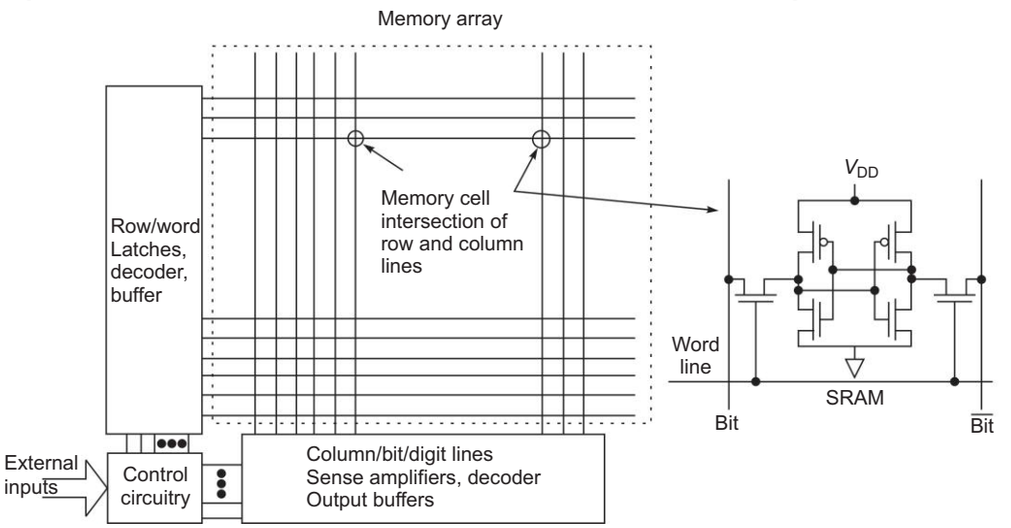


Fig. 3.48 SRAM with SRAM memory cell

the entire row of the array. Since the row line may be long and loaded periodically with the capacitive memory cells, a buffer is needed to drive the line. After a particular row line is selected, the column address is used to decode which of the bits from the row is the addressed information. At this point, data can be read into or out of the array through the column decoder.

The memory array is at the center of the RAM design. The CMOS SRAM memory cell is a cross-coupled connection of inverters. The cross-coupled inverters form a positive feedback circuit, forcing the outputs in opposite directions. When the row line (word line) is low, both pass transistors are off, and the datum in the cell is latched, as long as power is applied to the cell. When the row line goes high, the pass transistors turn on. If the pass transistors width/length ratio is ~ 4 times that of the SRAM cell transistors' width/length ratio, the data on the bit lines (column lines) are stored or written in the cell. If the widths are comparable in size, the effective resistance of the pass transistor is so large as to allow overwriting the cell.

2. DRAM

A dynamic RAM uses MOSFETs and capacitors that store data. Because the capacitor's charge leaks off, the stored data should be recharged every few milliseconds. Like a SRAM, the DRAM consists of row lines called word lines and column lines called bit or digit lines as shown in Figure 3.49.

The basic operations of row lines and column lines are same as SRAM.

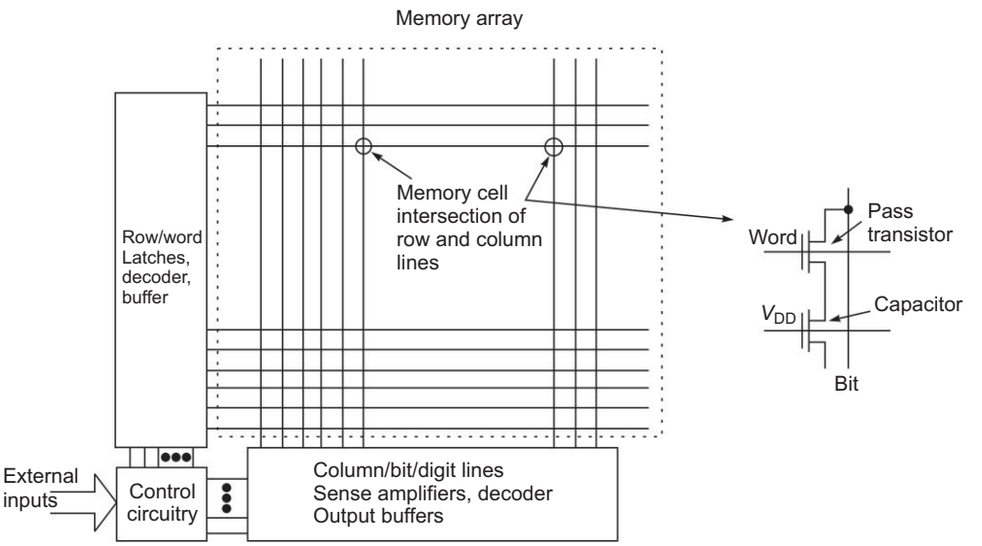


Fig. 3.49 DRAM with SRAM memory cell

The memory cell of a DRAM is different from that of a SRAM. The DRAM memory cell is made up of a pass transistor and a storage capacitor. The operation of DRAM cell is also different from those of a SRAM cell. The capacitor is implemented using a MOSFET operating in the inversion region. The size of the DRAM storage capacitor C_d, is simply the sum of the oxide capacitance of the MOSFET used as a capacitor and the depletion capacitance of the $n+$ implant to substrate. The row/word line and V_{DD} connections to the cell are made using polyl. In order to minimize the layout area of the DRAM array,

the scheme shown in Fig. 3.50 is used. In this configuration, the bit line is shared between two cells. When writing to cell0, the word0 line goes high and the datum on the bit line is written to the capacitor M01. This configuration is called a **folded bit-line configuration**.

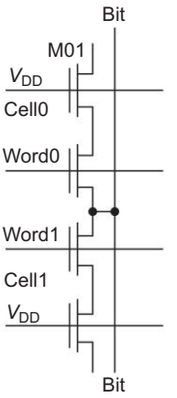


Fig. 3.50 Implementation of DRAM cell

3.11 □ SPECIAL DIGITAL CIRCUITS

There are some special-purpose integrated digital circuits such as Schmitt trigger, multivibrator circuits (both astable and monostable types), and on-chip voltage generators.

3.11.1 The Schmitt Trigger

The Schmitt trigger is a useful circuit used in generating clean pulses from noisy input signals. Figure 3.51 shows CMOS-based Schimitt triggers which are divided into parts depending on whether the output is high or low. If the output is low then *M*6 is on and *M*3 is off, and we are concerned with the *p*-channel portion when calculating the switching point voltages, while if the output is high, *M*3 is on and *M*6 is off, and we are concerned with the *n*-channel portion. Also, if the output is high, *M*4 and *M*5 are on, providing a dc path to V_{DD}. In transfer characteristics of the figure, the curve –A is represented by output from low to high whereas the *B* portion is the output changing from high to low. The transfer characteristics are basically hysteresis of a Schmitt trigger.

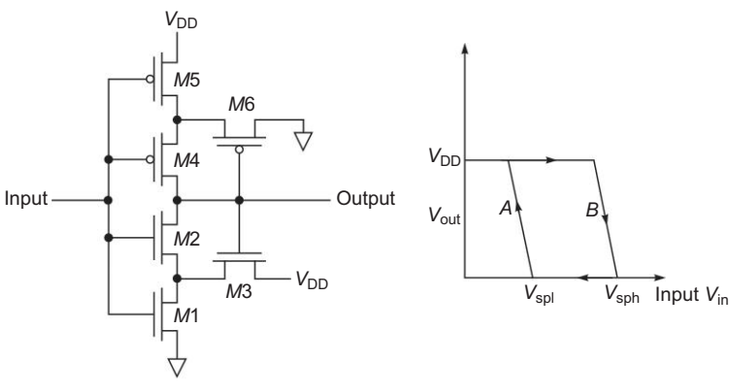


Fig. 3.51 Schematic of CMOS-based Schmitt trigger with transfer characteristics

The threshold voltage of $M1$, given by V_{THN} in this equation, is the zero body bias. It is defined as

$$V_h = V_{sph} - V_{spl}$$

where V_{sph} = High switching point voltage and V_{spl} = Low switching voltage point.
The ratio of device parameters of $M1$ and $M3$ can be written as

$$\frac{\beta_1}{\beta_3} = \left[\frac{V_{DD} - V_{sph}}{V_{sph} - V_{THN3}}\right]^2$$

where V_{THN3} = Threshold voltage of nMOS $M3$. The ratio of device parameters of $M5$ and $M6$ can be written as

$$\frac{\beta_5}{\beta_6} = \left[\frac{V_{spl}}{V_{DD} - V_{sph} - V_{THP6}}\right]^2$$

where V_{THP6} = Threshold voltage of nMOS $M3$.

High-Speed Schmitt Trigger

The effective switching resistances of the MOSFETS in Fig. 3.51 are difficult to reduce without changing the switching-point voltages. Figure 3.52 shows an alternative design of Schmitt trigger. This alternative design uses the basic inverter latch, or SRAM storage cell. Although the propagation times of this Schmitt trigger are less than the previous design, the switching point voltages are more difficult to define, which makes it less useful in some cases.

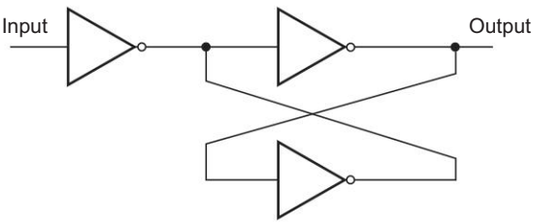


Fig. 3.52 Schmitt trigger with inverter latch

3.11.2 Multivibrator Circuits

Multivibrator circuits are circuits that employ positive feedback. There are three types of multivibrators: astable, bistable, and monostable. Astable multivibrator circuits have unstable states in output (high or low). The oscillators we have discussed are examples of astable multivibrators. The bistable multivibrator has stable states in either the high or low state. Flip-flops and latches are examples of the bistable multivibrator. A monostable multivibrator has a single stable state in output.

1. Monostable Multivibrator

Figure 3.53 shows CMOS-based monostable multivibrator consisting of NOR gate and inverter. Under normal conditions, input voltage is low and the output of the NOR gate, V_2 is high. The voltage V_2 is pulled high through the resistor, and the output of the inverter, V_3 becomes low. When a trigger pulse

is applied, input becomes high, both V_1 and V_2 go to zero volts and the output of the inverter, which is also the output of the monostable, becomes high. The stable low state is (changed to high with application of trigger.)

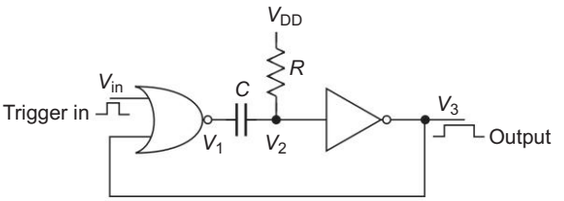


Fig. 3.53 Monostable multivibrator

The time taken to change stable state is

$$t = RC.\ln\left(\frac{V_{DD}}{V_{DD} - V_{sp}}\right) \tag{3.59}$$

where V_{sp} = Switching voltage of the inverter = $\frac{V_{DD}}{2}$.

2. *Astable Multivibrator*

Figure 3.54 shows CMOS based astable multivibrator which has no stable state and thus oscillates. We assume that the output V_3 becomes high, and high output causes V_1 to go high forcing V_2 low. The voltage across the capacitor after switching becomes low and so the output of the astable will become low V_3 is low when $V_1 = V_{sp1}$.

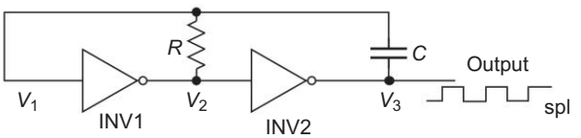


Fig. 3.54 CMOS-based astable multivibrator

3.11.3 Voltage Generators

Voltage pump shown in Fig. 3.55 is used to generate low voltages greater than V_{DD}. The transistor $M1$ is simply used to pull the point A to a voltage of $V_{DD} - V_{THN}$. When the output of the inverter goes high, the potential at point A increases to $V_{DD} + (V_{DD} - V_{THN})$. This turns $M2$ on and charges C_{load} to $2(V_{DD} - V_{THN})$ provided $C_1 >> C_{load}$ and the oscillator frequency allows the capacitors to fully charge or discharge before changing states. In most practical situations, C_{load} and C_1 are comparable in size, and the output of the pump is loaded with a dc load. The result is an output voltage with a start-up time, i.e., V_{out} does not immediately rise to $2(V_{DD} - V_{THN})$ but requires several oscillator cycles to reach steady state.

Figure 3.56 shows the negative voltage pump using p-channel MOSFETs to generate a voltage less than V_{SS}. The capacitors used in Figs. 3.55 and 3.56 can be replaced with MOSFETs. The main

requirement on a MOSFET used as a capacitor is that its V_{GS} remains greater than V_{THN} at all possible operating conditions. When the output of the inverter is high (V_{DD}), the voltage on the gate of C, is $2V_{DD} - V_{THN}$. In both cases (the output of the inverter high and low), $V_{GS} = V_{DD} - V_{THN}$, and the MOSFET remains in the strong inversion region.

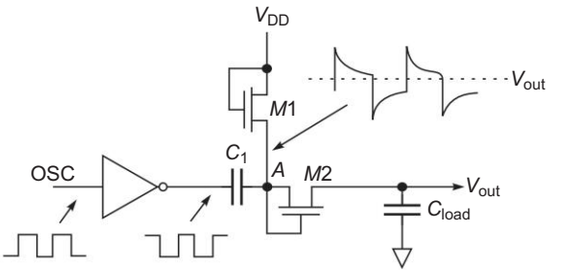


Fig. 3.55 Positive voltage pump for generation of voltage greater than V_{DD}

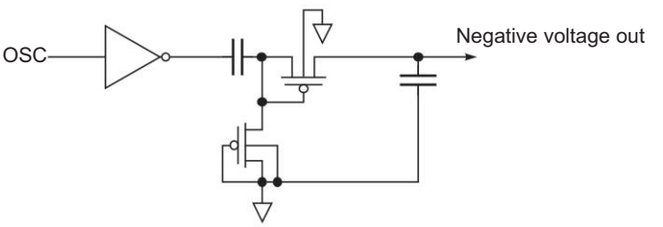


Fig. 3.56 Negative voltage pump for generation of voltage greater than V_{SS}

3.11.4 Generator for Higher Voltages

The voltage pumps discussed in the previous section are limited to voltages less than $2V_{DD}$ and greater than $-2V_{SS}$. Figure 3.57 shows a scheme for generating arbitrarily high voltages (limited by the breakdown voltage of the capacitors or the oxide breakdown of the MOSFETs). When *CLK* is low, the point

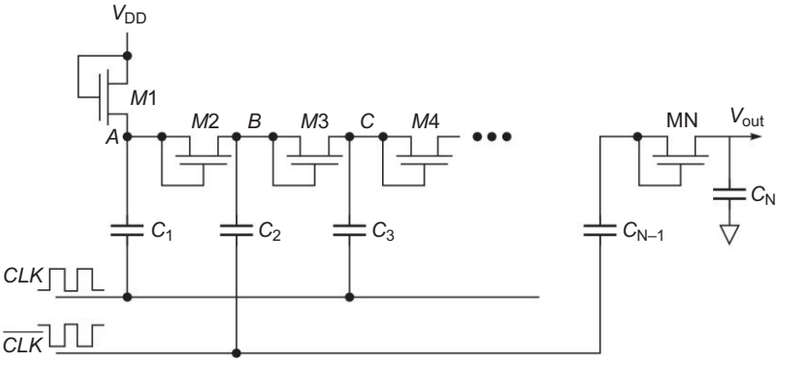


Fig. 3.57 Higher voltage generator by using voltage multiplier

A is pulled, by Ml, to $V_{DD} - V_{THN}$. When CLK goes high, point A swings up to $2V_{DD} - 2V_{THN}$. When CLK is low or $\overline{CLK}$ becomes high, the point B swings up to $3V_{DD} - 2V_{THN}$. When CLK is high or $\overline{CLK}$ becomes low, the point B swings down to $2V_{DD} - 2V_{THN}$. This operation proceeds through the circuit to the last stage of the multiplier. The output of the multiplier swings from $(N + I)V_{DD} - NV_{THN}$ down to $NV_{DD} - NV_{THN}$. For the reduction of the ripple, C_N capacitor can be made larger than the other capacitors.

3.12 ▫ CMOS DIGITAL SYSTEM DESIGN BY USING FSM

The complexity of CMOS digital-circuit design varies from a single inverter circuit to complex computer systems. A successful design requires an abstraction mechanism especially to model sequential circuits over a wide range of complexity. The classical model and design tool for clocked digital system is the Finite State Machine (FSM). Figure 3.58 shows an FSM model which combines combinational logic with a storage mechanism. In the finite state machine as shown in the figure, there are Z storage bits which makes 2^Z possible states and is constrained to be in one of these possible states during each discrete time step. The storage element should be edge-sensitive so that it changes states on clock edge. The next state, an output of the FSM (i.e., the state entered on the next clock edge) is a function of the present state and inputs:

$$\text{Next state} = f(\text{present state, inputs}) \tag{3.60}$$

The combinational logic block is one of the possible implementation of the logical relationships required to specify the complete machine over the set of possible next states:

$$\sum_{i=0,2^z-1} \text{Next state}_i = f(\text{present state, inputs}) \tag{3.61}$$

Each output of this truth table is a logical function for the z bits necessary to specify the next state. These logical functions relate the digital system map directly onto combinational logic. The steps for FSM design-based algorithmic state machine (ASM) are given below:

1. Make the ASM chart as per specifications.
2. Convert the ASM chart into FSM transition table while numbering each state and input conditions giving the next state required in response.
3. Make logical FSM block with FSM transition table.
4. Finally make the circuit by logical FSM block.

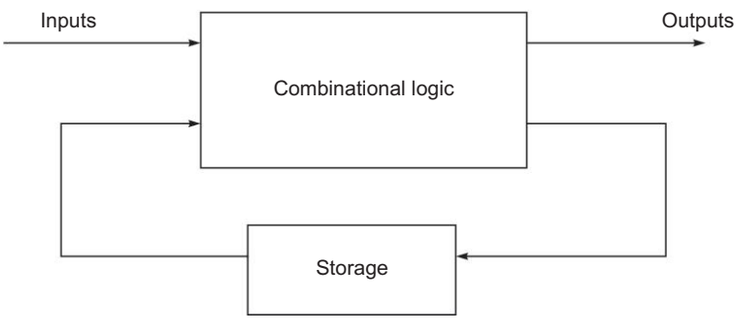


Fig. 3.58 FSM model

3.12.1 Pedestrian-Controlled Traffic-Light System

We consider a pedestrian-controlled traffic light system having four states—green, amber, red, and red amber. The light should normally display green. The light changes amber after a specified time while pressing a request button by a pedestrian. Then, the amber changes to red, and the red state is held for a sufficient time fixed by the timer for the pedestrian to cross the road. After which the light changes to red-amber and then, back to green. This traffic control is called a semi-automatic traffic control.

For an FSM based design for this system, an ASM chart is made as shown in Fig. 3.59(a). In the figure, we start with green. After a button is pressed, green changes to amber and after a specified time, red is displayed while the timer starts. After the requisite time is finished, red changes to red-amber. Then finally, red-amber changes to green.

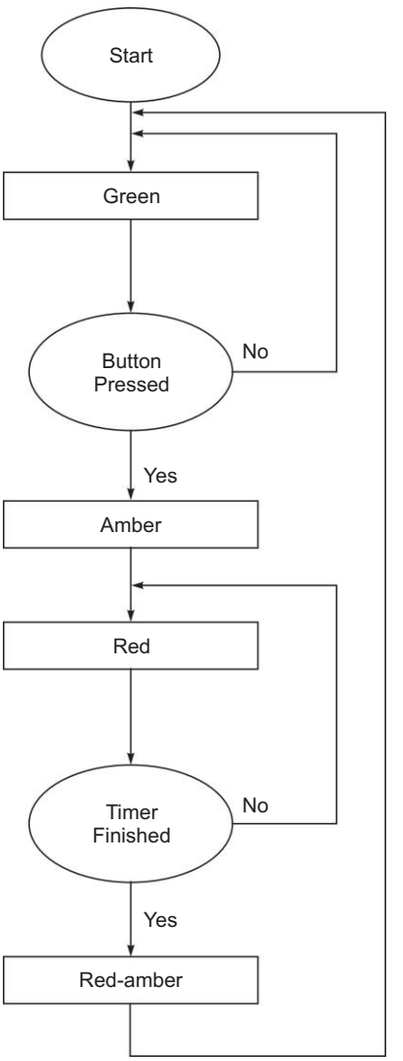


Fig. 3.59 (a) ASM chart of pedestrian-controlled traffic-light system

Table 3.1 represents FSM transition table which is made with an ASM chart. Every line of the transition table specifies a unique set of the state and input conditions, giving the corresponding outputs and the next state in response. The '*x*' symbol represents don't care input and the logical functions defining each bit can be derived from the table by using Karnaugh-map techniques. The different states—green, amber, red and red-amber—are represented by 00, 01, 10, and 11 respectively.

Table 3.1 FSM transition table

Button pressed	Timer finished	This state	Next state	Color traffic light
0	*X*	00	00	Green
1	*X*	00	01	Green
X	*X*	01	10	Amber
X	0	10	10	Red
X	1	10	11	Red
X	*x*	11	00	Red-amber

Using Karnaugh map with FSM transition table, the logic functions are made as follows:

$$D_{\mathrm{A}} = \bar{A}B + A\bar{B} = \overline{\overline{\bar{A}B}.\overline{A.\bar{B}}}$$

$$D_{\mathrm{B}} = \overline{\bar{A}B}..button + A\bar{B}.timeout = \overline{\overline{\bar{A}.B}..button.\overline{A\bar{B}}.timeout}$$

Figure 3.59(b) shows implementation of NAND-based pedestrian-controlled traffic-light system.

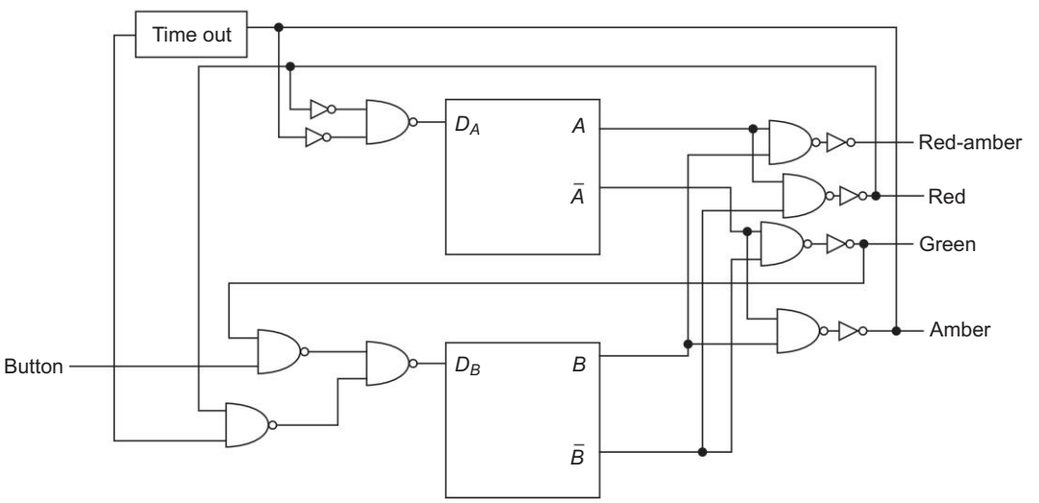


Fig. 3.59 (b) NAND-based pedestrian-controlled traffic-light system

Similarly, we can design a fully automatic traffic control system in which the transition from green to amber and red to red-amber is controlled by a timer, and also, a fully manual traffic control system in which the transition from green to amber and red to red-amber is controlled by button press.

3.12.2 Mealy and Moore Models of FSM

Apart from that, there are two different types of enhancement models of FSM—Mealy model and Moore models in which the output generations are different. In the enhancement model, there are two combinational logic circuits—next combinational and output combinational circuits. In the Mealy model, the output is a function of both present states and the inputs whereas in the moore model, the output is a function of only present states. Figure 3.60 shows block diagrams of Mealy and Moore FSM models. Both models of a sequential circuit have inputs, outputs, and internal states. A circuit may have both types of outputs. The Mealy model of a sequential circuit is referred to as a **Mealy FSM** or **Mealy machine**. The Moore model is referred to as a **Moore FSM** or **Moore machine**. Output y is a function of both input x and the present state of A and.

In a Moore model, the outputs of the sequential circuit are synchronized with the clock, because they depend only on flip-flop outputs that are synchronized with the clock. In a Mealy model, the outputs may change if the inputs change during the clock cycle. Moreover, the outputs may have momentary false values because of the delay encountered from the time that the inputs change and the time that the flip-flop outputs change. In order to synchronize a Mealy-type circuit, the inputs of the sequential circuit must be synchronized with the clock and the outputs must be sampled immediately before the clock edge. The inputs are changed at the inactive edge of the clock to ensure that the inputs to the flip-flops stabilize before the active edge of the clock occurs. Thus, the output of the Mealy machine is the value that is present immediately before the active edge of the clock.

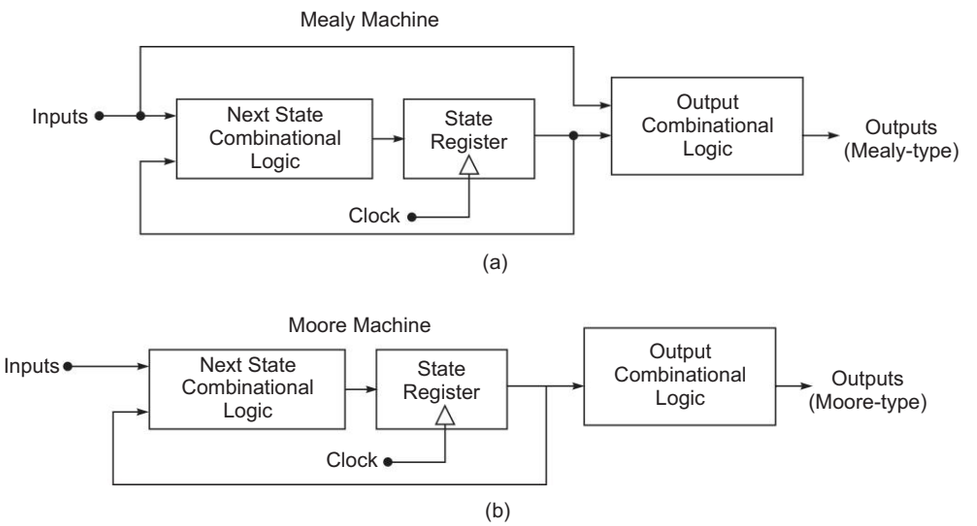


Fig. 3.60 Enhancement model of FSM: (a) Mealy machine (b) Moore machine

3.12.3 Enhancements of FSM Model to Complex System

The FSM model is a general model applicable to complex computer systems. ASM chart-based FSM design is suitable for small digital systems. More complicated systems are made by using micro-coded systems where the FSM storage is through a microcomputer-counter. The execution of instruction of a

microcomputer is controlled by micro-code using conditional micro-branches. Microprogram calls and returns are made by using an extra register. This system can be implemented with an extra data storage and data manipulation can be made from the results through controlling by micro-branches inside the FSM. So it is required to micro-program the storage or counter. A fetch micro-program moves an instruction from program storage to program counter where the FSM input chooses the appropriate execution of the micro-program.

The main difficulty with FSM based micro-programming is to produce the micro-code. The micro-code syntax should be designed for the exact architecture implemented. One of the possible solutions is to design a software tool configured to the architecture before use. High-level languages are designed as compilers to compile efficiently on to the architecture.

3.13 □ BIT SHIFTER CIRCUIT

The *n*-bit shifter is used to shift incoming data by up to *n*-1 place in a right-shift or left-shift direction. All shifts should be on an 'end-around' basis, so that any bit shifted out at one end of a data word will be shifted in at the other end of the word. We can achieve a capability to shift left or right by zero, one, two, or three places by designing a shift circuit. The nature of the shifter has been decided on its implementation. Data could be loaded from the output of the ALU and shifting can be done by shift register. Then, the outputs of each stage of the shift register would provide the required parallel output to be returned to the register. Many designers face the constraints of shifting from any bit to any place in TTL, MSI, and SSl logic design due to use of these shift registers. In the shifter design of TTL, MSI, SSI and VLSI circuits, the shifter should have:

- Input from parallel data bus lines
- Output lines for the shifted data
- Means of transferring input data to output lines with any shift from zero to three bits

In searching for a way to meet these requirements, we can use a switch like MOS pass transistor and transmission gate. To meet these requirements, one can use a switching architecture—crossbar switch. Figure 3.61 shows a 3x3 crossbar switch consisting of 9 MOS pass transistors with three input and output lines. Furthermore, 9 control signals ($sw_{11} - sw_{33}$) are used to shift the data for input lines to output lines. At least one transistor switch should be applied for each input and output line. In the figure, three switches must be operated to drive the crossbar switch. Here, the data bit can be transferred from any input line to other bit by applications of control signal.

The disadvantage of crossbar switches is that four switches have to be operated simultaneously for shifting of data from input lines to output lines. To remove this disadvantage, a barrel shifter is used. Figure 3.62 (a) shows a 3x3-bit barrel shifter circuit with three control signals SW_1, SW_2 and SW_3 . The interbus switches have their gate inputs connected in a staircase fashion in groups of three, and there are now three shift control inputs which must be mutually exclusive in the active state. For the operation of this shifter, one control signal is applied. CMOS transmission gates may be used in place of the simple pass transistor switches.

The structure of the barrel shifter is clearly one of high regularity and generality, and it may be readily represented in stick diagram form. The stick diagram of 3x3-bit barrel shifter using simple *n*-type switches is given in Fig. 3.62(b). The stick diagram clearly conveys regular topology and allows the choice of a standard cell from which complete barrel shifters of any size may be formed by replication of the standard cell.

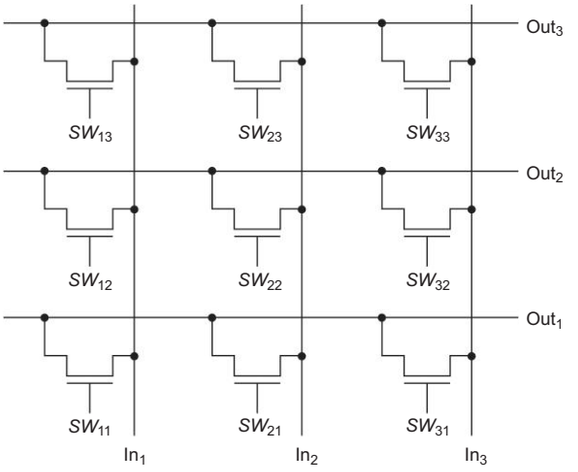


Fig. 3.61 A 3 × 3 crossbar shifter

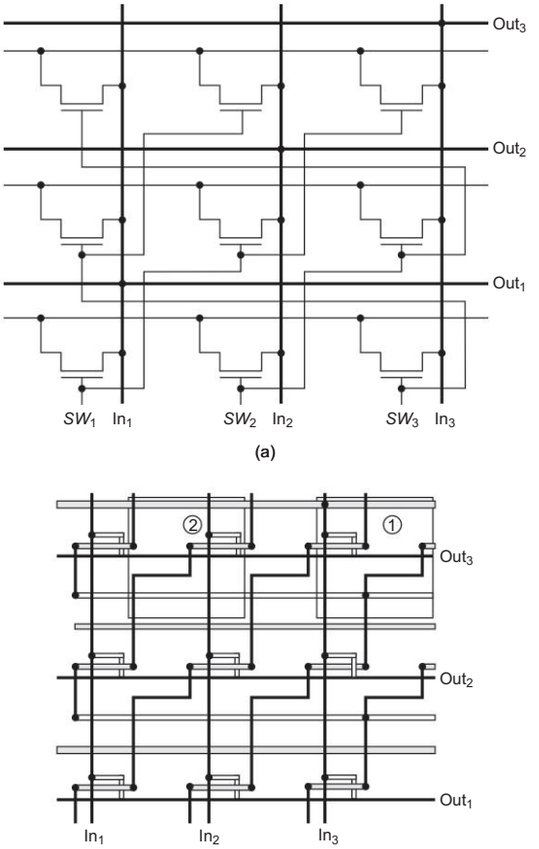


Fig. 3.62 3 × 3 Barrel shifter: (a) NMOS based (b) Its stick diagram

3.14 ◻ COMBINATIONAL PLDS

A combinational Programmable Logic Device (PLD) consists of an AND array and an OR array to provide an AND-OR sum-of-product implementation. There are three types of combinational PLDs, differing in the placement of the programmable connections in the AND-OR array-Programmable Read-Only Memory (PROM), Programmable Array Logic (PAL), and Programmable Logic Array (PLA). Figure 3.63 shows the configuration of the three PLDs. The PROM consists of a fixed AND array acting as a decoder and a programmable OR array. The programmable OR gates implement the Boolean function in sum-of-minterms form. The PAL consists of a programmable AND array and a fixed OR array. The AND gates are programmed to provide the product terms for the Boolean functions, which are logically summed in each OR gate. The PLA is the most flexible PLD in which both the AND and OR arrays can be programmed. The product terms in the AND array may be shared by any OR gate to provide the required sum-of-products implementation. The names PAL and PLA emerged from different vendors during the development of PLDs. The implementation of combinational circuits with PROM was demonstrated in Section 3.10.1.

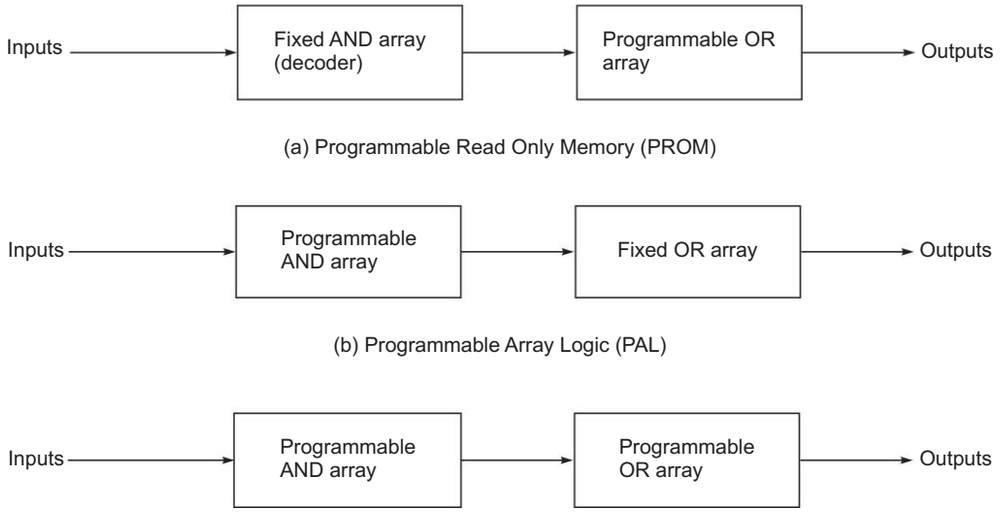


Fig. 3.63 Basic configuration of three PLDs

3.14.1 Programmable Logic Array

The PLA is a programmable logic device with both programmable AND and OR arrays. Like PROM, it is a sum-of-product structure, except that the PLA does not provide full decoding of the variables and does not generate all the min-terms. The decoder is replaced by an array of AND gates that can be programmed to generate any product term which are then connected to OR gates to provide the sum of products for the required Boolean functions. For implementation of PLA with CMOS technology, it is required to convert all AND and OR gates with NAND-NOT (NAND gate followed by NOT gate) and NOR-NOT (NOR gate followed by NOT gate) gate combination respectively. The internal logic of a

PLA with three inputs and two outputs is shown in Fig. 3.63. The diagram uses the array logic graphic symbols for complex circuits. Each input goes through a buffer-inverter combination, shown in the diagram with a composite graphic symbol, that has both the true and complement outputs. Each input and its complement is connected to the inputs of NAND gate followed by an inverter, as indicated by the intersections between the vertical and horizontal lines. The outputs of the NAND-NOT gates are connected to the inputs of each NOR gate followed by NOT gate. The output x of the NOR-NOT gate goes to an XOR gate, where the other input can be programmed to receive a signal equal to either logic '1' or logic '0'. The output is inverted when the XOR input is connected to 1 (since $x \oplus 1 = x'$). The output does not change when the XOR input is connected to 0 (since $x \oplus 0 = x$). The particular Boolean functions implemented in the PLA of Fig. 3.64 are

$$f_1 = AB + AC + \bar{A}\bar{B}\bar{C} = \overline{\overline{AB} + \overline{AC} + \overline{A}\overline{B}\overline{C}}$$

$$f_2 = \overline{(AC + B\bar{C})} = \overline{\overline{AC} + \overline{B}\overline{C}}$$

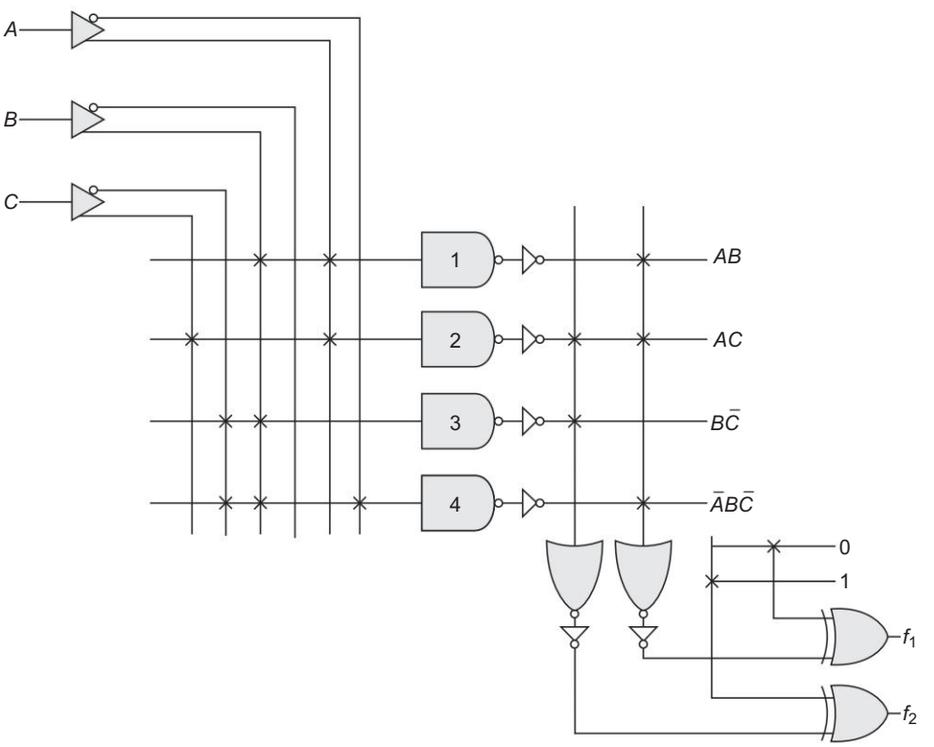


Fig. 3.64 PLA with three inputs and two outputs

The product term NAND-NOT gate is determined from the inputs whose cross points are connected and indicated by 'x'. The output of an NOR-NOT gate gives the logical sum of the selected product terms. The output may be complemented or left in its true form, depending on the logic being realized.

The fuse map of a PLA can be specified in a tabular form. For example, the programming table that specifies the PLA of Fig. 3.63 is listed in Table 3.2. The PLA programming table consists of three sections. The first section lists the product terms numerically. The second selection specifies the required paths between inputs and NAND-NOT gates. The third section specifies the paths between the NAND-NOT and NOR-NOT gates. The paths between the inputs and the NAND-NOT gates are specified under the column head "Inputs" in the programming table. A '1' in the input column specifies a connection from the input variable to the NAND-NOT gate. A '0' in the input column specifies a connection from the complement of the variable to the input of the AND gate. A dash specifies a blown fuse in both the input variables and its complement. It is assumed that an open terminal in the input of an AND gate behaves like a 1.

Table 3.2 PLA programming table

Product term		Inputs			Outputs	
					(C)	*(T)*
		A	*B*	*C*	f_1	f_2
$\overline{\overline{AB}}$	1	1	1		1	–
$\overline{\overline{AC}}$	2	1		1	1	1
$\overline{\overline{B\overline{C}}}$	3		1	0	–	1
$\overline{\overline{\overline{A}\overline{BC}}}$	4	0	0	0	1	–

The paths between the NAND-NOT and NOR-NOT gates are specified under the column head "Outputs." The output variables are marked with 1's for those product terms which are included in the function. Each product term that has a 'I' in the output column requires a path from the output of the NAND-NOT gate to the input of the NOR-NOT gate. Those marked with a dash specify a blown fuse. It is assumed that an open terminal in the input of a NOR-NOT gate behaves like a '0'. Finally, a *T* (true) output dictates that the other input of the corresponding XOR gate be connected to 0, and a *C* (Complement) specifies a connection to 1. The size of a PLA is specified by the number of inputs, the number of product terms, and the number of outputs. A typical integrated-circuit PLA may have 16 inputs, 48 product terms, and 8 outputs. For *n* inputs, *k* product terms, and *m* outputs, the internal logic of the PLA consists of *n* buffer-invener gates, *k* AND gates, *m* OR gates, and *m* XOR gates. There are $2n \times k$ connections between the inputs and the AND array, $k \times m$ connections between the AND and OR arrays, and *m* connections associated with the XOR gates.

3.14.2 Programmable Array Logic

The PAL is a programmable logic device with a programmable AND array and a fixed OR array. Because only the AND gates are programmable, the PAL is easier to program than, but is not as flexible as, the PLA. For implementation of PAL with CMOS technology, it is required to convert all AND and

OR gates with NAND-NOT and NOR-NOT gate combination respectively. Figure 3.65 shows the logic configuration of a typical PAL with four inputs and four outputs. Each input has a buffer-inverter gate and each output is generated by a fixed OR gate. There are four sections in the unit, each composed of a NAND-NOT and NOR-NOT array that is *three wide*—the term used to indicate that there are three programmable NAND-NOT gates in each section and one fixed NOR-NOT gate. Each NAND gate has 10 programmable input connections, shown in the diagram by 10 vertical lines intersecting each horizontal line. The horizontal line symbolizes the multiple-input configuration of the NAND-NOT (equivalent to AND) gate. One of the outputs is connected to a buffer-inverter gate and then fed back into two inputs of the NAND-NOT gates. A typical commercially available PAL integrated circuit may have eight inputs, eight outputs. And eight sections, each consisting of an eight-wide AND-OR (NAND-NOT and NOR-NOT) array. The output terminals are sometimes driven by three-state buffers or inverters.

In designing with a PAL, the Boolean functions must be simplified to fit into each section. In PLA, a product term cannot be shared among two or more OR gates. So, each function can be simplified by itself, without regard to common product terms. The number of product terms in each section is fixed. As an example of using a PAL in the design of a combinational circuit in Fig. 3.65, we consider the following Boolean functions, given in sum-of-minterms form:

$$f_1(A, B, C. D) = \sum (2.\ 12.\ 13)$$
$$f_2(A, B.\ C.\ D) = \sum (7.\ 8, 9, 10, 11, 12, 13.\ 14.\ 15)$$
$$f_3(A.\ B, C, D) = \sum (0.\ 2, 3, 4, 5, 6, 7, 8.\ 10.\ 11.\ 15)$$
$$f_4(A.\ B, C, D) = \sum (1.\ 2.\ 8.\ 12.\ 13)$$

Simplifying the four logic functions, a minimum number of terms are obtained in the following Boolean functions:

$$f_1 = AB\overline{C} + \overline{A}\overline{B}C\overline{D} = \overline{\overline{AB\overline{C}} + \overline{\overline{A}\overline{B}C\overline{D}}}$$

$$f_2 = A + BCD = \overline{\overline{A} + \overline{BCD}}$$

$$f_3 = \overline{A}B + CD + \overline{B}\overline{D} = \overline{\overline{A\overline{B}} + \overline{CD} + \overline{B\overline{D}}}$$

$$f_4 = AB\overline{C} + \overline{A}\overline{B}C\overline{D} + A\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D = f_1 + + A\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D$$
$$= \overline{\overline{AB\overline{C}} + \overline{\overline{A}\overline{B}C\overline{D}} + \overline{A\overline{C}\overline{D}} + \overline{\overline{A}\overline{B}\overline{C}D}}$$

The PAL programmable table for four Boolean functions is shown in Table 3.3 where the inputs of AND gates need be programmed. The table is divided into four sections with three product terms in which two product terms are used for the first two sections and others use more than two product terms. The fuse map of Table 3.3 is given in Fig. 3.65.

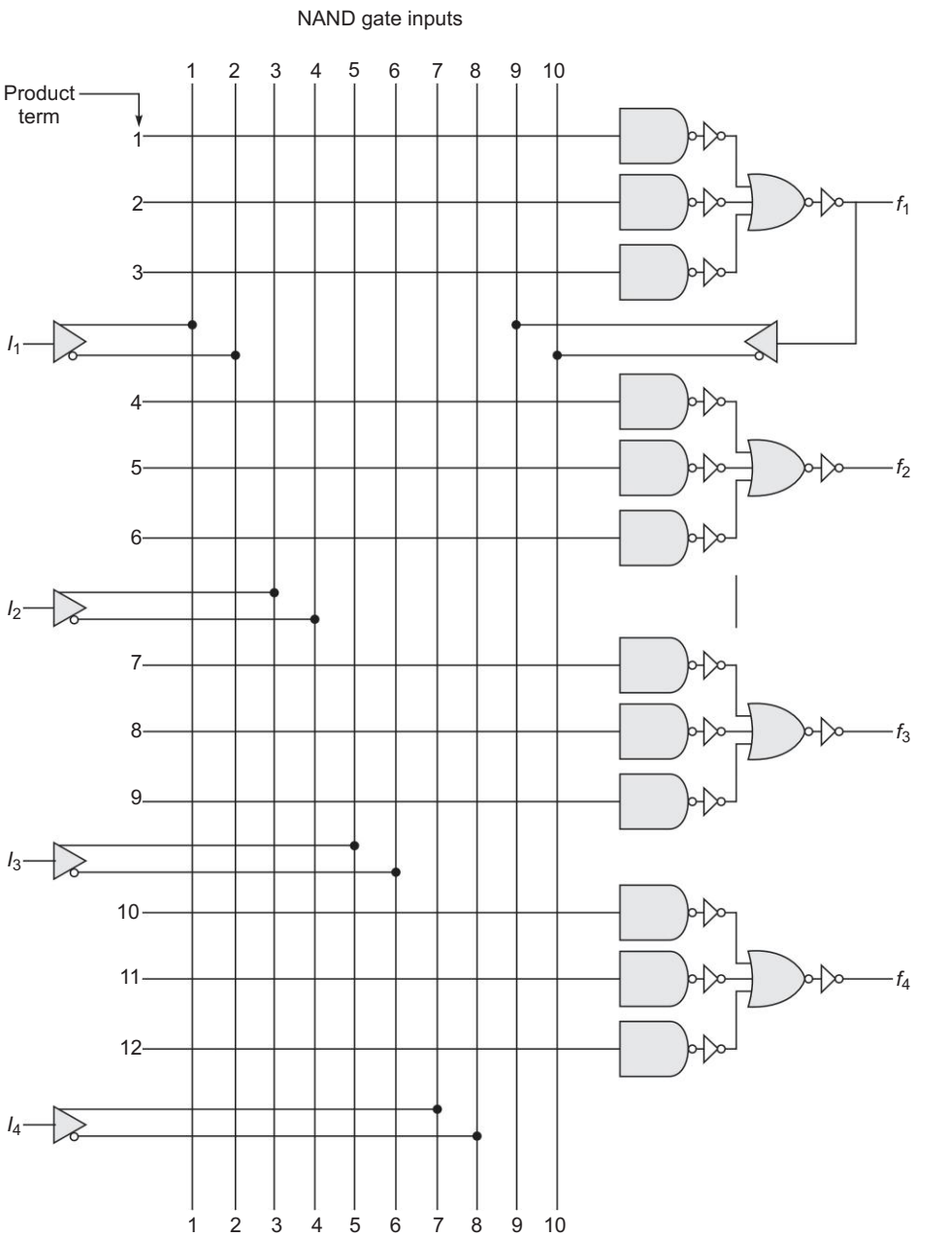


Fig. 3.65 PAL with four inputs, four outputs and three NAND-NOR structures

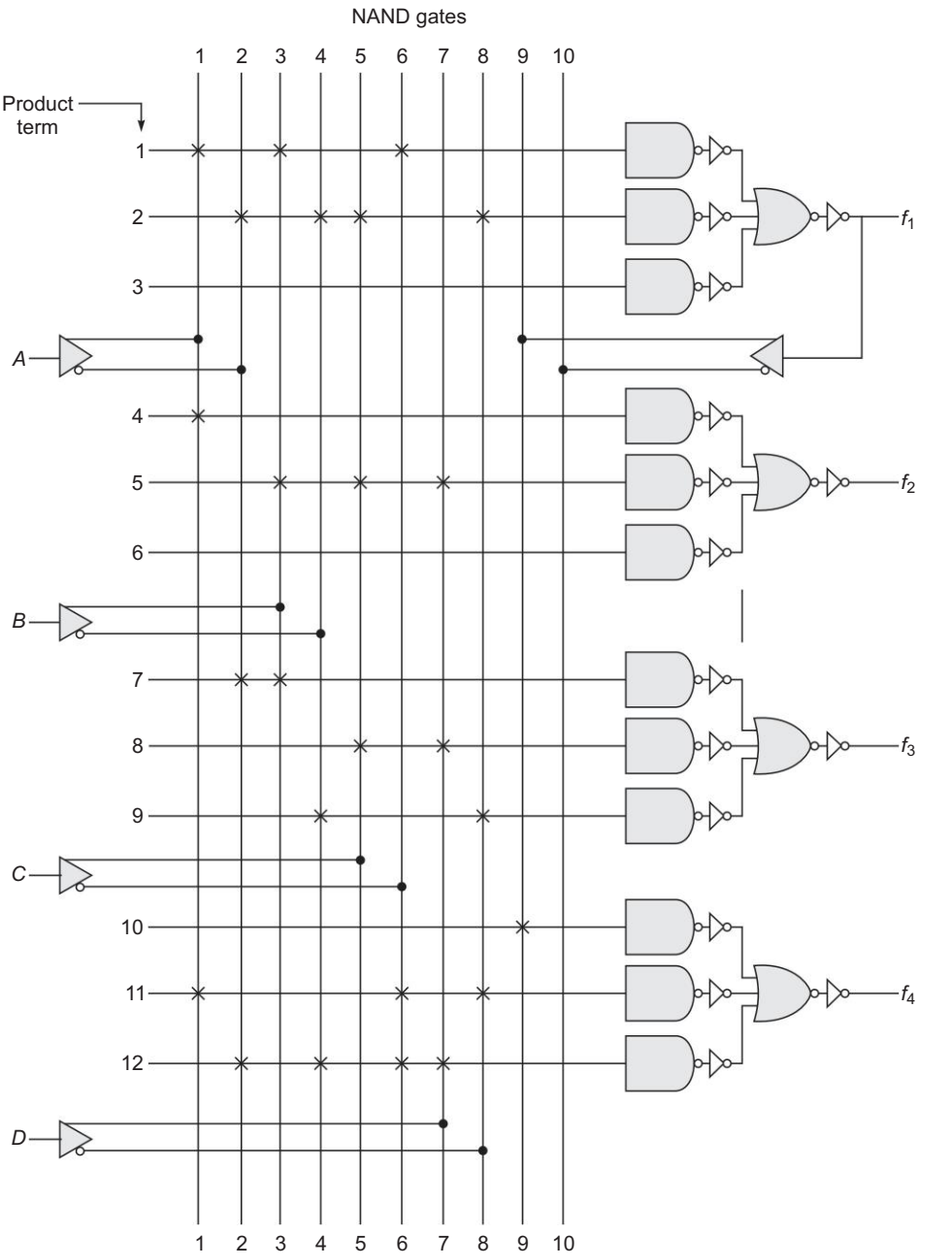


Fig. 3.65 PAL fuse map for Table 3.3

Table 3.3 PAL Programmable Table for Four Boolean Functions

Product Term	AND Inputs					Outputs
	A	B	C	D	f_1	
1	1	1	0			$f_1 = ABC' + A'B'CD'$
2	0	0	1	0		
3						
4	1					$f_2 = A + BCD$
5		1	1	1		
6						
7	0	1				$f_3 = A'B + CD + B'D'$
8			1	1		
9		0		0		
10					1	$f_4 = w + AC'D' + A'B'C'D$
11	1		0	0		
12	0	0	0	1		

REFERENCES

3.1 R.L. Geiger, P.E. Allen, and N.R. Strader, *VLSI-Design Techniques for Analog and Digital Circuits,* McGraw-Hill Publishing Co., 1990, ISBN 0-07-023253-9.

3.2 N.H.E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design,* Addison-ed., 1993, ISBN 0-201-53376-6.

3.3 M.I. Elmasry, *Digital MOS Integrated Circuits II,* IEEE Press, 1992, ISBN 0-87942-275-0, IEEE order number: PC0269-1.

3.4 J.P. Uyemura, *Circuit Design for Digital CMOS VLSI,* Kluwer Academic Publishers, 1992.

3.5 M. Shoji, *CMOS Digital Circuit Technology,* Prentice-Hall, 1988, ISBN 0-13-138850-9.

3.6 J. Yuen and C. Svensson, "New Single-Clock CMOS Latches and Flipflops with Improved Speed and Power Savings," *IEEE Journal of Solid-State Circuits,* Vol. 32, No. l, pp. 62–69, 1997.

3.7 T. Hamada, "A Split-Level Diagonal Bit-Line (SLDB) Stacked Capacitor Cell for 256Mb DRAMs," 7992 *IEDM Technical Digest,* pp. 799–802.

3.8 T. Mohihara et al., "Disk-Shaped Capacitor Cell for 256Mb Dynamic Random-Acess Memory," *Japan Journal of Applied Physics,* Vol. 33, Part 1, No. 8, pp. 4570–4575, August 1994.

3.9 J.H. Ahn et al., "Micro Villus Patterning (MVP) Technology for 256Mb DRAM Stack Cell," *1992 Symposium on VLSI Technical Digest of Technical Papers,* pp. 12–13.

3.10 K. Sagara et al., "Recessed Memory Array Technology for a Double Cylindrical Stacked Capacitor Cell of 256M DRAM," *IEICE Trans. Electron.,* Vol. E75-C, No. 11, pp. 1313–1322, November 1992.

3.11 R.D. Pashley and S.K. Lai, "Flash Memories: The Best of Two Worlds," *IEEE Spectrum,* 1989, pp. 30–33.

3.12 D. Frohman-Bentchkowsky, "FAMOS-A New Semiconductor Charge Storage Device," *Solid-State Electronics,* Vol. 17, pp. 517–529, 1974.
3.13 R. Gregorian, K.W. Martin, and G.C. Temes, "Switched-Capacitor Circuit Design," *Proceedings of the IEEE,* Vol. 71, No. 8, August, pp. 941–966, 1983.
3.14 J.F. Dickson, "On-Chip High-Voltage Generation in MNOS (sic) Integrated Circuits Using an Improved Voltage Multiplier Technique," *IEEE Journal of Solid State Circuits,* Vol. SC-11, No. 3, June 1976.
3.15 P. Zagar, *4 MEG DRAM Tutorial-Course Notes,* Micron Semiconductor, Boise, Idaho.
3.16 J. Wu, Y. Chang, and K. Chang, "1.2V CMOS Switched-Capacitor Circuits," *ISSCC,* 1996.

EXERCISES

3.1 Estimate the delay through seven *n*-channel MOSFETs connected in series, similar to Fig. P3.1.

The size of the MOSFETs is $\frac{10}{2}$.

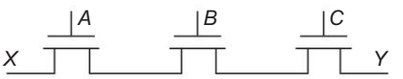


Fig. P.3.1

3.2 Repeat Problem 3.1 using *p*-channel MOSFETs.
3.3 Estimate t_{PHL} and t_{PLH} for the circuits shown in Fig. P3.2 when the capacitor is increased to 1 pF. Verify your hand calculations with SPICE.

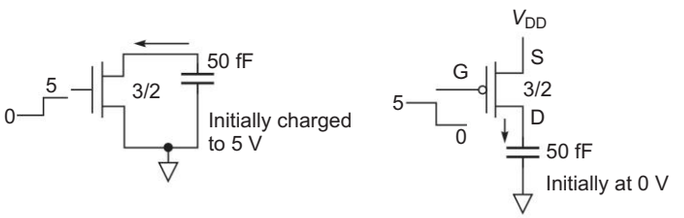


Fig. P3.2

3.4 Calculate R_n and R_p and process characteristics, time constant *s* with MOSFETs having 0.9 micron width and 0.6 micron length.
3.5 Design and simulate the dc characteristics of an inverter with V_{SP} approximately equal to V_{THN}. Estimate the resulting noise margins for the design.
3.6 Estimate the oscillation frequency of a 31-stage ring oscillator using minimum-size inverters.
3.7 Design a buffer to drive a 50 pF capacitive load from an inverter with size of $\frac{150}{50}$. The $t_{PHL} + t_{PLH}$ should be less than 10 ns.
3.8 Design a layout and simulate the operation of a CMOS AND, OR gate and Exclusive OR gate with a V_{SP} of approximately 1.5 V.

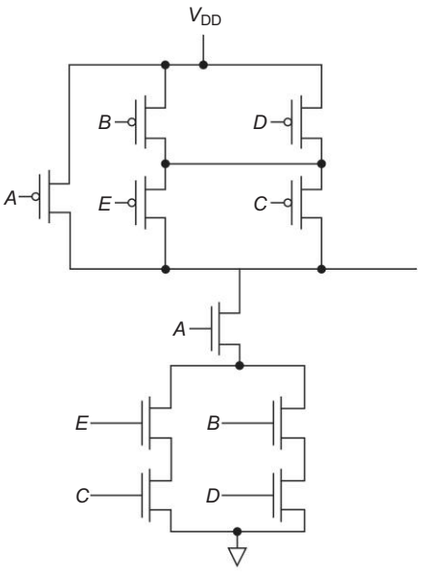


Fig. P3.3

3.9 Design CMOS-based 8-bit Gray-to-Binary coder and decoder, 14-to-1 multiplexer, CMOS-based 4-input NOR and 4-input NAND gate circuit

3.10 Design and simulate the operation of a CMOS AOI half adder and CMOS AOI exclusive NOR gate circuit using static logic gates.

3.11 Sketch the schematic of an OR gate with 20 inputs. Comment on your design.

3.12 Sketch the CMOS schematic of a static logic gate that implements $A + BC \bullet D$. Estimate the worst-case delay through the gate when driving a 50 pF load capacitance.

3.13 What logic function does the circuit of Fig. P3.3 implement?

3.14 Design a tri-state buffer that has propagation delays under 20 ns when driving a 1 pF load. Assume that the maximum input capacitance of the buffer is 100 pF.

3.15 Sketch the schematic of a three-input XOR gate implemented in AOI logic.

3.16 Calculate the switching-point voltage of the gate shown in Fig. P3.4. What logic function does this circuit implement?

3.17 Estimate the minimum and maximum output voltages for the gate of Fig. P3.4

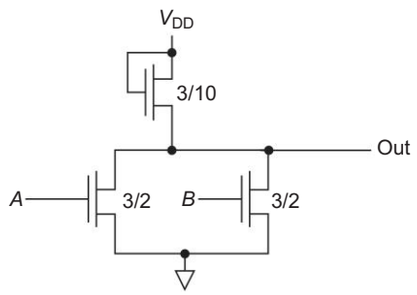


Fig. P3.4

3.18 Design a 3-to-1 four-bit word multiplexer by using CMOS device.

3.19 Design a 10-bit parity generator circuit by using CMOS device.

3.20 Design a 5-bit 2's complement substrater–adder word multiplexer by using CMOS device.

3.21 If we increase the width of the *n*-channel pass transistor, what happens to the delay times? The gate of the pass transistor is driven from some other logic on the chip. What happens to the capacitance seen by this logic when we increase the width of the pass transistor?

3.22 Design a half-adder circuit using TGs.

3.23 Estimate and simulate the delay through 10 TGs (assume minimum size) connected to a 100 pF load capacitance.

3.24 Sketch the schematic of an 8 to 1 DEMUX using *n*-channel pass transistors. Estimate the delay through the DEMU*X* when the output is connected to a 50 pF load capacitance.

3.25 Sketch an *SR* FF made with NOR gates using minimum-size MOSFETs. Show all four logic transitions possible for the FF.

3.26 Draw the clocked *D* FF by using CMOS. Show the FF clocking in a logic 1 and 0. What are the set-up and hold times for your design?

3.27 Design the FF and explain the operation of the FF shown in Fig. P3.5.

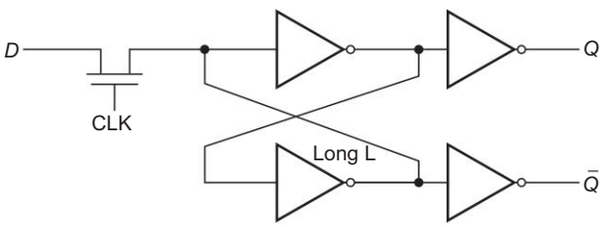


Fig. P3.5

3.28 Explain the operation of the non-overlapping clock generator circuit made using minimum-size MOSFETs in Fig. P3.6. Assume that the input clock signal is running at 50 MHz. Show how both ϕ_1 and ϕ_2 are non-overlapping.

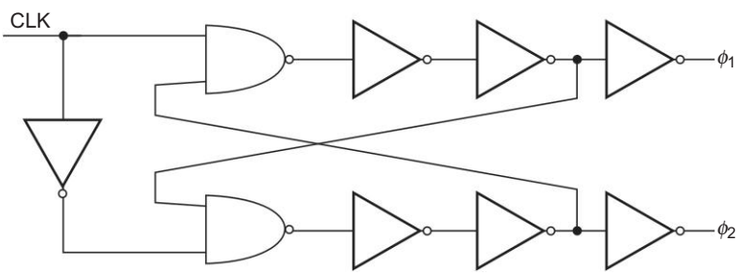


Fig. P3.6

3.29 Design a PE gate that will implement the logical function $F = ABC + DE$ and explain its operation.

3.30 Implement an XOR gate using Domino logic. Explain the operation of the resulting implementation.

3.31 The circuit shown in Fig. P3.7 is the implementation of a high-speed adder cell (1 bit). What type of logic was used to implement this circuit? Using timing diagrams, describe the operation of the circuit.

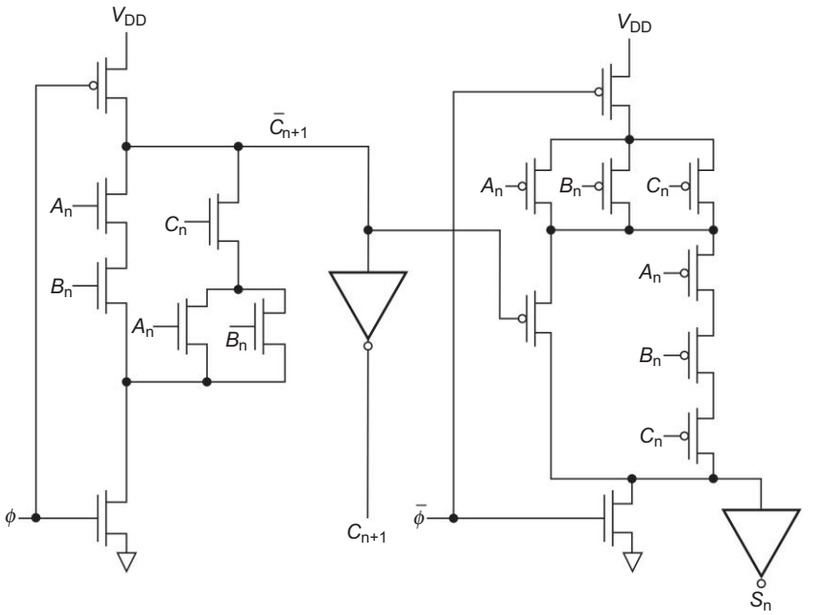


Fig. P3.7

3.32 Discuss the design of a two-bit adder using the adder cell of Fig. P3.7. If a clock, running at 20 MHz, is used with the two-bit adder, how long will it take to add two words? How long will it take if the word size is increased to 32 bits?

3.33 Design a full-adder circuit using PE logic.

3.34 Design an NP logic half-adder cell.

3.35 Show that the circuit in Fig. P3.8 is an edge-triggered flip-flop in which a single phase clock is used.

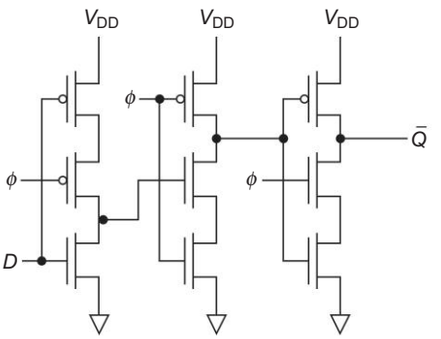


Fig. P3.8

3.36 Estimate the capacitance that the pass transistor of Fig. P3.9 contributes to the word line in Fig. P3.9a. Assume that the MOSFET is minimum size, that is, 3 mm by 2 μn.

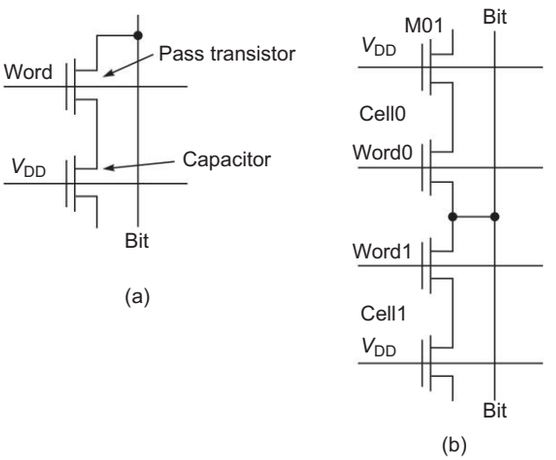


Fig. P3.9

3.37 Explain why it is important to equilibrate the bit lines prior to reading the data from the memory cells.

3.38 Which DRAM architecture, i.e., folded or open, results in the smallest memory bit (mbit) layout size? What is the major advantage of using the folded bit line architecture over the open?

3.39 Design a Schmitt trigger with $V_{SPH} = 3$ and $V_{SPL} = 2.5$ V.

3.40 Estimate t_{PHL} and t_{PLH} for the Schmitt trigger driving a 100 pF load capacitance.

3.41 Design a Schmitt trigger-based oscillator with an output frequency of 10 MHz and its operation.

3.42 Design an astable multivibrator with an output oscillation frequency of 20 MHz by using CMOS. What is the V_{sp} of the inverters?

3.43 Design a nominally 7 V voltage generator that can supply at most 1 uA of dc current (7 MEG resistor).

3.44 Design a fully manual traffic-control system with button press having four states—green, amber, red, and red amber by using FSM model.

3.45 Design a fully automatic traffic-control system with timer time out having four states—green, amber, red, and red amber by using an FSM model.

4

CMOS-based Analog Circuit

In an analog integrated circuit, there are basic passive components—resistance and capacitance—and basic building blocks—current source, sinks, and amplifiers. Using basic building blocks and components, we can design analog circuits. In this chapter, we shall discuss analog components, analog basic building blocks and analog-circuits-based CMOS device.

4.1 □ PASSIVE COMPONENTS

Passive components are operated without external power. In this section, we discuss passive components such as resistance and capacitance which can be used in CMOS based analog circuit.

4.1.1 Sheet Resistance

Figure 4.1 shows a uniform slab of conducting material ρ of width W, thickness t, and length L acting as a sheet resistance. The value of resistance is given by

$$R_{AB} = \rho L/A = \rho L/(t.W) \tag{4.1}$$

where $A = t.W$ = area of cross section of slab. The sheet resistance is defined as a resistance of square slab ($L = W$) which is $R_s = \rho/t$. The same concept can be applied to a MOSFET transistor. The channel resistance of a MOSFET can be written as

$$R = ZR_S = \frac{L}{W} R_S \tag{4.2}$$

where Z = L/W.

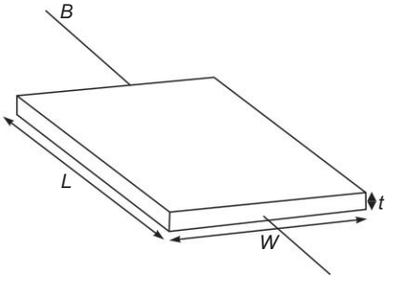


Fig. 4.1 Sheet-resistance model

4.1.2 Capacitance

A parallel plate consisting of dielectric material between two plates can be used as a capacitance which can be written as

$$C = \frac{kA}{D} \tag{4.3}$$

where D = Thickness of dielectric material, A = Area of cross section of the parallel plate, and k = Dielectric constant of dielectric material. The same parallel-plate capacitance concept can be used in MOSFET ($W = L$) where oxide layer between gate and substrate is used as capacitance.

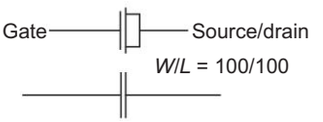


Fig. 4.2 MOSFET as a capacitance

4.2 □ ANALOG MOSFET MODELS

For use of MOSFET/CMOS in analog circuits, it is required to discuss low-frequency and high-frequency MOSFETs models which include transconductance, output resistance, and capacitances of the MOSFETs.

4.2.1 Low-Frequency Small-Signal MOSFET Model

In analog circuit, MOSFET devices are normally operated in saturation, and in the saturation region, the drain current I_{ds} of nMOSFET from Eq. (2.13) can be written as

$$I_{ds} = \frac{\beta}{2}(V_{gs} - V_{THN})^2 [1 + \lambda_c (V_{ds} - V_{ds,\,sat})] \tag{4.4}$$

where $V_{ds\ Dat} << V_{ds}$, λ_c = Channel-length modulation parameter, $\beta = \dfrac{\mu c_{ox} W}{L}$, μ = Electron mobility, and C_{ox} = Gate oxide capacitance.

Figure 4.3 shows a MOSFET circuit and its small-signal low-frequency model. The dc sources will be labeled with V_{GS} and the ac sources will be with subscripts (e.g., v_{gs}). We are assuming in the following that $V_{GS} \gg v_{gs}$. Since the MOSFET is in the saturation region, $V_{DS} > V_{GS} - V_{THN}$, the total (ac + dc) drain current is given by

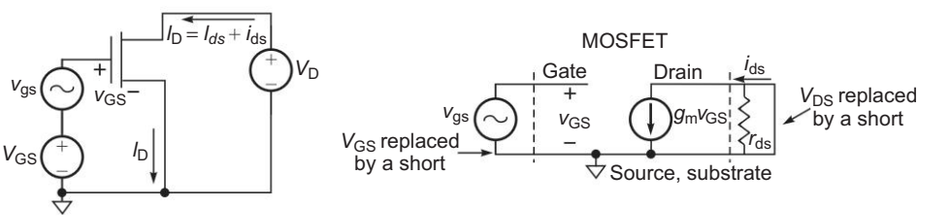


Fig. 4.3 Basic *n*MOSFET circuit and its small signal low frequency ac model

$$I_D = I_{ds} + i_{ds} = \frac{\beta}{2}(V_{GS} + v_{gs} - V_{THN})^2 (1 + \lambda_c.V_{ds}) \tag{4.5}$$

The forward transconductance, g_m, of the MOSFET is given by

$$g_m = \left.\frac{\partial I_D}{\partial v_{GS}}\right|_{V_{GS}=\text{Constant}} = \beta(V_{GS} + v_{gs} - V_{THN})\,(1 + \lambda_c.V_{ds}) \tag{4.6}$$

From the small-signal low-frequency ac equivalent circuit, it is seen that V_{ds} and V_{GS} are replaced by a short and for small signal, $v_{gs} << V_{GS}$. Since the modulation parameter $\lambda_c < 1$ and $\lambda_c . V_{ds} << 1$, hence we can write

$$g_m = \beta(V_{GS} - V_{THN}) = \sqrt{2\beta I_{ds}} \tag{4.7}$$

We can write output resistance as

$$r_{ds}^{-1} = \left.\frac{\partial I_D}{\partial V_{ds}}\right|_{V_{GS}=\text{Constant}} = \beta(V_{GS} - V_{THN})^2 \lambda_c = \lambda_c . I_{ds}$$

$$r_{ds} = \frac{1}{\lambda_c I_{ds}} \tag{4.8}$$

The maximum voltage gain can be written as

$$A_v = \frac{\partial V_{ds}}{\partial V_{gs}} = -\frac{1 \Big/ \dfrac{\partial V_D}{\partial V_{ds}}}{1 \Big/ \dfrac{\partial I_D}{\partial V_{gs}}} = -\frac{\sqrt{2\beta I_{ds}}}{\lambda_c I_{ds}} = -\frac{\sqrt{2\beta}}{\sqrt{I_{ds}.\lambda_c}} \tag{4.9}$$

4.2.2 High-Frequency Small-Signal MOSFET Model

For high-frequency small-signal MOSFET model, we have to consider the MOSFET capacitances a shown in Fig. 4.4. In the figure, the capacitances of drain diffusion region, source diffusion layers and gate over field region are denoted by C_{db}, C_{sb} and C_{gb} respectively. The capacitances of gate-drain and gate-source are represented by C_{gd} and C_{gs} respectively.

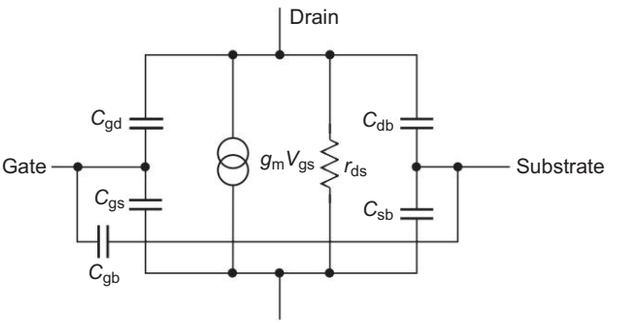


Fig. 4.4 High-frequency small-signal model

Figure 4.5 shows the simplified small-signal equivalent circuit where substrate and gate are shorted. In the figure, the input ac voltage v_s of frequency f is applied in gate G with reference to source S, and Z_s is the input impedance between gate and source. The current gain can be written as

$$\frac{i_{ds}}{i_g} = \frac{\beta(V_{gs} - V_{THN})}{j2\pi f(C_{gb} + C_{gs} + C_{gd})} \tag{4.10}$$

where V_{gs} = Gate voltage applied for making channel in *n*MOS.

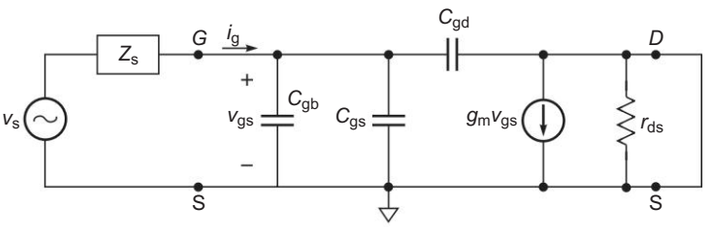


Fig. 4.5 Simplified high-frequency small-signal model

4.3 ▫ CURRENT SOURCE/SINK

The current source and sink are basic building blocks used extensively in CMOS IC design. In current source/sink, the output impedance should be infinite and capable of generating a constant current over a wide range of voltages. Because of finite values of output impedance, the performance of the current source will become limited. In this section, the complexity and performance of MOS current sources and sinks are discussed.

4.3.1 Current Mirror with *n*MOS as Current Source

The basic current mirror consisting of two misfits M_1 and M_2 is shown in Fig. 4.6. In both transistors, the source is connected to V_{SS}. The current passing through M_2 corresponding to V_{GS} is a multiple of the current in M_1. If the MOSFETs are of the same size, the same drain current flows in each MOSFET, provided M_2 is in the saturation region. So, assuming M_1 and M_2 to be of same size, the drain current

$$I_{D1} = I_{D2} = \frac{\beta}{2}(V_{GS} - V_{THN})^2$$

In the figure, the reference drain current is given by

$$I_{D1} = \frac{V_{DD} - V_{GS} - V_{SS}}{R} = \frac{\beta}{2}(V_{GS} - V_{THN})^2$$

The output resistance of M_2 is given by

$$r_0 = \frac{1}{\lambda_c I_{D1}} \tag{4.11}$$

where $I_{D1} = I_{D2}$. If the sizes of M_1 and M_2 are not equal then I_{D1} and I_{D2} are written as

$$I_{D1} = \frac{\beta_1}{2}(V_{GS} - V_{THN})^2$$

$$I_{D2} = \frac{\beta_2}{2}(V_{GS} - V_{THN})^2$$

The ratio of I_{D1} and I_{D2} is written as

$$\frac{I_{D2}}{I_{D1}} = \frac{\beta_2}{\beta_1} = \frac{W_2}{W_1} \tag{4.12}$$

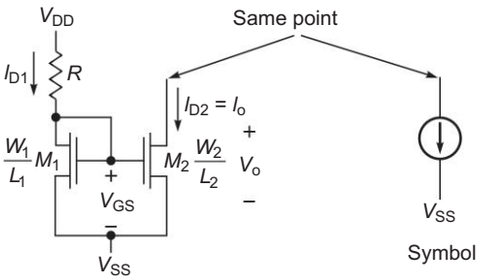


Fig. 4.6 Current mirror with *n*MOSFET and its symbol

4.3.2 Current Mirror using *p*MOSFET

Figure 4.7 shows current source using two *p*MOSFETs M_1 and M_2. Like a current mirror using *n*MOSFET, two *p*MOSFETs are biased with V_{DD} to form a current source. The output resistance depends on the biasing of transistors. Current mirror using *p*MOSFETs is acting as the current source. The output resistance is given by

$$r_0 = \frac{1}{\lambda_c I_0} \tag{4.13}$$

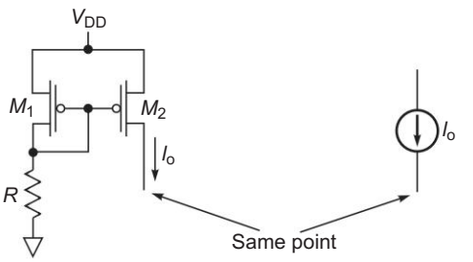


Fig. 4.7 Current mirror with *p*MOSFET and its symbol

4.3.3 Current Mirror using Cascaded Form

The cascade connection of a basic current mirror is used to increase the output resistance of a current source or sink. Figure 4.8 shows a triple cascaded current mirror. The output resistance for double cascade is given by

$$R_0 = r_{04}(1 + g_{m4}\, r_{02}) + r_{02} \approx g_{m4}\, r_0^2 \tag{4.14}$$

where $r_0 = r_{02} = r_{04}$. The output resistance for triple cascade is given by

$$R_0 = r_{06}(1 + g_{m6}(r_{04}(1 + g_{m4} r_{02}) + r_{02}))] + r_{04}(1 + g_{m4} r_{02}) + r_{02} \approx g_{m6} g_{m4} r_0^2$$

where $r_0 = r_{02} = r_{04} = r_{06}$.

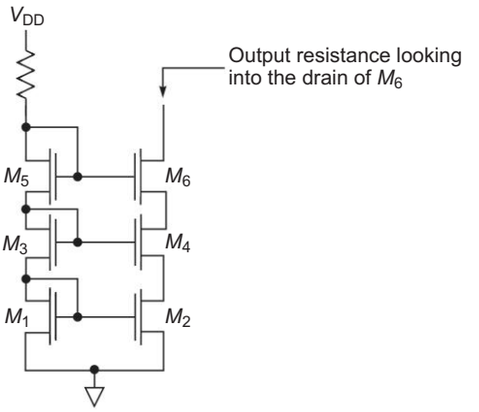


Fig. 4.8 Triple-cascaded current mirror

4.3.4 Regulated Cascaded Current Sink

Figure 4.9 shows the regulated cascade current mirror as a current sink. This current mirror also uses negative feedback to stabilize the output current and increase the output impedance. The transistors M_2 and M_4 make up the negative feedback loop, which stabilizes I_0. If I_0 were to increase then, since a constant current flows through M_3, the voltage on the node A would rise, which, in turn, would increase the current through M_2. Since the current through M_1 is also a constant amount, the voltage at the node B would decrease, thereby offsetting the increase in I_a by decreasing V_{GS4}.

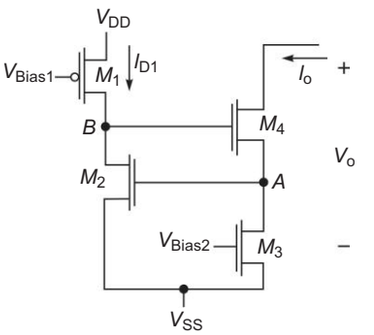


Fig. 4.9 Regulated cascaded current sink

This current mirror offers some significant advantages over any of the previously discussed current mirrors. First, the output impedance of this current mirror is significantly higher. The small-signal equivalent circuit is shown in Figure 4.10. Since M_1 and M_3 are applied with dc bias sources V_{bias}, only their resistances, r_{o1} and r_{o3}, are considered in the small-signal model and transconductance of these transistors are not considered.

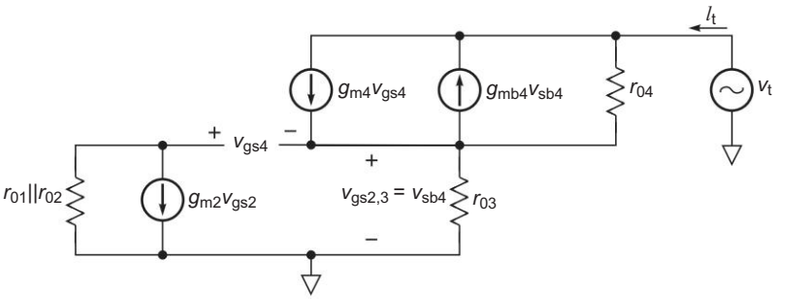


Fig. 4.10 Small-signal equivalent circuit of Fig. 4.10

Assuming all the transistor sizes are same, output resistance R_{out} can be written

$$R_{out} \approx g_{m2}\, g_{m4}\, (r_{01}||r_{02})\, r_{03}\, r_{04} \approx g_m^2 r_0^3/2 \tag{4.15}$$

where, $g_{m2} = g_{m4}$ and $r_{01} = r_{02} = r_{03} = r_{04}$

4.4 ☐ VOLTAGE DIVIDERS

In CMOS integrated circuit design, we can derive reference voltages from the power supplies using the resistor and the MOSFET. The voltage divider formed with two resistors has the advantage of simplicity, temperature insensitivity, and process insensitivity. The main problem with this circuit is that in order to reduce the power dissipation (i.e., the current through the resistors), the resistors must be made large. Large resistors require a large area which may not be practical in integrated circuits. The voltage divider can be made by the resistor and the MOSFET. Since the resistance has the problem of power dissipation, the voltage divider can be made by a CMOS having *n*- and *p*-MOSFET. The voltage divider with CMOS has the advantage that the layout can be small. Figure 4.11 shows voltage dividers using two resistances R_1 and R_2, using R_1 and *n* MOSFET and using CMOS.

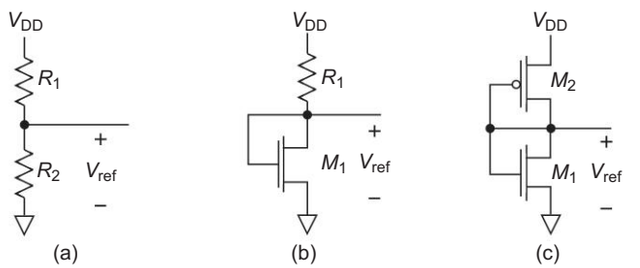


Fig. 4.11 Voltage divider: (a) Using resistances R_1 and R_2 (b) Using *n*MOSFET (c) Using CMOS

The reference voltage V_{ref} in Fig. 4.11(a) with resistances R_1 and R_2 is given by

$$V_{ref} = \frac{R_2 V_{DD}}{R_1 + R_2} \tag{4.16}$$

From Fig. 4.11(b), we can write

$$I_D = \frac{V_{DD} - V_{ref}}{R_1} = \frac{\beta_1}{2}\,(V_{ref} - V_{THN})^2$$

Considering $V_{DD} << V_{ref}$, we get V_{ref} for Fig. 4.11(b) as

$$V_{ref} = V_{THN} + \sqrt{\frac{2I_D}{\beta_1}} = V_{THN} + \sqrt{\frac{2V_{DD}}{R_1\beta_1}}$$

From Fig. 4.11(c), using CMOS, we can write

$$I_{D1} = I_{D2} = \frac{\beta_1}{2}(V_{ref} - V_{SS} - V_{THN})^2 = \frac{\beta_2}{2}(V_{DD} - V_{ref} - V_{THP})^2$$

From the above equation, we get V_{ref} for Fig. 4.11(c) as

$$V_{ref} = \frac{V_{DD} - V_{THP} + \sqrt{\frac{\beta_1}{\beta_2}}(V_{SS} + V_{THN})}{\sqrt{\frac{\beta_1}{\beta_2}} + 1} \tag{4.17}$$

Figure 4.12 shows a three-MOSFET voltage divider consisting of two *p*MOSFETs and one *n*MOSFET where two reference voltages are available. In this configuration, two *p*MOSFETs are made in their own well to avoid body effect. The difficult task in the design of this divider are the reference voltages needed to be determined.

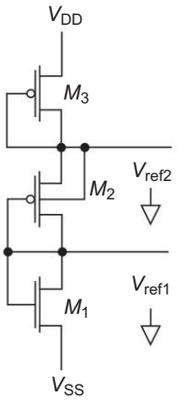


Fig. 4.12 Three-MOSFET voltage divider

4.5 □ MOS AMPLIFIERS

Single-stage amplifier is an important device. In this direction, aMOSFET-based amplifier with a MOSFET transistor as an active load, can save significant amounts of chip area. An active load can also produce higher values of resistance when compared with a passive resistor, resulting in higher gains. There are two types of MOSFET-based active loads–gate-drain connected load consisting of a MOSFET with the gate and drain shorted, and current source load amplifier consisting of a MOSFET with the gate and source shorted. The later configuration of the active load has a higher gain and higher output impedance, but a lower bandwidth than the former load. Current source load amplifiers are preferred when external feedback is applied to the amplifier to set the gain.

4.5.1 Single-Stage Common Source Amplifier

As discussed earlier, there are two types of load used in a common-source amplifier—gate-drain connected load and gate-source connected load.

1. Gate-Drain Connected Load

Figure 4.13 shows a common source voltage amplifier consisting of two *n*MOSFETs with gate-drain connected active load. In each configuration, M_1 and M_2 are assumed to be biased in the saturation region. The MOSFET amplifier can be analyzed by using small-signal analysis. Here, by replacing M_2 with a resistor of $\frac{1}{g_{m2}}$ and replacing M_1 with a current source of value $g_{m1}.V_{in}$ where $\frac{1}{g_{m2}} << r_{01}$, the low-frequency equivalent circuit is shown in the figure. The small-signal gain of this circuit is given by

$$\frac{v_{out}}{v_{in}} = \frac{-i_d(1/g_{m2})}{i_d(1/g_{m1})} = -\frac{g_{m1}}{g_{m2}} \tag{4.18}$$

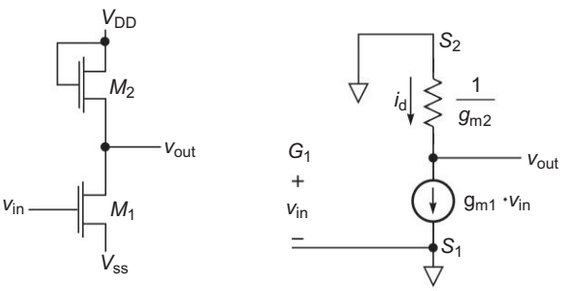


Fig. 4.13 Single-stage common-source amplifier with its equivalent circuit

Figure 4.14 shows a common-source voltage amplifier consisting of CMOS with *n*MOSFET as a gate-drain connected active load. In each configuration, M_1 and M_2 are assumed to be biased in the saturation region.

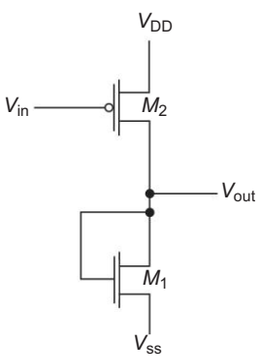


Fig. 4.14 Single-stage CMOS amplifier

The small-signal gain of this circuit is given by

$$\frac{v_{out}}{v_{in}} = -\frac{g_{m2}}{g_{m1}} \tag{4.19}$$

2. Gate-source Connected Load

Figure 4.15(a) shows a common source amplifier consisting of CMOS. In this figure, *p*MOSFET M_2 is used as a gate-source connected active load and an *n*MOSFET M_1 is used as a single-stage amplifier. Figure 4.15(b) shows a common-source amplifier consisting of two *p*MOSFETs. In this figure, the *p*MOSFET M_1 is used as a gate-source connected active load and the *p*MOSFET M_2 is used as a single-stage amplifier.

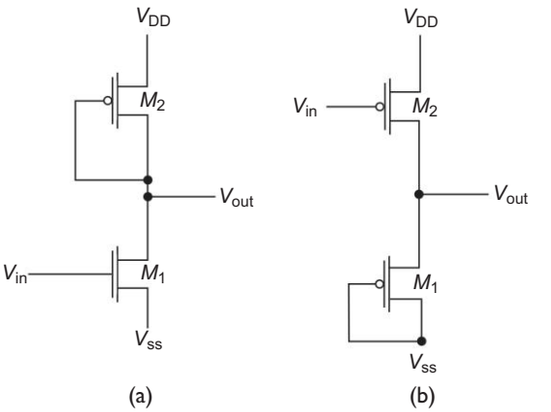


Fig. 4.15 Common source amplifier using gate-source active load: (a) Using CMOS (b) Using two *p*MOSFETs

4.5.2 Common-gate Amplifier

Figure 4.16 shows a common-gate Amplifier consisting of CMOS with *n*MOSFET M_1 as a common gate amplifier and *p*MOSFET M_2 as a gate-source connected active load. The voltage gain of this amplifier with neglected body effect is written as

$$\frac{v_{out}}{V_{in}} = \frac{-i_d(1/g_{m2})}{-i_d(1/g_{m1})} = \frac{g_{m1}}{g_{m2}} \tag{4.20}$$

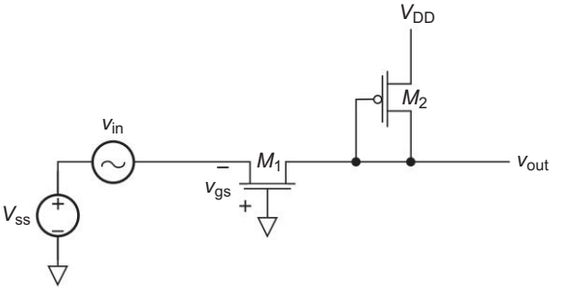


Fig. 4.16 Common-gate amplifier

4.5.3 Cascaded Amplifier

The amplifier discussed so far has a limitation of low amplification of gain due to its lower output resistance. Figure 4.17 shows a cascaded amplifier having two advantages over the basic common source and common-gate amplifier with current source load discussed previously; the gain is greater due to the large output resistance, and the absence of the Miller capacitance keeps the input time constant, preventing its increase. The small-signal model for the basic amplifier is shown in Fig. 4.17(b). The gain of M_1, (output voltage v_{d1}) can be written as

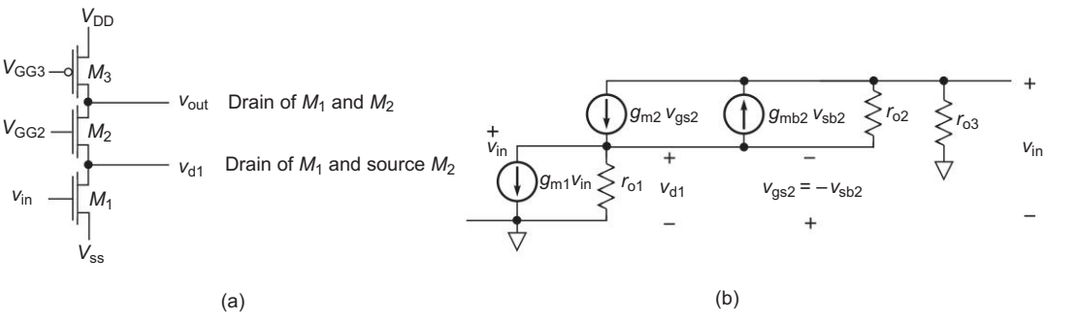


Fig. 4.17 Cascaded amplifier: (a) Circuit (b) Its equivalent circuit

$$A_{v1} = \frac{v_{d1}}{v_{in}} = \frac{-i_d (r_{in2} \| r_{01})}{i_d . (1/g_{m1})} \tag{4.21}$$

where r_{in2} is small-signal resistance seen looking into the source of M_2 and written as

$$r_{in2} = \frac{1}{g_{m2}} \left(1 + \frac{r_{03}}{r_{02}} \right) \tag{4.22}$$

So A_{v1} is written as

$$A_{v1} = \frac{v_{d1}}{v_{in}} = \frac{g_{m1}}{g_{m2}} \left(1 + \frac{r_{03}}{r_{02}} \right) \tag{4.23}$$

The gain of M_2 (where v_{d1} is input voltage and v_{out} is output voltage) is written as

$$A_{v2} = \frac{v_{out}}{v_{d1}} = \frac{-i_d r_{02} (1 + g_{m2} r_{01}) \| r_{03}}{-i_d \dfrac{1}{g_{m2}} \left(1 + \dfrac{r_{03}}{r_{02}} \right)} = \frac{g_{m2} [(r_{02} (1 + g_{m2} r_{01})) \| r_{03}]}{\left(1 + \dfrac{r_{03}}{r_{02}} \right)} \tag{4.24}$$

The overall gain is the product of A_{v1} and A_{v2} written as

$$A_v = A_{v1} . A_{v2} = -g_{m1} . [(r_{02} (1 + g_{m2} r_{01})) \| r_{03}] \approx -g_{m1} r_{03} \tag{4.25}$$

In this circuit, there is no significant improvement in voltage gain in comparison to common source amplifier because of limitation in output resistance r_{03} of M_3. Figure 4.18 shows dual cascaded amplifier

in which M_3 and M_4 make current source load. The output resistance is in parallel combination of the resistance looking into the drain of M_3 with resistance looking into the drain of M_2. So, we can write output resistance as

$$R_0 = [(r_{02}(1 + g_{m2}r_{01})] \| [r_{03}(1 + g_{m3}r_{04})] \tag{4.26}$$

The voltage gain is written as

$$A_v = \frac{v_{out}}{v_{in}} \approx -g_{m1}R_0 \tag{4.27}$$

The main drawback of cascaded amplifier connection is that since there are more transistors in the circuit, more supply voltages will be required to keep the devices in saturation.

4.5.4 Push-pull Amplifier

Figure 4.18 shows a push-pull amplifier consisting of inverter operating with both M_1 and M_2 in the saturation region. This configuration can sink and source equal amounts of current. In the figure, both M_1 and M_2 act as common source amplifiers. The resistance in the sources of M_1 and M_2 is the parallel combination of the resistance seen looking into the source of M_1 and M_2. The gain of this configuration is given by

$$A_v = \frac{v_{out}}{v_{in}} = \frac{-i_d(r_{01} \| r_{02})}{i_d.(1/g_{m1} \| 1/g_{m2})} = -(g_{m1} + g_{m2})(r_{01} \| r_{02}) \tag{4.28}$$

The push-pull amplifier is popular as an output amplifier because of equal source and sink capability and the ability to pull the output to V_{DD} or V_{SS}. This push-pull configuration is used as a buffer which gives current drive capability.

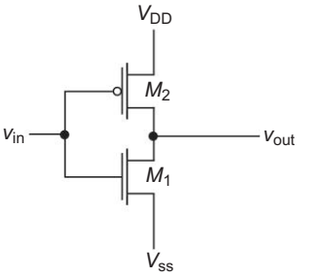


Fig. 4.18 Push-pull amplifier

4.5.5 Transimpedance Amplifier

Figure 4.19 shows a transimpedance amplifier using CMOS and *n*MOSFET. In this amplifier, voltage is out and current is in. Instead of voltage gain, we consider current gain. The input impedance of this amplifier is written as

$$R_{in} = \frac{1}{g_{m3}} \tag{4.29}$$

The input current is written as

$$i_{in} = g_{m3} v_{in} \tag{4.30}$$

The current through M_1 and M_2 is written as

$$i_d = g_{m1} v_{in} = \frac{g_{m1}}{g_{m3}} i_{in} \tag{4.31}$$

$$\frac{i_d}{i_{in}} = \frac{g_{m1}}{g_{m3}} \tag{4.32}$$

So the current gain is adjusted with the size of M_1 and M_3. The transresistance can be written as

$$R_T = \frac{v_{out}}{i_{in}} \approx -\frac{i_d (1/g_{m2})}{i_d \frac{g_{m3}}{g_{m1}}} = \frac{g_{m1}}{g_{m2} g_{m3}} \tag{4.33}$$

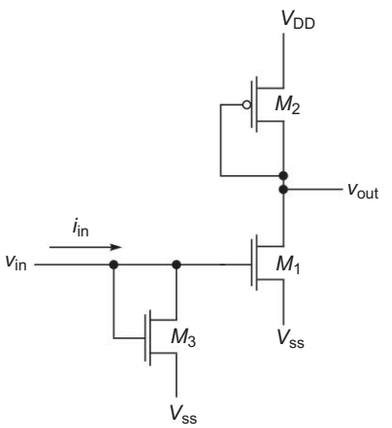


Fig. 4.19 Trans-impedance amplifier

Adding M_3 in a common-source amplifier, time constant reduces due to presence of $\frac{1}{g_{m3}}$ in R_T.

4.5.6 Class A Amplifier

Figure 4.20 shows a class A common source amplifier. In this case, M_1 can only sink current from the load capacitance and the current source/sources current to the load capacitance. The maximum rate at which the load capacitance can be charged is called the slew rate (SR) and is given by

$$SR = \frac{dV}{dt} = \frac{I}{C} \tag{4.34}$$

where I is current passing M_1.

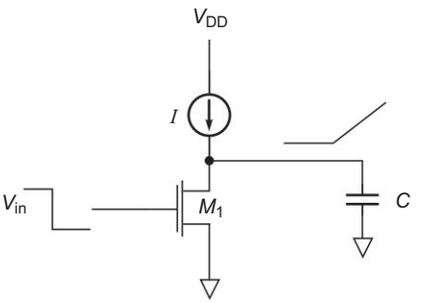


Fig. 4.20 Class A amplifier

Designing an amplifier that eliminates the current source limitations in charging or discharging the load capacitance is desirable for slew-rate limitation.

4.5.7 Class AB/Class B Amplifier

Figure 4.21 (a) shows the amplifier configuration in which the value of the batteries V_{GG2} and V_{GG1} determines the class (A, B, or AB) of the amplifier. When the input goes positive, V_{GS2} will increase while V_{SG1} will decrease. This causes the increase of current in M_2 and the decrease of the current in M_1. The magnitude of the input voltage when M_1 turns off is determined by V_{GG1}. This configuration does not exhibit output slew-rate limitations due to the absence of a current source in series with the charging or discharging path. The output voltage swing is limited to a threshold voltage below V_{DD} and above V_{SS}. The floating batteries V_{GG1} and V_{GG2} can be designed in the circuit shown in Fig. 4.21(b). The gate of M_6 is biased with a dc voltage so that its dc drain current is equal to the dc current through M_3. MOSFET M_3 is simply a current source. The gate-source voltages of M_4 and M_5 are constant (because of the constant current provided by M_3) and used as the biasing voltage needed to bias M_1 and M_2 on. This configuration is useful as an output buffer in an op-amp when the gate of M_6 is connected to an active load used in a differential pair and the gate of M_3 is connected to the same bias voltage as used in the diff-amp current source.

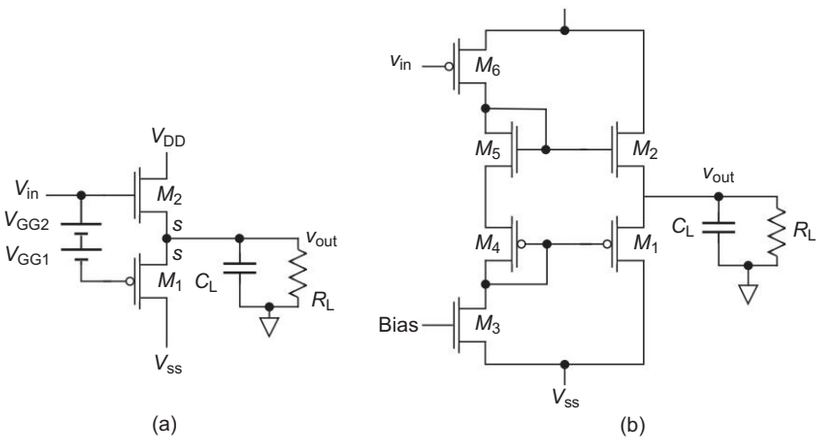


Fig. 4.21 Class AB amplifier: (a) Basic configuration (b) Complementary source-follower configuration

4. 5. 8 Feedback Amplifiers

A feedback amplifier is an amplifier in which a portion of the output signal is fed back and superimposed on its input signal to improve significant modification in the performance. The transfer of the energy from the output of the electrical network to its input is called feedback. Figure 4.22 shows feedback amplifier consisting of internal amplifier and a feedback network. The variables labeled as X value in the figure is either voltage or current. The internal amplifier of gain A has output X_0 and input X_i. The feedback extracts $X_f = \beta X_0$, (where β = Feedback ratio) and adds it to the input. So the input X_i is changed to $X_i = X_S \pm X_f$ from V_S. When $X_i = X_S + X_f$, the feedback is positive and when $X_i = X_S - X_f$, the feedback is negative. The voltage gain of internal amplifier is

$$A = \frac{X_0}{X_i} \tag{4.35}$$

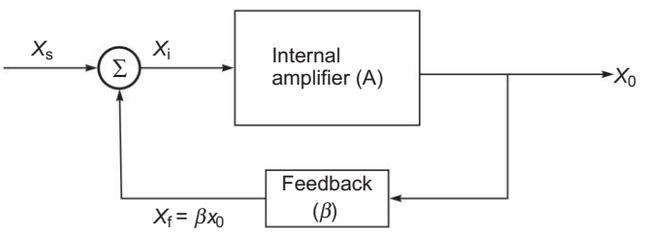


Fig. 4.22 Basic feedback amplifier

The overall voltage gain due to addition of feedback is given by

$$A_f = \frac{X_0}{X_S} = \frac{X_0}{X_i \pm X_f} = \frac{X_0 / X_i}{1 \pm X_f / X_i}$$

The ratio X_f/X_i is called **loop gain** or **feed factor** which is written as

$$\frac{X_f}{X_i} = \frac{X_f}{X_0} \cdot \frac{X_0}{X_i} = A\beta \tag{4.36}$$

The overall voltage gain is also called **closed-loop gain** which is rewritten as

$$A_f = \frac{X_0}{X_S} = \frac{A}{1 \pm A\beta} \tag{4.37}$$

where $(1 \pm A\beta)$ is called **return difference**. As per return difference, there are two types of feedback—positive feedback and negative feedback.

(i) For negative feedback, the return difference is $1 + A\beta$, i.e., return difference >1.

$$A_f = \frac{A}{1 + A\beta} \tag{4.38}$$

(ii) For positive feedback, the return difference is $1 - A\beta$, i.e., return difference <1.

$$A_f = \frac{A}{1 - A\beta} \tag{4.39}$$

(iii) If $1 + A\beta = 0$ then closed-loop gain A_f is infinity. It means that the feedback amplifier is then capable of sustaining an output even though its input is zero.

Positive feedback occurs when the system output is added to the system input, whereas negative feedback occurs when the system output is subtracted from its input. For gain stabilization and increase of bandwidth, negative feedback will be considered. For negative feedback, the degree of stabilization can be obtained through differentiation of closed loop gain A_f with respect to A as

$$\frac{dA_f}{A_f} = \frac{1}{1+A\beta}\frac{dA}{A} \tag{4.40}$$

The sensitivity of the transfer gain for the feedback amplifier is given by

$$S = \frac{dA_f}{A_f} \Big/ \frac{dA}{A} = \frac{1}{1+A\beta} \tag{4.41}$$

The desensitivity of the negative feedback is written as

$$D = \frac{1}{S} = 1 + A\beta \tag{4.42}$$

Due to negative feedback, the lower half-power frequency and upper half-power frequency can be written as

$$\text{Lower half-power frequency } f_{if} = \frac{f_l}{1+A_m\beta} \tag{4.43}$$

$$\text{Upper half-power frequency } = f_{hf} = f_h\,(1 + A_m\beta) \tag{4.44}$$

where, A_m = Midband gain, f_l = Lower half-power frequency without negative feedback and f_h = Upper half-power frequency without negative feedback. From the above expression, it is clear that $f_{lf} < f_l$ and $f_{hf} < f_h$. The extended bandwidth is written as

$$(f_{hf} - f_{if}) - (f_h = f_l) = f_h A_m \beta\, \frac{f_l A_m \beta}{1 + A_m \beta} \tag{4.45}$$

The positive feedback can be useful if controlled, its applications will be discussed in a later chapter. However, methods for minimizing positive feedback effects will be discussed here.

(a) Input Mixing and Output Sampling Network Figure 4.23 shows the feedback amplifier (Fig. 4.22) network consisting of input mixing and ouput mixing. In both input mixing and output sampling network, either Thevenin or Norton equivalent is used. In this section, we will discuss different types of input and output network. In all types, the feedback network consists of resistance $R_{\beta i}$ at input side and resistance $R_{\beta 0}$ at output side of the amplifier.

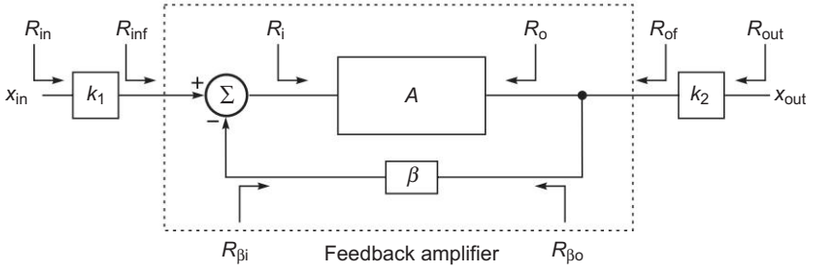


Fig. 4.23 Feedback amplifier network

(b) Input Mixing There are two types of mixing of feedback with input—series mixing and shunt mixing. Figure 4.24(a) shows series input mixing in which input voltage $v_i = v_S - v_f$ whereas for shunt mixing, the summation at the input can be written as currents $i_i = i_s - i_f$, as shown in Fig. 4.24(b).

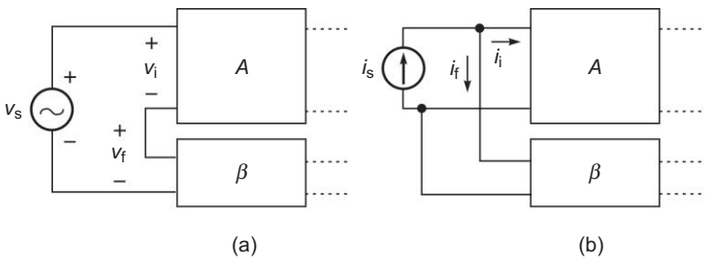


Fig. 4.24 Input mixing: (a) Series (b) Shunt

The input impedance of series input mixing is obtained as

$$R_{inf} = R_{in}(1 + A\beta) \tag{4.46}$$

The input impedance of shunt input mixing can be written as

$$R_{inf} = \frac{R_{in}}{1 + A\beta} \tag{4.47}$$

(c) Output Sampling There are two types of extraction of feedback from the output—output series sampling and shunt output sampling. Figure 4.25(a) shows series output sampling in which the feedback senses or samples the voltage across R_L whereas for shunt sampling, the feedback senses or samples the current at the output as shown in Fig. 4.25(b).

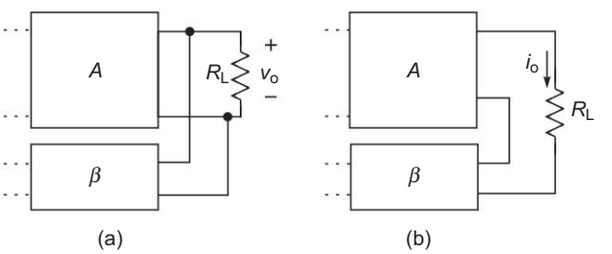


Fig. 4.25 Input sampling: (a) Shunt (b) Series

The output impedance of series output sampling is obtained as

$$R_{0f} = R_0(1 + A\beta) \tag{4.48}$$

The output impedance of shunt output sampling can be written as

$$R_{of} = \frac{R_o}{1 + A\beta} \tag{4.49}$$

1. Serier-Shunt Feedback Voltage Amplifier using CMOS

Figure 4.26 shows series-shunt feedback voltage amplifier using CMOS. In the figure, input variables, x_s, x_t, and x_p correspond to the voltages, v_s, v_i and v_f, such that the equation, $v_i = v_s - v_f$, can be written; the input mixing is in series. The current flowing through resistance R_G will make current mixing. The output sampling is of the shunt type since the feedback network and the basic amplifier are in parallel (the β network is connected to the same node as the output and v_o is the voltage being sampled). This amplifier employs series mixing and shunts sampling and is referred to as a series-shunt amplifier.

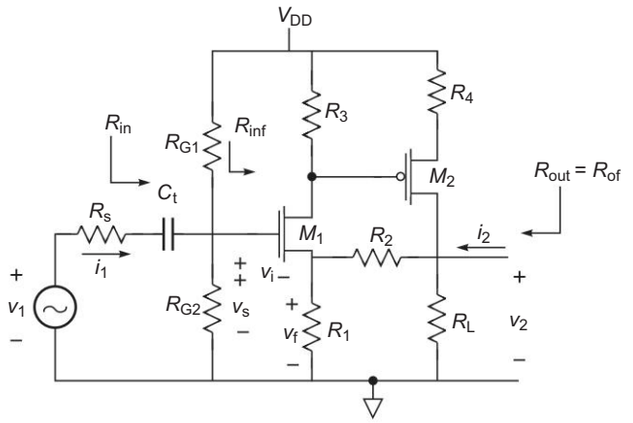


Fig. 4.26 CMOS-based series-shunt feedback amplifier

The closed-loop small-signal equivalent circuit is shown in Fig. 4.27. In the figure, the feedback path between nodes 3 and 4 with feedback variable v_f appears across R_1. The feedback factor β is written as

$$\beta = \frac{v_f}{v_2} = \frac{R_2}{R_1 + R_2} \tag{4.50}$$

The R_0 is written as

$$R_0 = \frac{v_0}{i_2} = R_L \| (R_1 + R_2) \tag{4.51}$$

The R_{in} is written as

$$R_{in} = \frac{v_1}{i_1} = R_S + (R_{G1} \| R_{G2}) \tag{4.52}$$

The input impedance R_{inf} and output impedance R_{0f} are written as

$$R_{inf} = \frac{v_s}{i_s} = R_{in}(1 + A\beta) \tag{4.53}$$

$$R_{of} = \frac{v_2}{i_2} = \frac{R_0}{1 + A\beta} \tag{4.54}$$

The closed-loop gain is written as

$$A_f = \frac{v_2}{v_s} = \frac{A}{1 + A\beta} \tag{4.55}$$

where A is open-loop gain.

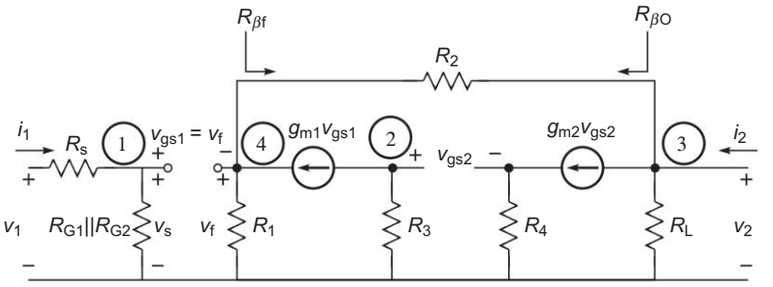


Fig. 4.27 Closed-loop small-signal equivalent circuit

2. CMOS-Based Shunt-Shunt Feedback

Figure 4.28 shows a CMOS-based shunt-shunt feedback amplifier in which the feedback mixes current at its input and samples voltage at the output. In the ideal case, $R_{\beta i}$ is infinite and $R_{\beta 0}$ is zero. In the figure, the basic amplifier and the feedback network are in parallel at both the input and the output. It shunts the basic amplifier; therefore, the input variable is a current. Since the feedback path is in parallel with or shunts the output signal, hence, the output signal is a voltage. In the figure, the feedback network consists of R_2 and R_5. The closed-loop equivalent circuit of the figure is shown in Fig. 4.29. In the equivalent circuit, r_{oi} and r_{02} are included as feedback path in basic amplifier. Since the gain is very small compared to the feedback path of β network, the internal feedback path of the amplifier is

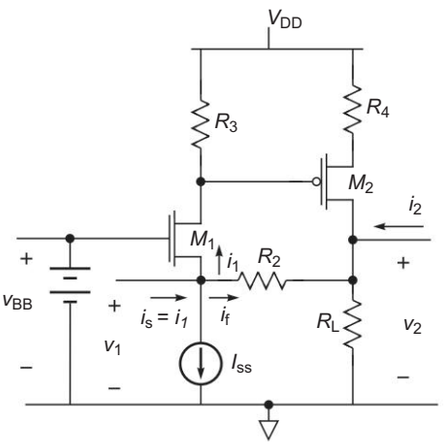


Fig. 4.28 CMOS shunt-shunt amplifier

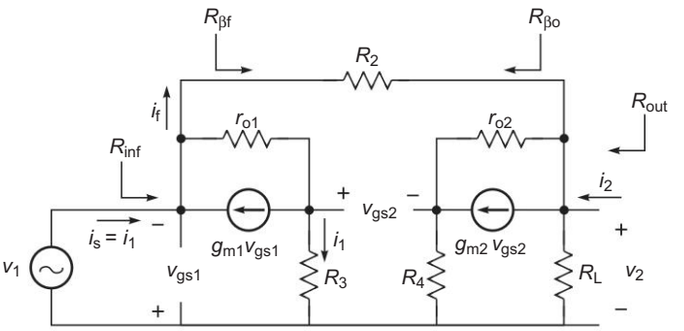


Fig. 4.29 Closed loop equivalent circuit of Fig. 4.28

negligible. Since the input variable is a current and the output variable is a voltage, the closed-loop parameter A_f of the basic amplifier has units of *V/I*, also known as a trans-impedance amplifier. Since-open-loop parameter $A\beta$ should always be unit less and positive, β will have units of *I/V* (mhos). The closed-loop gain is written as

$$A_f = \frac{v_0}{i_s} = \frac{A}{1 + A\beta} \tag{4.56}$$

The open-loop gain is written as

$$A = \frac{v_2}{i_s} = \frac{v_2}{v_{g2}} \cdot \frac{v_{g2}}{v_1} \cdot \frac{v_1}{i_s} \tag{4.57}$$

where, the input impedance can be written as

$$R_i = \frac{v_1}{i_s} = \frac{1 + \dfrac{R_3}{r_{01}}}{g_{m1} + \dfrac{1}{r_{01}}} \| R_2 \tag{4.58}$$

The voltage gain of MOSFET M_2 is written as

$$\frac{v_2}{v_{g2}} = \frac{-g_{m2}(R_L \| R_2 \| [(1 + g_{m2}R_4)r_{02} + R_4]}{1 + g_{m2}R_4 + \dfrac{R_4}{r_{02}}} \tag{4.59}$$

The voltage gain of M_1 is written as

$$\frac{v_{g2}}{v_1} = \frac{-g_{m1}R_3 + \dfrac{R_3}{r_{01}}}{1 + \dfrac{R_3}{r_{01}}} \tag{4.60}$$

3. CMOS-based Series-Series Feedback Amplifier

Figure 4.30 shows a CMOS-based series-series amplifier where input variables are in terms of voltages and the output variable is considered to be a current since the feedback network is in series with amplifier output. Ideally, the values of $R_{\beta i}$ and $R_{\beta 0}$ are zero. The feedback resistance R_F is in series with the input. The input of the amplifier is a voltage and the output is a current. So it is called transconductance amplifier where amplification A will be I/V and the units of feedback factor β will be V/I.

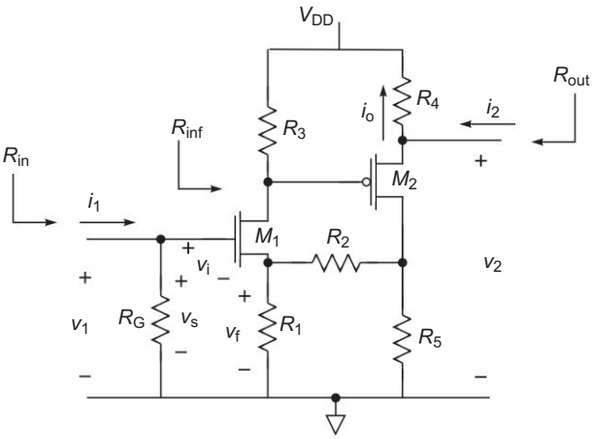


Fig. 4.30 CMOS based series-series feedback amplifier

The closed-loop gain of the transconductance amplifier is

$$A_f = \frac{i_0}{v_s} = \frac{A}{1 + A\beta}$$

The input resistance with feedback is written as

$$R_{inf} = R_i (1 + A\beta)$$

The output impedance can be written as

$$R_{of} = R_0 (1 + A\beta)$$

The R_i and R_0 are found from the equivalent circuit which is shown in Fig. 4.31. From the input mixing portion of equivalent circuit, $R_{\beta i}$ is written as $R_2 + R_5$ and $R_{\beta 0}$ is $R_1 + R_2$. In the figure, R_S is the source resistance. The output impedance and input impedance without feedback is written as

$$R_0 \approx R_4 + \frac{1}{g_{m2}} \tag{4.61}$$

$$R_i \approx R_G$$

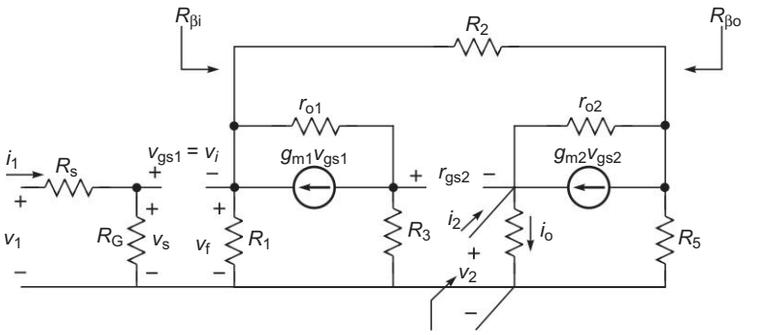


Fig. 4.31 Closed-loop small-signal equivalent of Fig. 4.31

4. CMOS-Based Shunt-series Feedback Amplifier

Figure 4.32 (a) shows CMOS-based shunt-series amplifier where both input and output variables are in terms of current. Here, both amplification A and feedback factor β will be *I/I*. The feedback network is in series with amplifier output. Ideally, the input impedance is to be very low and output impedance should be very high. In the figure, the dc current source is viewed as an ac open circuit and dc source. V_{GG} and V_{DD} are considered as an ac short circuit. Figure 4.32(b) shows shunt mixing in which $i_1 = i_s - i_f$. The small-signal output variable is the current i_0. The input of the amplifier is a current and the output is also a current. The closed-loop gain of the transconductance amplifier is

$$A_f = \frac{i_0}{i_s} = \frac{A}{1 + A\beta} \tag{4.62}$$

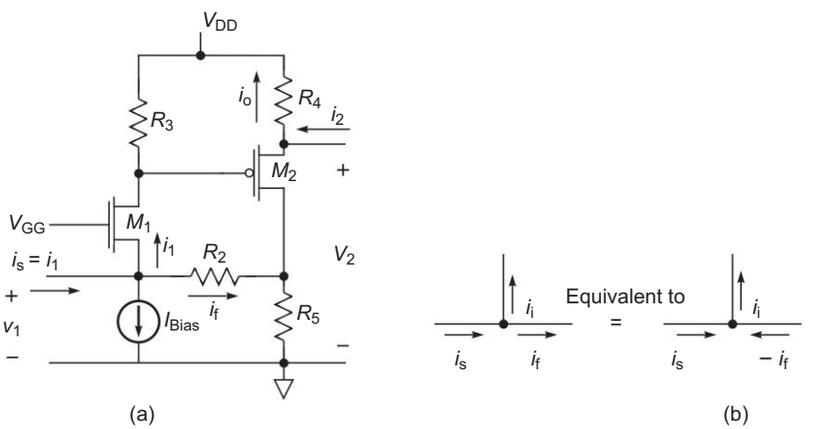


Fig. 4.32 CMOS-based shunt-series feedback amplifier

The input resistance with feedback is written as

$$R_{inf} = R_i/(1 + A\beta) \tag{4.63}$$

The output impedance can be written as

$$R_{of} = R_0(1 + A\beta) \tag{4.64}$$

where $R_0 \approx R_4$ and $R_i \approx (1/g_{m1})||r_{01}||r_{02}$ (4.65)

4.5.9 Differential Amplifier

A differential amplifier is an amplifier that amplifies the difference between two signals. The diff-amp is a fundamental building block in a CMOS operational amplifier/analog integrated circuit design. We discuss three basic types of differential amplifiers: the source-coupled pair, the source cross-coupled pair, and the current-differential amplifier.

1. Source-Coupled Pair Differential Amplifier

The source-coupled pair differential amplifier consisting of M_1 and M_2 is shown in Fig. 4.33. The MOSFET current mirror made using M_5 and M_6 is used to provide a source current for the pair of I_{ss}. We assume that M_1 and M_2 of the source coupled pair are of equal size, so that $\beta_1 = \beta_2 = \beta$. Summing the ac and dc components of the current at the sources of M_1 and M_2 yields

$$I_{SS} = I_{D1} + I_{D2} \tag{4.66}$$

If we label the input voltages at the gates of M_1 and M_2 as v_{i1} and v_{i2}, we can write the difference as

$$v_{D1} = v_{i1} - v_{i2} = v_{gs1} - v_{gs2} \tag{4.67}$$

When the gates of M_1 and M_2 are grounded then

$$I_{D1} = I_{D2} = I_{SS}/2 \tag{4.68}$$

At saturation, the drain current of M_1 and M_2 are written as

$$I_{D1} = \frac{\beta}{2}(V_{gs1} - V_{THN})^2 \tag{4.69}$$

$$I_{D2} = \frac{\beta}{2}(V_{gs2} - V_{THN})^2 \tag{4.70}$$

So the difference of input voltages is found to be

$$v_{D1} = v_{gs1} - v_{gs2} = \sqrt{\frac{2}{\beta}}(\sqrt{I_{D1}} - \sqrt{I_{D2}}) \tag{4.71}$$

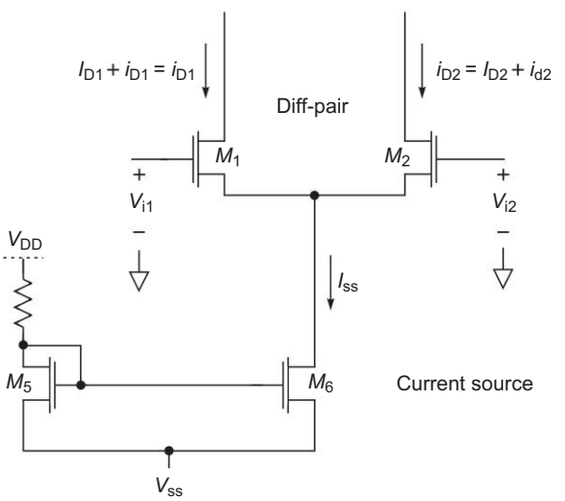


Fig. 4.33 MOSFET based source-coupled differential amplifier

So I_{D1} can be written from the equations (4.67) to (4.71),

$$I_{D1} = \frac{I_{SS}}{2}\left[1+\sqrt{\frac{\beta V_{D1}{}^2}{I_{SS}}-\frac{\beta^2 V_{D1}{}^4}{4I_{SS}^2}}\right] \tag{4.72}$$

$$I_{D2} = \frac{I_{SS}}{2}\left[1-\sqrt{\frac{\beta V_{D1}{}^2}{I_{SS}}-\frac{\beta^2 V_{D1}{}^4}{4I_{SS}^2}}\right] \tag{4.73}$$

The difference in current is

$$\Delta I_D = I_{D1} - I_{D2} = \frac{2I_{SS}}{2}\sqrt{\frac{\beta V_{D1}{}^2}{I_{SS}}-\frac{\beta^2 V_{D1}{}^4}{4I_{SS}^2}}$$

$$= \frac{I_{SS}\beta V_{D1}}{2I_{SS}}\sqrt{\frac{2I_{SS}}{\beta}-V_{D1}{}^2} \tag{4.74}$$

For small-signal conductance,

$$g_m = \frac{d(\Delta I_D)}{dV_{D1}} = \frac{\beta}{2}\sqrt{\frac{2I_{SS}}{\beta}} = \sqrt{\frac{\beta I_{SS}}{2}} \tag{4.75}$$

Current Source Load The CMOS differential amplifier is most often used with a current source load as shown in Fig. 4.34. When the gates of M_1 and M_2 are grounded, under these circumstances, a current $I_{ss}/2$ flows in M_1-M_4. The drain voltage of M_4 is at the same potential as the gate of M_3 and M_4. Therefore, the potential difference between the drain of M_1 and M_2 is zero. The drain of M_4 is at the same potential as the gate of M_3 used to bias the next stage at a particular current level.

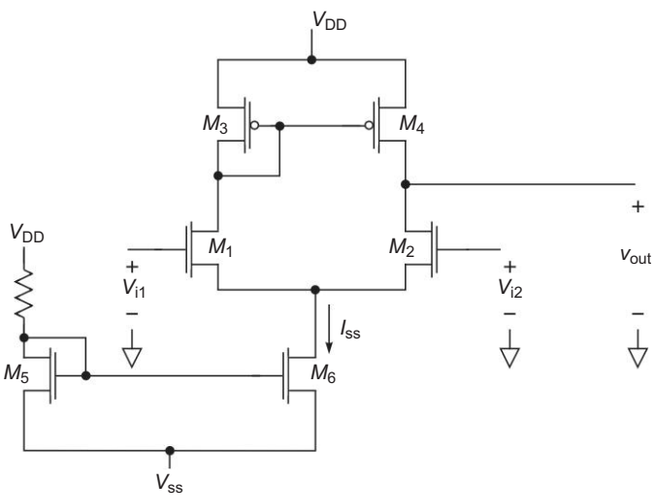


Fig. 4.34 Differential amplifier with current source load M_3 and M_4

2. Source Cross-Coupled Pair

Figure 4.35 shows a source cross-coupled pair which is very useful in practical circuit design. We assume that both inputs are connected to ground, all *n*-channels and all *p*-channels are the same size and a current I_{ss} flows in all the MOSFETs in the circuit.

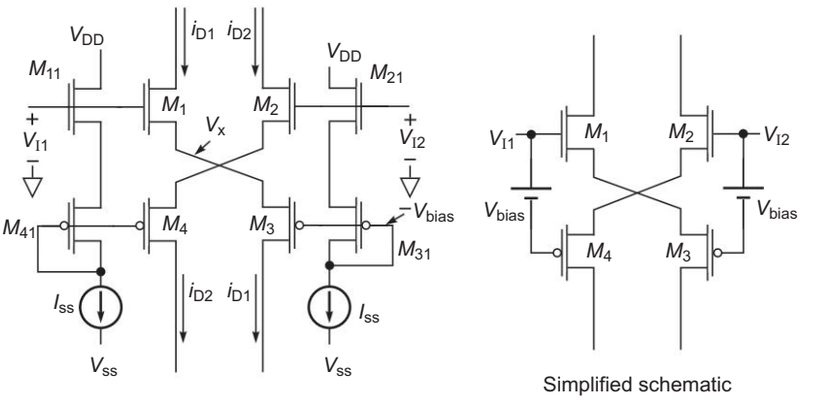


Fig. 4.35 Cross-coupled differential amplifier with nMOSFET input devices

With the gate of M_2 grounded, the gate of M_3 is at a constant potential which we will label V_{bias} (or approximately –2.4 V with the above sizes). We will label the potential at the sources of M_1 and M_3 as V_x. Assuming that M_1 and M_4 remain in the saturation region, we can write the input voltages as

$$v_{I1} = \sqrt{\frac{2i_{D1}}{\beta_1}} + V_x + V_{THN} \tag{4.76}$$

$$i_{D1} = \frac{\beta_3}{2}(V_x + V_{bias} - V_{THP})^2 \tag{4.77}$$

From the equations (4.74)–(4.77), we can write

$$V_x = \sqrt{\frac{2i_{D1}}{\beta_3}} - V_{bias} + V_{THP} \tag{4.78}$$

So the input voltage can be written as

$$v_{I1} = \sqrt{\frac{2i_{D1}}{\beta_1}} + \sqrt{\frac{2i_{D2}}{\beta_3}} - V_{bias} + V_{THN} + V_{THP} \tag{4.79}$$

So i_{D1} can be written as

$$i_{D1} = \frac{1}{2}(v_{I1} + V_{bias} - V_{THP} - V_{THN})^2 \frac{\beta_1\beta_3}{\left(\sqrt{\beta_1} + \sqrt{\beta_3}\right)^2} \tag{4.80}$$

Similarly, i_{D2} can be written as, assuming M_2 and M_3 are in saturation region with input voltage at M_1 grounded,

$$i_{D2} = \frac{1}{2}(v_{I2} + V_{bias} - V_{THP} - V_{THN})^2 \frac{\beta_2\beta_4}{\left(\sqrt{\beta_1} + \sqrt{\beta_4}\right)^2} \tag{4.81}$$

The difference in voltage is $v_{DI} = v_{I1} - v_{I2}$ and difference in current is written as $\Delta I_D = i_{D1} - i_{D2}$. Figure 4.36 shows a cross-coupled differential amplifier with current source active load.

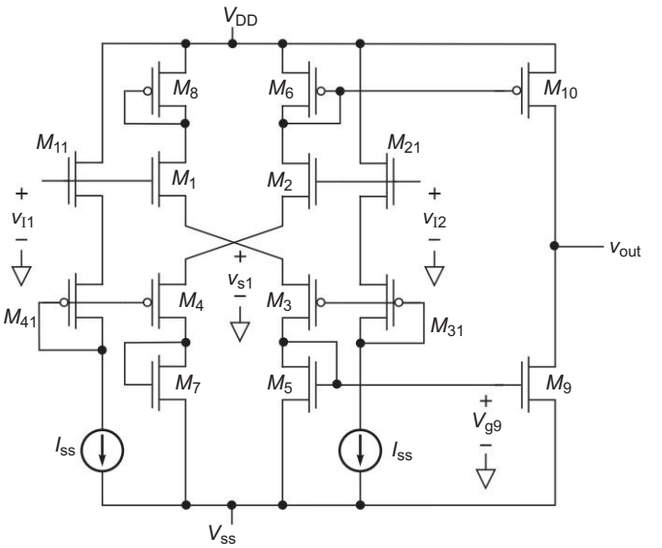


Fig. 4.36 Cross coupled Differential amplifier with common source load

3. Wide-Swing Current Differential Amplifiers

An increasingly important concern with the advent of low-voltage design is the need for improved common-mode range. Current differential amplifier is the amplifier where input swing of currents are allowed. Figure 4.37 shows a current differential amplifier where M_1 and M_4 are of same size. If

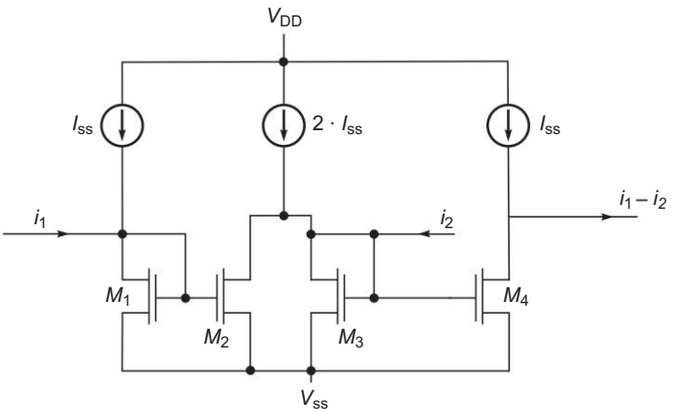


Fig. 4.37 Current differential amplifier

both i_1 and i_2 are zero, then current I_{SS} flows in all MOSFETS. When i_1 increases above zero but is less than I_{SS}, it leads to increase of current in M_1 and M_2 and decrease of drain current in M_3 to keep $i_{D1} + i_{D2} = 2I_{SS.}$ The input impedance of current differential amplifier is small signal resistance, i.e., $R_{in} = \dfrac{1}{g_m}$

4.6 ☐ OPERATIONAL AMPLIFIER

The operational amplifier (op-amp) is a fundamental building block in analog integrated circuit design. A block diagram of the two-stage op-amp with output buffer is shown in Fig. 4.38. The first stage of an op-amp is a differential amplifier. This is followed by another gain stage, such as a common source stage, and finally an output buffer. If the op-amp is intended to drive a small purely capacitive load, which is the case in many switched-capacitor or data-conversion applications, the output buffer is not used. If the op-amp is used to drive a resistive load or a large capacitive load (or a combination of both), the output buffer is used. Design of the op-amp consists of determining the specifications, selecting device sizes and biasing conditions, compensating the op-amp for stability, simulating and characterizing the op-amp A_{0L} (open-loop gain), CMR (Common-Mode Range on the input), CMRR (Common-Mode Rejection Ratio), PSRR (Power Supply Rejection Ratio), output voltage range, current sourcing/sinking capability, and power dissipation.

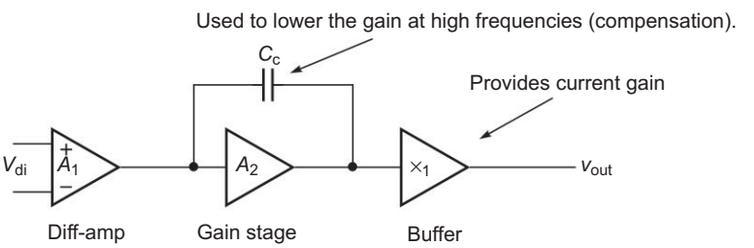


Fig. 4.38 Block diagram of two-stage op-amp with output buffer

4.6.1 Basic CMOS Op-Amp Design

The basic op-amp design is shown in Fig. 4.39 in which we will use this configuration to illustrate the design of a CMOS op-amp.

1. Differential Amplifier Stage with Bias Current, I_{ss}

Selection of the current I_{ss} is determined by gain, CMR, CMRR, power dissipation, noise, matching considerations, and slew rate. The small-signal gain of the differential amplifier is given by

$$A_1 = g_{m1}(r_{02} \| r_{04}) = \frac{2\sqrt{\beta}}{(\lambda_2 + \lambda_4)\sqrt{I_{SS}}} \tag{4.82}$$

The selection of the bias current is made by achieving the above characteristics. In general (increasing W while holding L constant), we make the MOSFETs in the differential pair at a given bias current and the lower V_{GS}, such that CMR is increased, input-referred noise is lowered, matching is better, and gain is increased. The main drawbacks are increased layout area and parasitic capacitances and thus, lower speed.

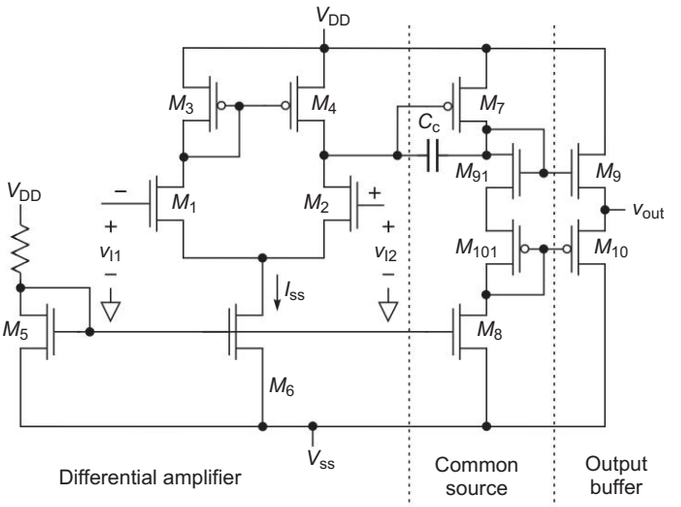


Fig. 4.39 Operational amplifier with output buffer

2. Second Stage

In the design of the second stage consisting of M_7, M_8, M_{91}, and M_{101} transistors, the selection of the biasing current is crucial. The same considerations that applied to selection of the differential amplifier biasing current apply to the selection of the second-stage biasing current. With both inputs to the op-amp, (i.e., the gates of M_1 and M_2), at the same potential, the same current flows in M_3 and M_4. The result is that the drain of M_4 is at the same potential as its gate. The gain of the second stage is given by

$$A_1 = -g_{m7}(r_{07} \| r_{08}) = \frac{-\sqrt{2\beta I_{D7}}}{(\lambda_7 + \lambda_8) I_{D7}} \tag{4.83}$$

where, $1/g_{m91} + 1/g_{m101}$ is small compared to $(r_{07} \| r_{08})$. The open-loop gain of first stage and second stage is given by

$$A_L = A_1 . A_2 = -g_{m1}(r_{02} \| r_{04}).[g_{m7}(r_{07} \| r_{08})] \tag{4.84}$$

3. Buffer Stage

The final stage in the op-amp design is to select the compensation network which is called **output buffer.** The closed-loop gain of an op-amp can be written in terms of the feedback equation by

$$A_{\mathrm{CL}} = \frac{A_L}{1 + A_L \beta} \tag{4.85}$$

In Fig. 4.39, the compensating network consists of capacitance C_c which is included in small-signal model of the op-amp (Fig. 4.39). Figure 4.40 shows a small signal model of the op-amp. In the model, R_1, R_2, C_1 and C_2 can be written as

$$\begin{aligned} R_1 &= (r_{02} \,\|\, r_{04}) \\ R_2 &= (r_{07} \,\|\, r_{08}) \\ C_1 &= C_{\mathrm{gs}\,7} + C_{\mathrm{gd}\,7}\,(1 + |A_2|) + C_{\mathrm{db}\,4} + C_{\mathrm{gd}\,4} + C_{\mathrm{db}\,2} + C_{\mathrm{gd}\,2} \\ C_2 &= C_{\mathrm{gd}\,7}(1 + 1/|A_2|) + C_{\mathrm{db}7} + C_{\mathrm{db}8} + C_{\mathrm{gd}\,8} + C_{\mathrm{gd}\,9} + C_{\mathrm{gd}\,10} \end{aligned} \tag{4.86}$$

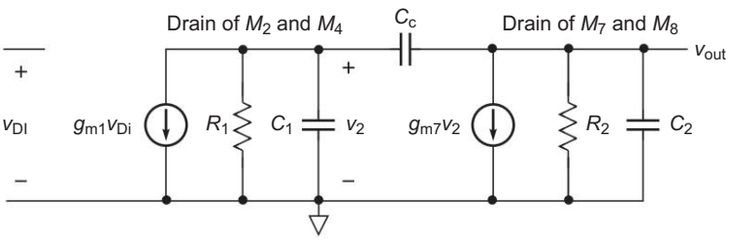


Fig. 4.40 Small-signal model of two-stage op-amp

The compensating capacitance C_c can be written as $C_c = \frac{g_{m1}}{2\pi f}$, where f = operating frequency and the slew rate (which is maximum rate of output voltage change with time) is written as

$$SR = \frac{dV_{\mathrm{out}}}{dt} = \frac{I_{\mathrm{SS}}}{C_c} \tag{4.87}$$

4.6.2 Characterizing the Op-Amp

For an op-amp design, it is required to characterize the op-amp to determine its characteristics.

1. Input Offset Voltage

One of the important parameters for an op-amp parameter is the input offset voltage. In an ideal case, if we ground both inputs to the op-amp, the output voltage is zero. In reality, the output voltage is nonzero because of mismatches in the input stage, and the potential on the output of the second stage is not well defined, leading to a **systematic offset.** The voltage in output due to systematic offset is called input offset voltage which is written as

$$V_{\mathrm{offset}} = \frac{V_{\mathrm{DD}}}{A_{\mathrm{L}}} \text{ or } \frac{V_{\mathrm{SS}}}{A_{\mathrm{L}}} \tag{4.88}$$

The offset voltage of an op-amp is modeled as a dc voltage source in series with the noninverting input of an offset free op-amp. This is shown in Fig. 4.41.

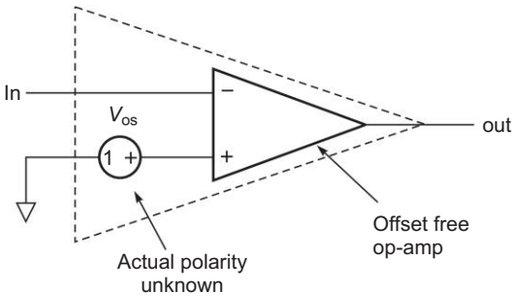


Fig. 4.41 Compensation of input offset voltage

2. Common Mode Rejection Ratio

The CMRR of an op-amp is calculated by multiplying the common-mode gain, A_c, of the input differential amplifier with the gain of the second stage and dividing the result into the open-loop gain, A_{0L}. The CMRR of an op-amp in dB is written as

$$\text{CMRR} = 20\log\left(\frac{A_L}{A_c A_2}\right) = 20\log\left(\frac{A_1}{A_c}\right) \tag{4.89}$$

which shows that the op-amp CMRR is determined by the differential stage. The offset voltage is generated in output if the common-mode signal is changed to ΔV_c. So the offset voltage is represented in terms of CMRR as

$$V_{\text{offset}} = \frac{\Delta V_c}{\text{CMRR}} \tag{4.90}$$

where ΔV_c = Change in common-mode signal.

3. Power Supply Rejection Ratio

The Power Supply Rejection Ratio (PSRR) is a parameter used to describe how well an amplifier rejects noise or changes on the V_{DD} and V_{SS} in power buses. This parameter can be extremely important in precision analog design. The positive PSRR is defined by

$$\text{PSRR}^+ = \frac{A_L}{v_{\text{out}}/v^+} \tag{4.91}$$

where v_{out} = Output voltage if both input voltages are zero and v^+ = Positive biasing voltage supply. The negative PSRR is defined by

$$\text{PSRR}^- = \frac{A_L}{v_{\text{out}}/v^-} \tag{4.92}$$

where v_{out} = Output voltage if both input voltages are zero and v^- = Negative biasing voltage supply.

4.6.3 Two-stage Op-amp Without Buffer

Fig. 4.42 shows a two-stage CMOS op-amp without buffer. This configuration is used for the comparator where the load is fully capacitive. The load capacitance C_L is used in estimating the op-amp compensation. Since the load capacitance affects the location of the second pole, it is possible that the compensated amplifier/load capacitance combination is unstable.

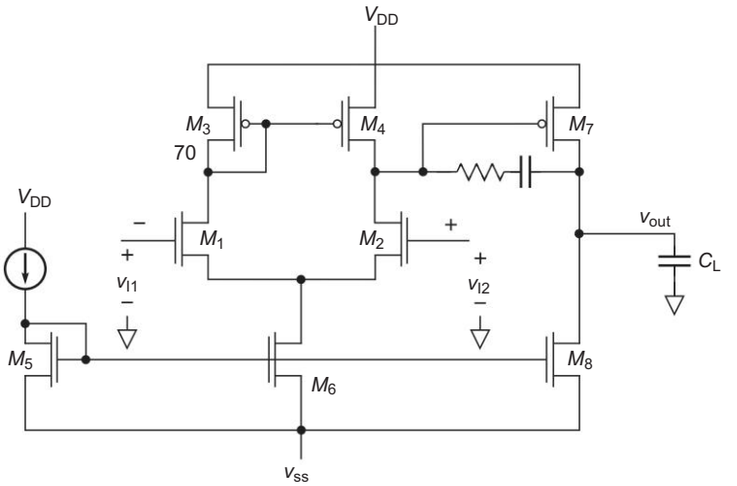


Fig. 4.42 Two-stage op-amp without buffer

4.6.4 Operational Transconductance Amplifiers

The Operational Transconductance Amplifier (OTA) is basically an op-amp without an output buffer which has only capacitive loads. An OTA can be defined as an amplifier where all nodes are low impedance except the input and output nodes. Figure 4.43 shows an OTA which is different from basic op-amp without buffer. The drain of M_4 is a high-impedance node and not the input or output of the amplifier. We assume that $\beta_1 = \beta_2$, $\beta_{31} = \beta_{41}$. The drain currents of i_{D31} and i_{D41} for M_{31} and M_{41} are written as

$$i_{D31} = i_{D41} = \frac{g_{m1}}{2}(v_{i1} - v_{i2}) = i_D$$

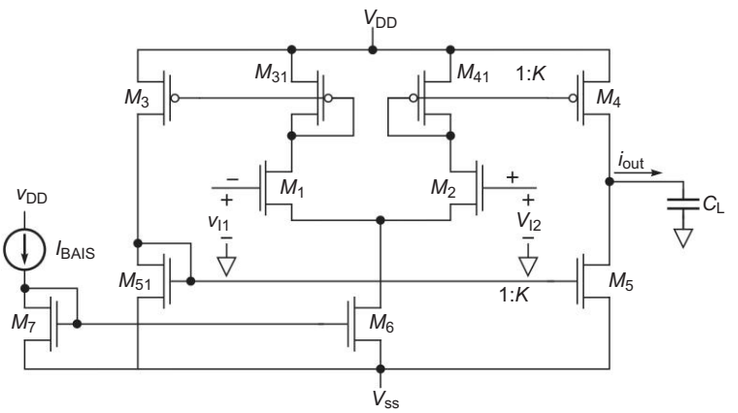


Fig. 4.43 Basic configuration of OTA

If the impedance of the capacitor is large compared to $(r_{04} \| r_{05})$ then the output voltage of the OTA is given by

$$V_{\text{out}} = 2\,K\,i_{\text{D}}\,(r_{04} \| r_{05}) \tag{4.93}$$

where $\beta_4 = K\,\beta_{41} = K\,\beta_{31}$.

The voltage gain is written as

$$A_{\text{v}} = \frac{v_{\text{out}}}{v_{\text{i2}} - v_{\text{i1}}} = Kg_{\text{m}}(r_{04} \| r_{05}) \tag{4.94}$$

where we have assigned the non-inverting input of the OTA as the gate of M_2. However, as the name states, we are interested in the transconductance (current out, voltage in) of the amplifier. If the impedance of the load capacitance is small compared to the output resistance then the output current flows mainly in the external load and is equal to

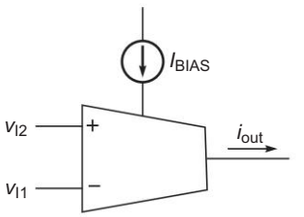


Fig. 4.44 Schematic symbol of an OTA

$$i_{\text{out}} = i_{\text{D4}} - i_{\text{D5}} = 2Ki_{\text{D}} \tag{4.95}$$

The symbol for the OTA is shown in Fig. 4.44.

1. OTA High-pass Filter

Figure 4.45 shows a high-pass OTA circuit having capacitance connected in input. The current input is written as

$$i_{\text{in}} = i_{\text{out}} = j\omega C\,(v_{\text{in}} - v_{\text{out}}) \tag{4.96}$$

The output voltage is given by

$$v_{\text{out}} = i_{\text{in}}/g_{\text{m}} = \frac{j\omega C}{g_{\text{m}}}\,(v_{\text{in}} - v_{\text{out}}) \tag{4.97}$$

The voltage gain can be written as

$$\frac{v_{\text{out}}}{v_{\text{in}}} = \frac{j\omega C/g_{\text{m}}}{1 + j\omega C/g_{\text{m}}} \tag{4.98}$$

The cut-off frequency of a high-pass filter is written as

$$f_{\text{cut}} = \frac{1}{2\pi C/g_{\text{m}}} \tag{4.99}$$

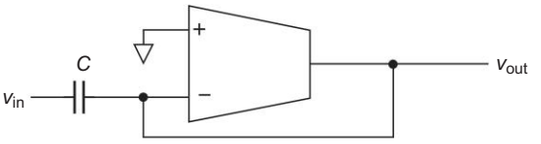


Fig. 4.45 High-pass filter using OTA

2. OTA Low-pass Filter

Figure 4.46 shows a high-pass OTA circuit having capacitance connected in output. The current output is written as

$$i_{out} = g_m (v_{in} - v_{out}) \tag{4.100}$$

The output voltage is given by

$$v_{out} = i_{in}. \frac{1}{j\omega C} = \frac{g_m}{j\omega C} (v_{in} - v_{out}) \tag{4.101}$$

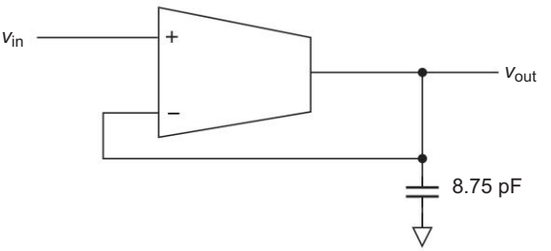


Fig. 4.46 Low-pass OTA circuit

The voltage gain can be written as

$$\frac{v_{out}}{v_{in}} = \frac{1}{1 + j\omega C/g_m} \tag{4.102}$$

The cut-off frequency of a low-pass filter is written as

$$f_{cut} = \frac{1}{2\pi C/g_m} \tag{4.103}$$

3. OTA Biquadratic Filter

Figure 4.47 shows a biquad OTA filter which can be useful as a low-pass, high-pass, band-pass, or band-reject filter with input conditions. We assume that the transconductances of each stage are equal. The ideal transconductance amplifier has infinite input and output resistances. The OTA circuit in the figure has a modest output resistance. The output resistance of the OTA causes an error in the transfer

function in filter applications. The MOSFET M_9 is used to keep the drain of M_3 at the same potential as the drain of M_4 and thus eliminating errors due to channel-length modulation. The bias voltages can be selected so that M_4 and M_5 are kept in the saturation region.

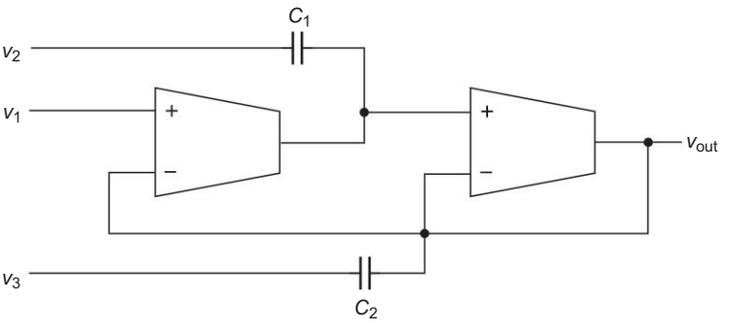


Fig. 4.47 Biquad filter using OTA

For low-pass filter configuration of the figure, input conditions are $v_{in} = v_1$ with v_2 and v_3 grounded and the transfer function is written as

$$\frac{v_{out}}{v_{in}} = \frac{g_m^2}{s^2C_1C_2 + sC_1g_m + g_m^2} \tag{4.104}$$

where $s = j\omega$.

For high-pass filter configuration of the figure, input conditions are $v_{in} = v_3$ with v_1 and v_2 grounded and the transfer function is written as

$$\frac{v_{out}}{v_{in}} = \frac{s^2C_1C_2}{s^2C_1C_2 + sC_1g_m + g_m^2} \tag{4.105}$$

For band-pass filter configuration of the figure, input conditions are $v_{in} = v_2$ with v_1 and v_3 grounded and the transfer function is written as

$$\frac{v_{out}}{v_{in}} = \frac{sC_1g_m}{s^2C_1C_2 + sC_1g_m + g_m^2} \tag{4.106}$$

For band-reject filter configuration of the figure, input conditions are $v_{in} = v_1 = v_3$ with v_2 grounded and the transfer function is written as

$$\frac{v_{out}}{v_{in}} = \frac{s^2C_1C_2 + g_m^2}{s^2C_1C_2 + sC_1g_m + g_m^2} \tag{4.107}$$

4. Other Applications of OTA

Figure 4.48a shows an OTA with a unity feedback factor whereas Fig. 4.48b shows the configuration used to determine the open-loop gain of the OTA. The resistor/capacitor combination stabilizes the dc output voltage of the OTA in which the MOSFETs used in the OTA must operate in the saturation

region. The feedback resistor should be much larger than the output resistance of the OTA, and the *RC* product should be much longer than the reciprocal of the lowest frequency. This is equivalent to stating that *R* and *C* of Fig. 4.48b should have no effect on the ac performance of the circuit.

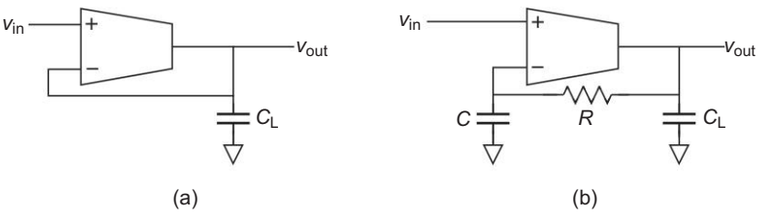


Fig. 4.48 OTA with (a) unity feedback, and (b) dc stability

An op-amp can be designed by using an OTA and buffer circuit. The buffer is used to isolate the OTA from the load. The load can be fully resistive, capacitive, or a combination of both. Compensating the OTA with buffer is somewhat different from compensating the OTA alone. Figure 4.49 shows an OTA with buffer used as a source follower. The output of the amplifier is of low impedance, i.e., the output impedance of the source follower. Therefore, the load capacitance should have little effect on the phase margin of the amplifier. In the figure, gate source capacitance of M_1 is used as an compensating capacitance where as output voltage of OTA is same as that of source follower.

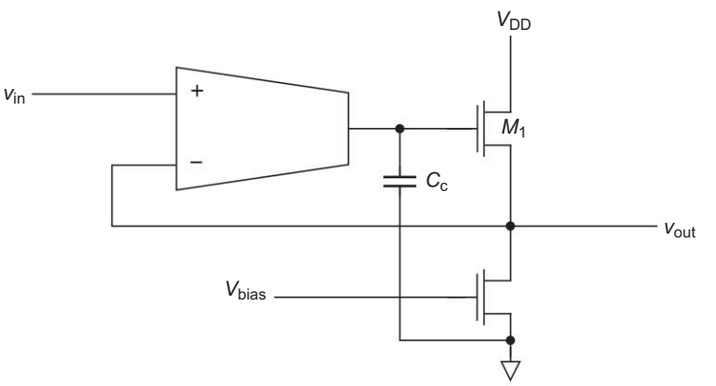


Fig. 4.49 An OTA with source-follower buffer

4.6.5 Differential Output Op-Amp

The diferential output op-amp symbol is shown in Fig. 4.50. The open-loop gain of the op-amp is given in terms of the inputs (v_+ and v_-) and outputs whose gain is written as

$$A_L = \frac{v_{0+} - v_{0-}}{v_+ - v_-} \tag{4.108}$$

whereas for single-ended output op-amp $A_L = \dfrac{v_{0+}}{v_+ - v_-}$ (4.109)

If we ignore v_{0-}, then the differential output op-amp behaves just like a single-ended op-amp. For linear applications, the op-amp is used with a feedback network, the inputs are related by $v_+ = v_-$ and outputs are related to $v_{o+} = v_{o-}$.

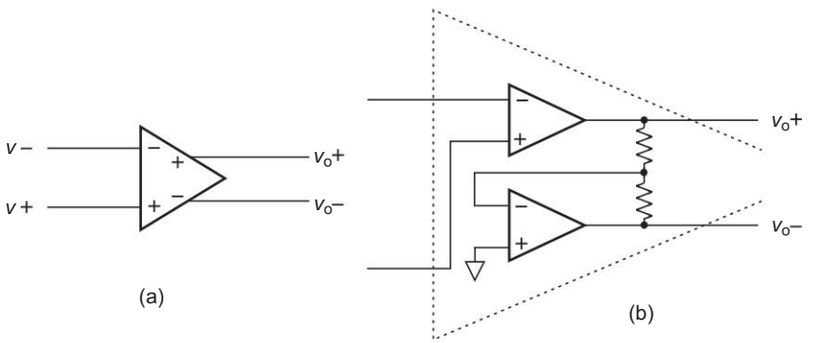


Fig. 4.50 Differential output op-amp: (a) Symbol (b) Differential op-amp using two single-ended op-amp

Figure 4.50(b) shows two single-ended op-amps to form a differential output op-amp. This implementation works at low frequency ranges. However, because the op-amps are in different topologies, the phase response of each op-amp is different. This means that Eq. (4.109) will not hold for higher (> 1 kHz) frequencies because the time delays through each configuration are sufficiently different.

Figure 4.51 shows a cascade of differential output op-amps, without feedback network. The stray capacitance between the interconnecting metal lines which carry the signals between op-amps and the substrate and other noise source are shown. If the metal lines run close to one another then the noise voltage will couple with each signal wire. Since the diff-amp, on the input of the op-amp, rejects common signals, so the coupled noise is not passed to the next op-amp in the string. Variations on the power supply rails are rejected as well. If the differential op-amp is symmetric then changes on the power supply couple evenly into both outputs, having little effect on the difference, or desired signal, coming out of the op-amp. For these reasons, good coupled noise rejection device. So the differential op-amp is a very much useful in any mixed-signal integrated circuit. Mixed-signal means that both analog and digital circuits are present on the chip.

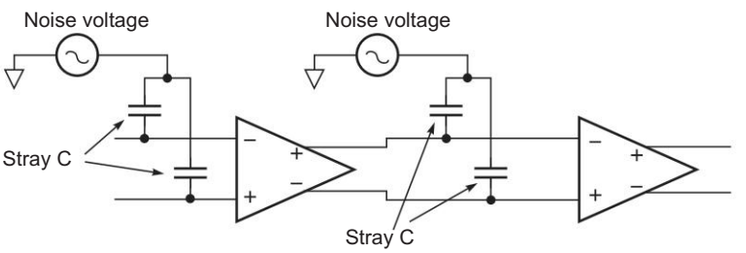


Fig. 4.51 Differential op-amp representing parasitic capacitance and noise

The average output voltage of diff op-amp is called common-mode output voltage and in terms of supply voltage, it is given by

$$V_{CM} = \frac{V_{DD} + V_{SS}}{2} \tag{4.110}$$

The output voltage of diff op-amp can be written to include the common-mode output voltage:

$$v_{0+} + V_{CM} = -v_{0-} + V_{CM} \tag{4.111}$$

4.6.6 Common-Mode Feedback Circuit

A circuit called "common-mode feedback" (CMFB) is used to sense the average value of the op-amp outputs (i.e., V_{CM}). The output of the CMFB circuit is fed back into the op-amp to adjust V_{CM} to the correct value. Figure 4.52 shows CMFB circuit in which the inputs of the circuit are v_{0+} and v_{0-} and the output is V_{CMFB}. The unique aspect of this circuit is its rejection of a difference-mode signal on its inputs and amplification of the common-mode signal. This is exactly the opposite of what a single diff-amp does. To be exact, the CMFB circuit amplifies the difference between the average of the outputs, $(v_{o+} + v_{o-})/2$ and V_{CM}. Through the use of feedback, the averages of the outputs and V_{CM} are made equal. This is similar to how feedback is used to set the inputs of an op-amp, v_+ and v_-, equal. The circuit operates over wide output voltage swings. The average of v_{0+} and v_{0-} is generated with the help of the resistor/capacitor combination and applied to the gate of M_1. This scheme ensures fully balanced outputs over a voltage range limited by the CMR of the buffers.

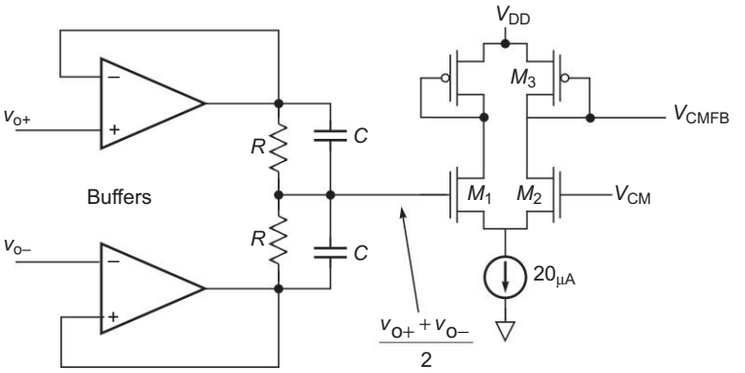


Fig. 4.52 CMFB circuit with improved voltage range

Figure 4.53 shows CMFB circuit with fully folded cascade op-amp which enhances bandwidth with its compensation circuit. The CMFB circuit combined with the folded-cascode OTA forms a feedback loop. As with any feedback loop, we must ensure stability. A common ac signal is applied to both inputs of the CMFB amplifier of the figure resulting in gain equal to approximately unity.

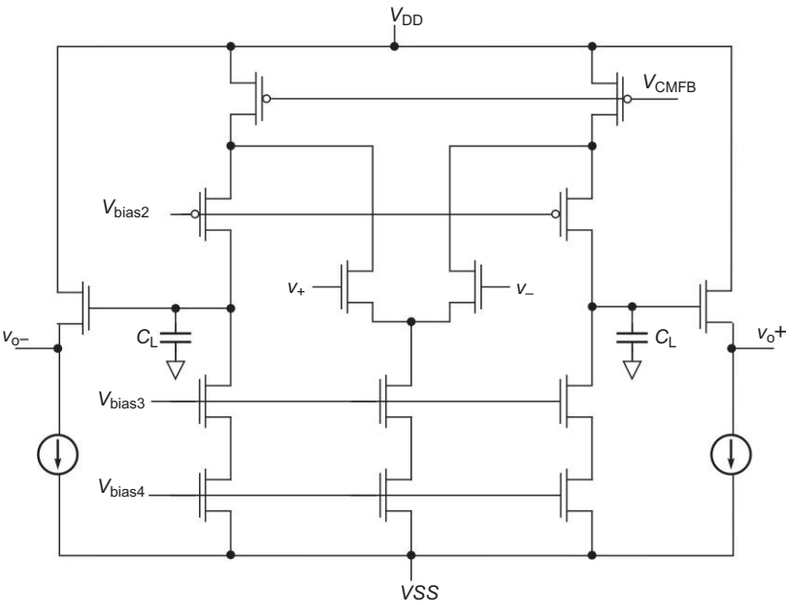


Fig. 4.53 CMFB circuit with fully differential folded cascode op-amp

Figure 4.54 shows the op-amp without output buffers implemented using gain-enhancement techniques. In the circuit of the figure, dc biasing voltages are used. We have not changed the transconductance of the input diff-amp till we can still compensate the OTA for a unity gain frequency of 1 MHz. The gain enhancement simply shifts the open-loop 3 dB frequency of the OTA downward.

The unity gain frequency of the OTAs we have discussed so far is given by $g_m/2\pi C_L$ where g_m is the transconductance of the input diff-amp and C_L is the load capacitance used to compensate the OTA.

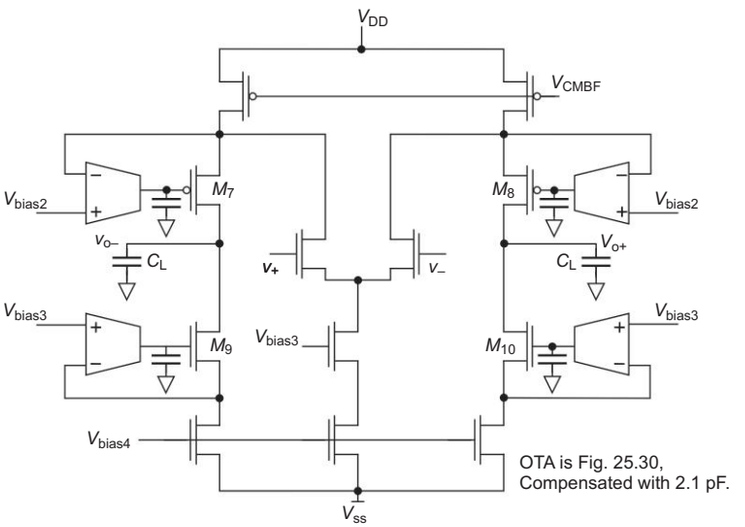


Fig. 4.54 Gain enhancement using OTA

□ REFERENCES

4.1 P.E. Allen and D.R. Holberg, *CMOS Analog Circuit Design,* Holt, Rinehart and Winston, 1987, ISBN 0-03-006587-9.

4.2 R.L. Geiger, P.E. Allen, and N.R. Strader, *VLSI—Design Techniques for Analog and Digital Circuits,* McGraw-Hill Publishing Co., 1990, ISBN 0-07-023253-9.

4.3 P.R. Gray and R.G. Meyer, *Analysis and Design of Analog Integrated Circuits,* 3rd ed., John Wiley and Sons, 1993, ISBN 0-471-57495-3.

4.4 P.R. Gray and R.G. Meyer, "MOS Operational Amplifier Design—A Tutorial Overview," *IEEE Journal of Solid State Circuits,* Vol. SC-17, No. 6, pp. 969–982, December 1982.

4.5 P.W. Tuinenga, *SPICE—A Guide to Circuit Simulation and Analysis Using PSPICE,* 3rd ed., Prentice Hall, 1995, ISBN 0-13-436049-4.

4.6 R. Hogervorst, K.J. de Langen, and J.H. Huijsing, *1.1.2 Low-Power Low-Voltage VLSI Operational Amplifier Cells,* 1996 ISC AS Tutorials.

4.7 A.S. Sedra and K.C. Smith, *Microelectronic Circuits,* Saunders College Publishing, 1991, ISBN 0-03-051648-X.

4.8 P.E. Allen and D.R. Holberg, *CMOS Analog Circuit Design,* Holt, Rinehart and Winston, 1987, ISBN 0-03-006587-9.

4.9 P.J. Hurst, "Exact Simulation of Feedback Circuit Parameters," *IEEE Transactions on Circuits and Systems,* Vol. 38, No. 11, pp. 1382–1389, November 1991.

4.10 R. Castello and P.R. Gray, "A High-Performance Micropower switched-Capacitor Filter," *IEEE Journal of Solid State Circuits,* Vol. SC-20, No. 6, pp. 1122–1132, December, 1985.

4.11 E. Seevinck and R.F. Wassenaar, "A Versatile CMOS Linear Transconductor/Square-Law Function Circuit," *IEEE Journal of Solid State Circuits,* Vol. SC-22, No. 3, pp. 366–377, June 1987.

4.12 M. Steyaert and W. Sansen, "A High-Dynamic-Range CMOS Op Amp with Low-Distortion Output Structure," *IEEE Journal of Solid State Circuits,* Vol. SC-22, No. 6, pp. 1204–1207, December 1987.

4.13 T.S. Fiez, H.C. Yang, J.J. Yang, C. Yu, and D.J. Allstot, "A Family of High-Swing CMOS Operational Amplifiers," *IEEE Journal of Solid State Circuits,* Vol. 24, No. 6, pp. 1683–1687, December 1989.

4.14 J.H. Botma, R.F. Wassenaar, and R.J. Wiegerink, "A Low-Voltage CMOS Op Amp with a Rail-to-Rail Constant-^ Input Stage and a Class AB Rail-to-Rail Output Stage," *Proceedings of the 1993 IEEE ISCAS,* p. 1314.

4.15 A.L. Coban, P.E. Allen, and X. Shi, "Low-Voltage Analog IC Design in CMOS Technology," *IEEE Transactions on Circuits and Systems,* Vol. 42, No.ll, November 1995.

4.16 P.R. Gray and R.G. Meyer, "MOS Operational Amplifier Design—A Tutorial Overview," *IEEE Journal of Solid State Circuits,* Vol. SC-17, pp. 969–982, December 1982.

4.17 L. Geiger and E. Sanchez-Sinencio, "Active Filter Design Using Operational Transconductance Amplifiers: A Tutorial," *IEEE Circuits and Devices Magazine,* pp. 20–32, March 1985.

4.18 M. Steyaert and W. Sansen, "A High-Dynamic-Range CMOS Op-Amp with Low-Distortion Output Structure," *IEEE Journal of Solid-State Circuits*, Vol. SC-22, No. 6, pp. 1204-1207, December 1987.

4.19 T.C. Choi, R.T. Kaneshiro, R. Broderson, and P.R. Gray, "High-Frequency CMOS Switched Capacitor Filters for Communication Applications," *IEEE Journal of Solid State Circuits,* Vol. SC-18, pp. 652–664, December 1983.

4.20 R. Hogervorst, J.P. Tero, R.G.H. Eschauzier, and J.H. Huijsing, "A Compact Power-Efficient 3 V CMOS Rail-to-Rail Input/Output Operational Amplifier for VLSI Cell Libraries," *IEEE Journal of Solid State Circuits,* Vol. 29, pp. 1505–1513, December 1994.

4.21 M. Banu, J.M. Khoury, and Y. Tsividis, "Fully Differential Operational Amplifiers with Accurate Output Balancing," *IEEE Journal of Solid State Circuits,* Vol. 23, No. 6, pp. 1410–1414, December 1988.

4.22 0 K. Bult and G.J.G.M. Geelen, "A Fast-Settling CMOS Op-Amp for SC Circuits with 90-dB DC Gain," *IEEE Journal of Solid State Circuits,* Vol. 25, pp. 1379–1384, December 1990.

4.23 B.Y. Kamath, R.G. Meyer, and P.R. Gray, "Relationship Between Frequency Response and Settling Time of Operational Amplifiers," *IEEE Journal of Solid State Circuits,* Vol. SC-9, pp. 347–352, December 1974.

EXERCISES

4.1 Determine the current that flows in M_3 (Fig. P4.1). Neglect oxide encroachment and the finite output resistance of M_3.

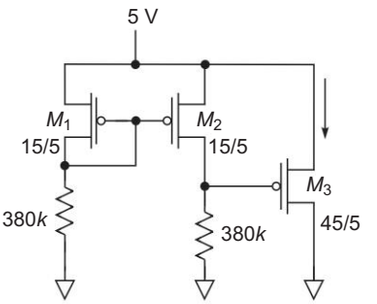


Fig. P4.1

4.2 Determine the currents that flow in Fig. P 4.1 if the size of M_2 is increased to $\frac{30}{5}$. Again, neglect oxide encroachment. Does V_{GS1} of V_{sa} change? Why?

4.3 Determine the ac current that flows in v_t in Fig. P4.2(a).

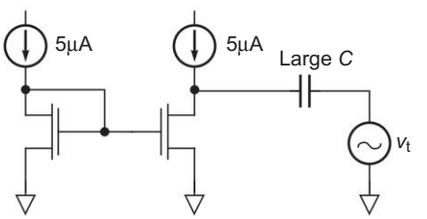


Fig. P4.2(a)

4.4 In order to reduce the power dissipation in current mirrors, one might be tempted to design a circuit with a small diode, connected MOSFET as shown in Fig. P4.2(b). The problem with this design is that changes on the drain of M_2 can have a large effect on the gate voltage of M_1/M_2, ultimately being limited by the gate-source capacitance of M_2. Using SPICE simulations, show how the output current is affected by an abrupt change across M_2 of 1 V and **–1** V.

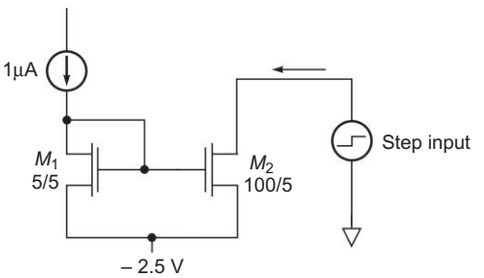


Fig. P4.2(b)

4.5 Suppose that a threshold voltage difference of 10 mV exists between the two MOSFETs used in a current mirror made using equal-size MOSFETs. How would you expect the difference in the drain currents to change with the MOSFET V_{GS}? Is matching between the currents better or worse with large V_{GS}? What happens to the minimum voltage across the current mirror if V_{cs} is increased?

4.6 Show that the gate-drain connected MOSFET shown in Fig. P4.3 behaves like a small-signal resistor of value $1/g_m$.

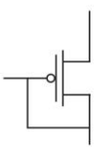


Fig. P4.3

4.7 This problem investigates how the body effect influences the gain of the source-follower amplifier. Consider the source follower shown in Fig. P4.4. Find the gain of this configuration, assuming an ideal current sink.

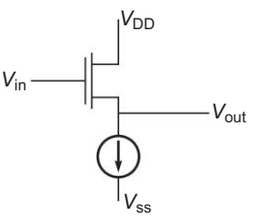


Fig. P4.4

4.8 For the amplifier section shown in Fig. P4.5, estimate the minimum and maximum voltages allowable on the output in order to keep all MOSFETs in the saturation region.

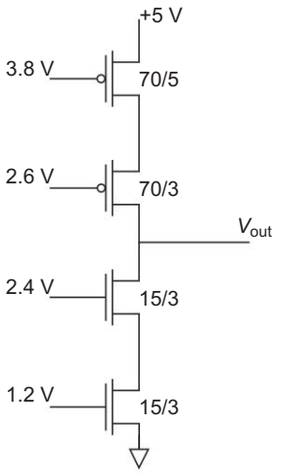


Fig. P4.5

4.9 An op-amp is designed so that the open-loop gain is guaranteed to be 150,000 ± 10 percent V/V. If the amplifier is to be used in a closed-loop configuration with $P = 0.1$ V/V, determine the tolerance of the closed-loop gain.

4.10 What is the maximum possible value of P using resistors in the feedback loop of a noninverting op-amp circuit? Sketch this op-amp circuit when $P = 1/2$.

4.11 Examine the feedback loop in Fig. P4.6. A noise source, v,,, is injected in the system between two amplifier stages. (a) Determine an expression for v_0 which includes both the noise and the input signal, v_s. (b) Repeat (a) for the case where there is no feedback ($p = 0$). (c) If $A_1 = A_2 = 200$, and feedback is again applied around the circuit, what value of P will be required to reduce the noise by one-half as compared to the case stated in (b)?

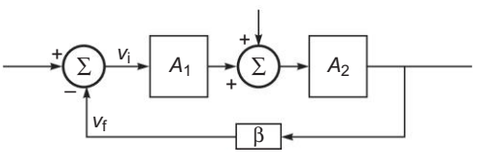


Fig. P4.6

4.12 An amplifier can be characterized as follows:
A series of these amplifiers are connected in cascade, and feedback is used around each amplifier. Determine the number of stages needed to produce an overall gain of 1,000 with a high-frequency rolloff (at–20 dB/decade) occurring at 100,000 rad/s. Assume that the first stage produces the desired high-frequency pole and that the remaining stages are designed so that their high-frequency poles are at least a factor of four greater.

4.13 Make a table summarizing the four feedback topologies according to the following categories: input variable, output variable, units of A_{0L}, units of P, method to calculate $R_{\beta i}$ and $R_{\beta o}$, and expressions for A_{CL}, R_{if}, and R_{of}.

4.14 Using the series-shunt amplifier shown in Fig. P4.7, (a) identify the feedback topology by labeling the mixing variables and output variable, (b) verify that negative feedback is employed, (c) draw the closed-loop small-signal model, and (d) find the expression for the resistors $R_{\beta i}$ and $R_{\beta o}$.

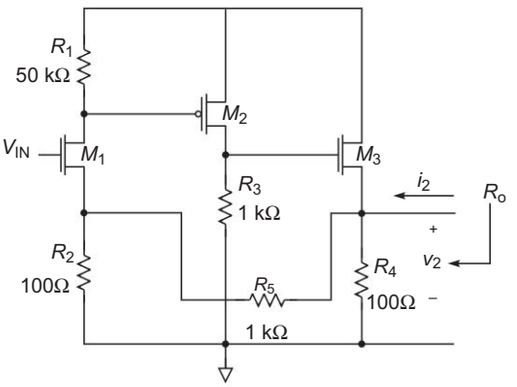


Fig. P4.7

4.15 Using Fig. P4.7 and the results from Problem 4.14, (a) draw the small-signal open-loop model for the circuit, (b) find the expressions for the open-loop parameters, A_{0L}, p\ /?,., and R_o, and (c) find the expressions for the closed-loop parameters, A_{CL} and R_{out}. Note that finding R_{in} is a trivial matter since the signal is input into the gate of M_l.

4.16 Using the series-shunt amplifier shown in Fig. P4.8, (a) verify the feedback topology by labeling the mixing variables and the output variable closed-loop small-signal model, and (b) find the values of $R_{\beta i}$ and $R_{\beta o}$.

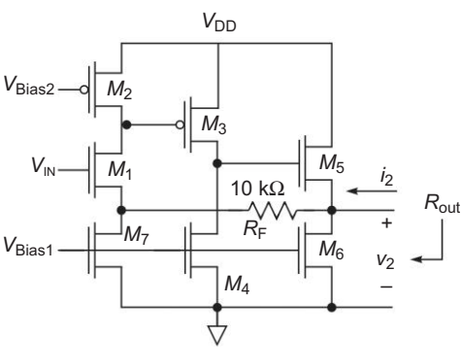


Fig. P4.8

4.17 Using the series-shunt amplifier shown in Fig. P4.8 and the results from Problem 4.16, (a) draw the small-signal open-loop model for the circuit, (b) calculate the open-loop parameters, A_{0L}, β, R_i, and R_0, and (c) Calculate the closed-loop parameters, A_{CL}, and R_{oul}.

4.18 A shunt-shunt feedback amplifier is shown in Fig. P4.9. (a) Identify the feedback topology by labeling the input mixing variables and the output variables. (b) Verify that negative feedback is employed. (c) Draw the closed-loop small-signal model.

4.19 Using the shunt-shunt amplifier shown in Fig. P4.9 and the results from Problem 4.18, (a) draw the small-signal open-loop model for the circuit, (b) calculate expressions for the open-loop parameters, A_{OL}, and R_o, and (c) the closed-loop parameters, A_{CL}, R_{in}, and R_{out}.

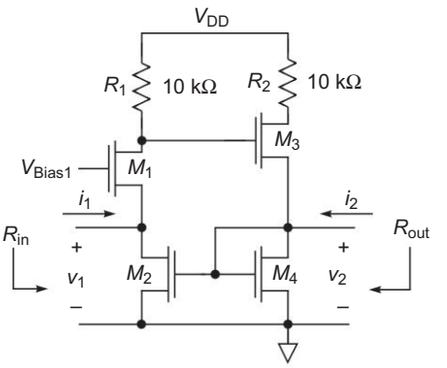


Fig. P4.9

4.20 Convert the shunt-shunt amplifier shown in Fig. P4.9 into a shunt-series feedback amplifier without adding any components. Identify the feedback.

4.21 A feedback amplifier is shown in Fig. P4.10. Identify the feedback topology and determine the value of the voltage gain, R_{in}, and R_{out}.

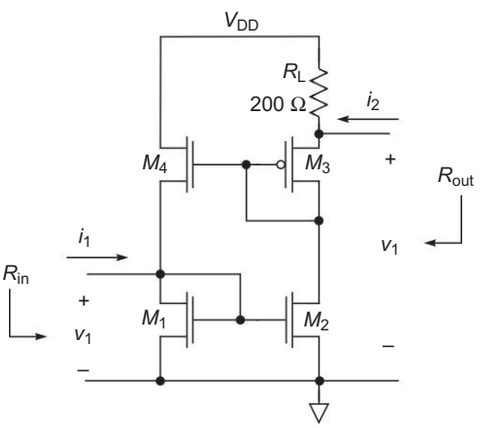


Fig. P4.10

4.22 For the diff-amp shown in Fig. P4.11(a), determine the drain current of M_1 as a function of the input voltage, $v_{i1} - v_{i2}$.

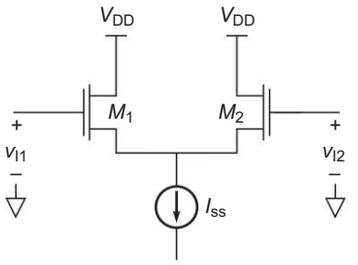


Fig. P4.11(a)

4.23 Determine the small-signal gain and the input common-mode-range (CMR) for the diff-amps shown in Fig. P4.11(b). All *n*-channels are $\frac{15}{5}$, all *p*-channels are 70/5, and the resistors are 3.8 mega-ohms.

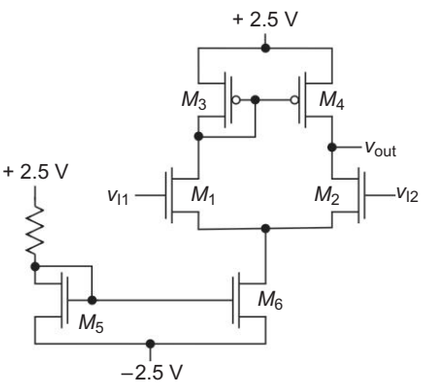


Fig. P4.11(b)

4.24 Determine the rms input noise voltage for the diff-amp of Fig. P4.10 over a bandwidth of 1 to 1 kHz. Assume $KF = 10^{\text{"}25}$ V^2F and $AF = 1.3$.

4.25 Determine the small-signal gain of the amplifier shown in Fig. P4.12. Also determine the input CMR of the amplifier if the minimum voltage across a current source/sink is 0.3 V.

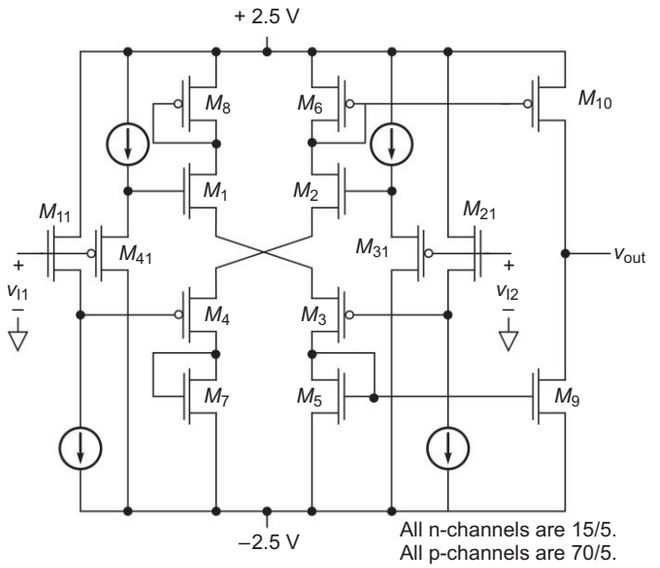


Fig. P4.12

4.26 Determine $v_{out} j v_{in}$ for the circuit of Fig. P4.13. Prove that this circuit is a voltage follower.

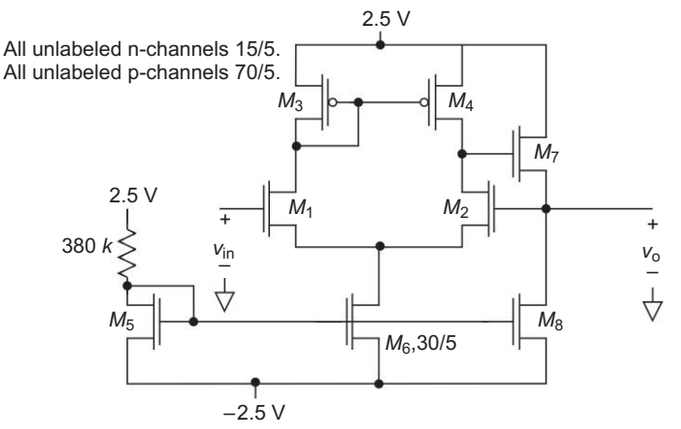


Fig. P4.13

4.27 Calculate the gain of the op-amp shown in Fig. P4.14, if the resistor is replaced with a 1 μA current source.

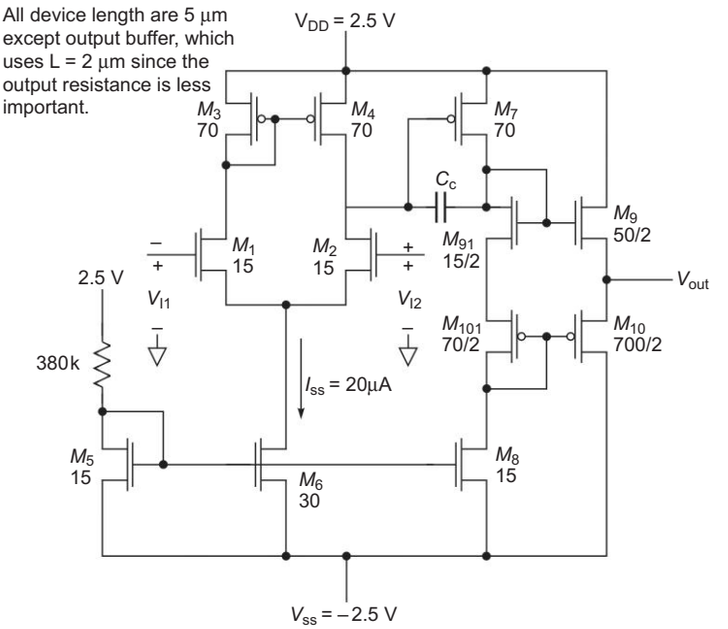


Fig. P4.14

4.28 Determine the maximum output source and sink currents for the op-amp of Fig. P4.15 for a maximum V_{as} (of M_9) or V_{sa} (of M_{10}) of 1.5 V.

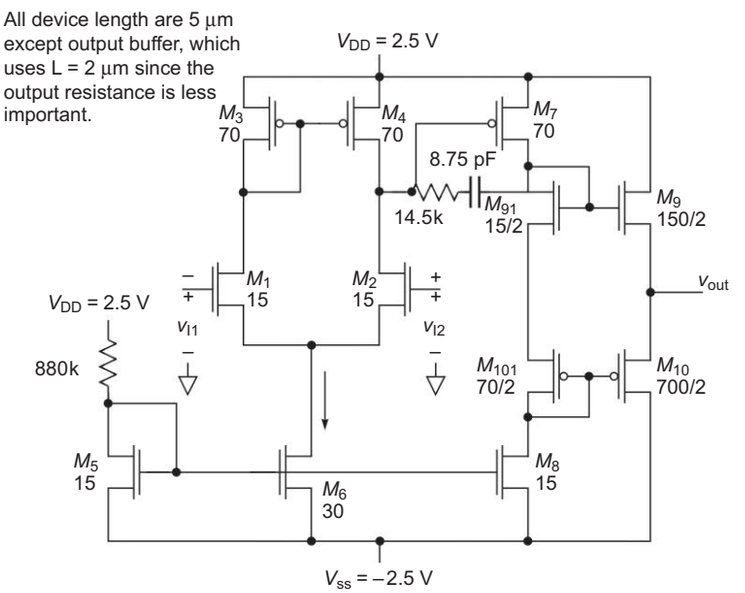


Fig. P4.15

4.29 Referring to the op-amp of Fig. P4.15, determine the small-signal resistance looking into
(a) the sources of M_1 and M_2 (neglect body effect)
(b) the drain of M_6
(c) the source of M_7
(d) the drains of M_1 and M_2
(e) the sources of M_9 and M_{10} (neglect body effect)

4.30 What is the small-signal resistance to ground at the drains of $\frac{M_1}{M_3}$, $\frac{M_4}{M_2}$, and $\frac{M_7}{M_{91}}$ in Fig. P4.15?

4.31 If width of M_7, in Fig. P4.15, is increased to 140 μm, how would M_8 be resized? With these sizes, what quiescent current would flow in M_9 and M_{10}?

4.32 Determine the input offset voltage, the output voltage swing, the CMRR, the power dissipation, PSRR, and the slew-rate limitations for Fig. P4.15.

4.33 If the OTA of Fig. P4.16 is made using 15/5 *n*-channels, 70/5 *p*-channels, and biased with a 10 uA current source, determine the small-signal, low-frequency gains $i_{out}/(v_{i1} - v_{i2})$ and $v_{out}/(v_{i1} - v_{i2})$.

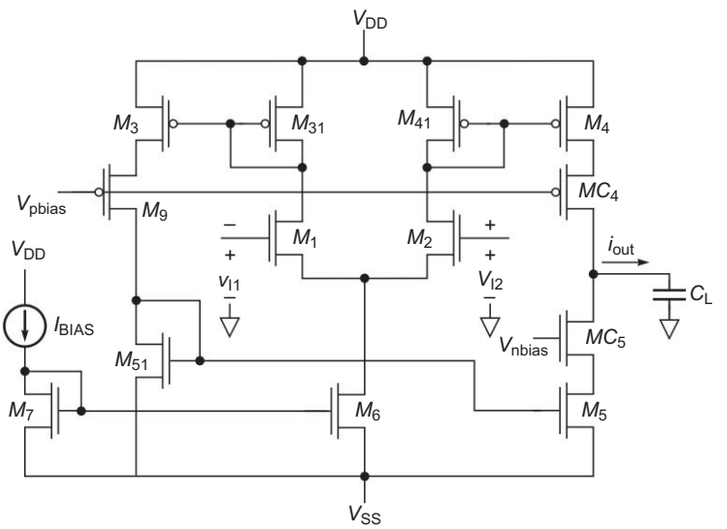


Fig. P4.16

4.34 Determine, using hand-calculations, the small-signal, low-frequency voltage gains of the OTA shown in Fig. P4.17 with V_{CM} = 0.8 V, 2.5 V, and 4.2 V.

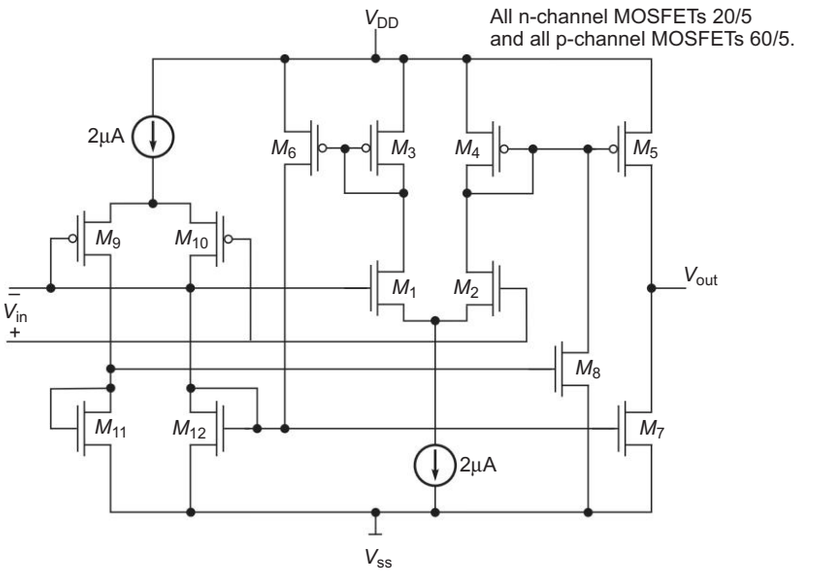


Fig. P4.17

4.35 Estimate the small-signal gains, i_0/v_{in} and v_0/v_{in}, of the OTA described in Fig. P4.18. What size of load capacitance would be needed to compensate the op-amp for f_u of 1 MHz?

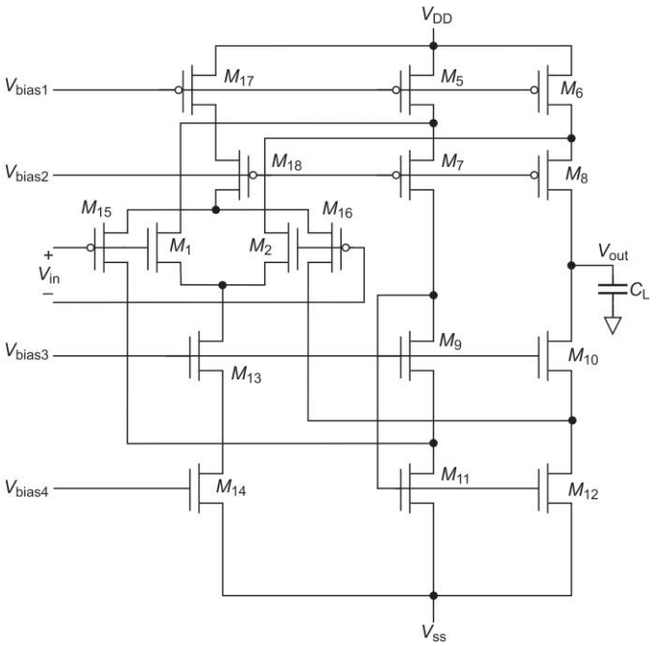


Fig. P4.18

4.36 Fig. P4.19 shows a floating current source made using MC_I and MC_2. Determine the size of MC_1/MC_2 so that 30 μA flows in M_5 and M_9.

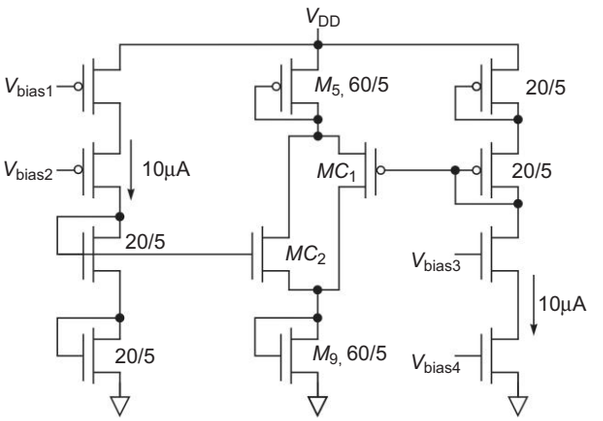


Fig. P4.19

5

CMOS Mixed Signal Circuit

The circuits dealing with both analog and digital signals are called mixed signal circuits. The design of these circuits is called as mixed signal circuit design. In this chapter, we discuss several circuits that are neither purely analog nor digital. Mixed signal circuits are mainly nonlinear circuits such as voltage comparators, adaptive biasing, and analog multiplexer.

5.1 ADAPTIVE BIASING

Adaptive biasing is a mixed signal circuit used for biasing of MOSFETS of amplifiers because of lowering power dissipation. Figure 5.1 shows an adaptive biasing differential amplifier circuit. When v_{I1} and v_{I2} are equal, the current sources I_{SS1} and I_{SS2} are zero and the amplifier takes current from I_{SS}. If v_{I1} becomes larger than v_{I2}, the current source I_{SS1} increases above zero, effectively increasing the diff-amp dc bias current and when v_{I2} becomes larger than v_{I1}, the current source I_{SS2} increases above zero. The maximum output current is limited to either $I_{SS} + I_{SS1}$ or $I_{SS} + I_{SS2}$.

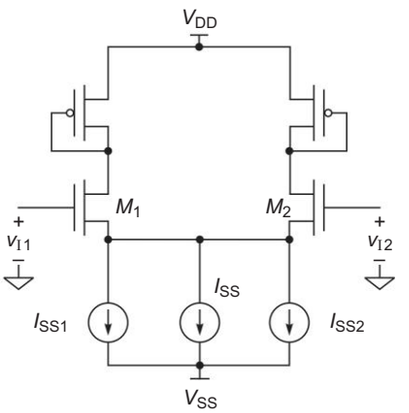


Fig. 5.1 Adaptive biased differential amplifier

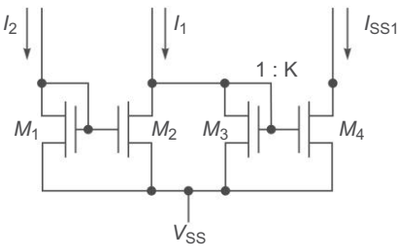


Fig. 5.2 Adaptive biasing for differential amplifier

The current diff-amp of Fig. 5.2 can be used to implement the current source for reduction of power dissipation by using adaptive biasing. If the currents I_1 and I_2 are equal then zero current flows in M_3 and M_4. When I_2 is greater than I_1, and then also, zero current flows in M_3 and M_4. If I_1 is larger than I_2, the difference between these two currents $(I_1 - I_2)$ flows in M_3.

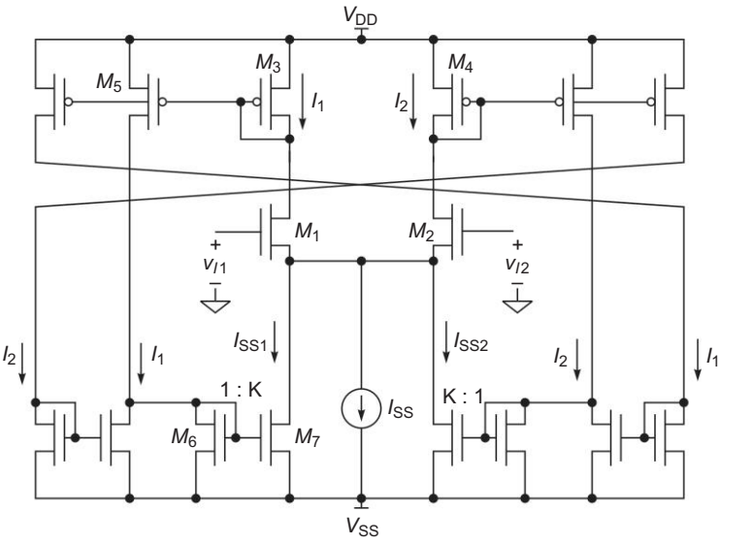


Fig. 5.3 Adaptively biased OTA

The same type of adaptive biasing can be used in differential amplifiers of OTA (Fig. 5.3) making K (= ratio of width of M_4 and M_3) equal to one which eliminates slew-rate limitations while at the same time not increasing static power dissipation. In practice, shutting off one side of the diff-pair, M_1/M_2, is difficult since the adaptive biasing has the effect of lowering the source potentials of the diff-pair, keeping both MOSFETs on. So for applying adaptive biasing to an OTA design, the value of K should be unity or less.

An adaptive biasing is also used in a voltage-follower amplifier which is shown in Fig. 5.4. This amplifier can only source current to a load. If v_{in} and v_{out} are equal, the current that flows in M_1 and M_2 is $I_{SS} + I_{D6}$. If v_{tn} is increased, the current in M_1 and M_3 increases. This causes the currents in M_4–M_6 to increase, effectively increasing the current of the diff-pair. The result is a large current available to

drive the load. Note that M_7 can be sized larger than the other MOSFETs to increase maximum output current.

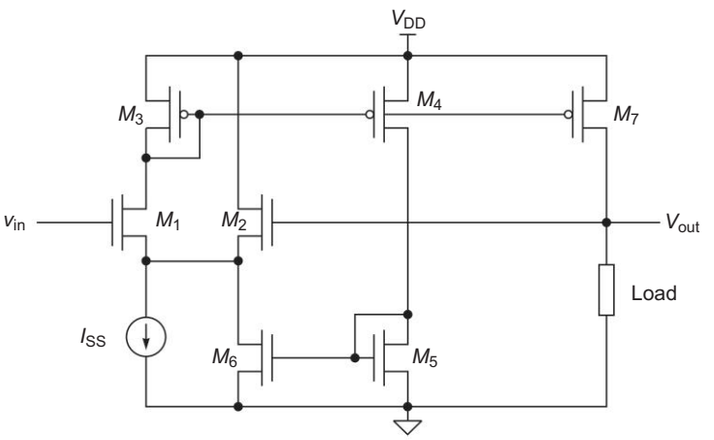


Fig. 5.4 Adaptive biasing in a voltage-follower amplifier

5.2 □ CMOS COMPARATOR DESIGN

Figure 5.5 shows basic block of a voltage comparator with its symbol which can be used as a decision-making circuit. The basic operation of a voltage comparator is as follows : If the positive voltage input (v_+) of the comparator is at a greater potential than the v_- input, the output of the comparator is a logic 1, whereas if the v_+ input is at a potential less than the v_- input, the output of the comparator is at a logic 0.

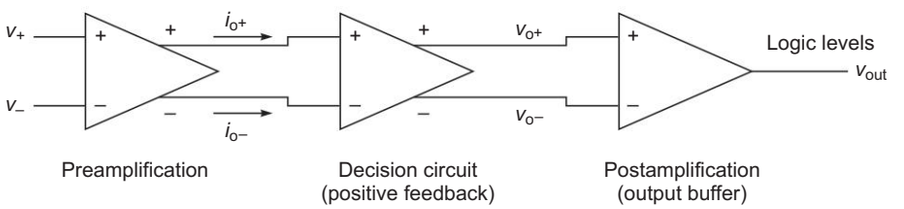


Fig. 5.5 Voltage comparator

The comparator consists of three stages: input amplifier, decision stage, and output buffer. The **input amplifier** is a pre-amplification stage which amplifies the input signal to increase the comparator sensitivity and isolate the input from noise originated with next stage. The next is the **decision-making stage** which is basically positive feedback used to determine larger input signals. The final stage is the **output buffer** which is the post-amplification stage, amplifying the information coming from the decision-making stage, and the output is a digital signal. In designing a comparator, the main considerations are input common-mode range, power dissipation, propagation delay, and comparator gain. Figure 5.6 shows CMOS-based voltage comparators having all three stages.

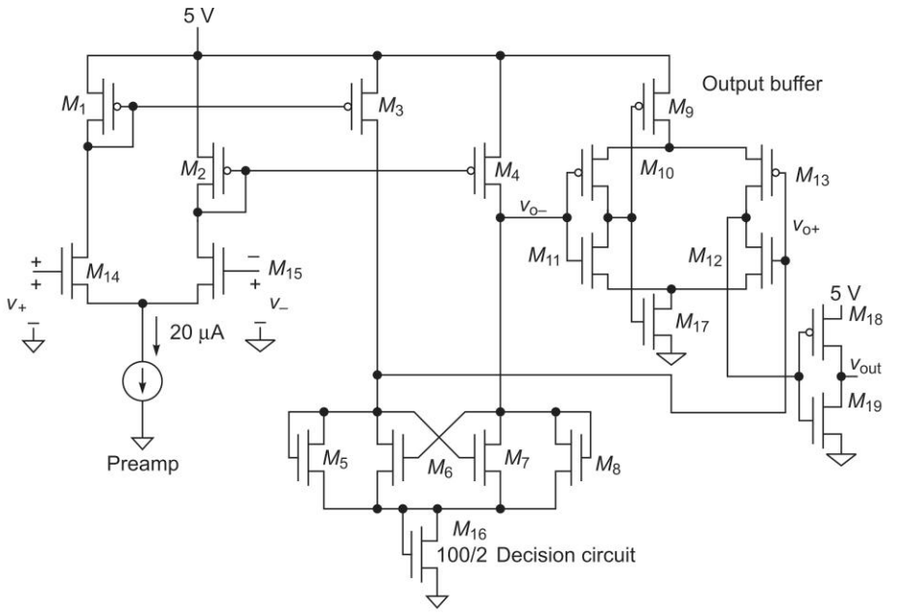


Fig. 5.6 CMOS voltage comparator

1. Pre-amplification

The pre-amplification circuit shown in Fig. 5.6 is a differential amplifier with active loads. The transconductance sets the gain of the stage, while the input capacitance of the comparator is determined by diff-amplifier MOSFETs. As per consideration of speed for comparator, the channel lengths of the MOSFETs are set (channel-length modulation gives rise to an unwanted offset voltage.). The output currents are written as

$$i_{0+} = \frac{g_m}{2}(v_+ - v_-) + \frac{I_{SS}}{2} = I_{SS} - i_{0-} \tag{5.1}$$

2. Decision Circuit

The decision circuit is the main part of the comparator which uses positive feedback from the cross-gate connection of M_6 and M_7 to increase the gain of the decision element. If we start to increase $i_{0_}$ and decrease i_{o+}, switching takes place when the drain-source voltage of M_7 is equal to V_{THN} of M_6. At this point, M_6 starts to take current away from M_5. This decreases the drain-source voltage of M_5 and thus starts to turn M_7 off. If we assume that the maximum value of v_{a+} or v_{0-} is equal to $2V_{THN}$, then M_6 and M_7 operate. Under these circumstances, the voltage across M_7 reaches V_{THN} when the current through M_7 is

$$i_{0-} = \frac{\beta_2}{2}(v_{0+} - V_{THN})^2 = \frac{\beta_2}{\beta_1} i_{0+} \tag{5.2}$$

where $\beta_2 = \beta_5 = \beta_8$ and $\beta_1 = \beta_6 = \beta_7$. The switching-point voltage is written as

$$V_{SPH} = v_+ - v_- = \frac{I_{SS}}{g_m} \cdot \frac{\beta_2/\beta_1 - 1}{\beta_2/\beta_1 + 1} = -V_{SPL} \tag{5.3}$$

3. Output Buffer

The final stage in a voltage comparator is the output buffer or post-amplifier which is used to convert the output of the decision circuit into a logic signal (i.e., 0 or 5 V). The output buffer circuit is a self-biasing differential amplifier. An inverter was added on the output of the amplifier as an additional gain stage and to isolate any load capacitance from the self-biasing differential amplifier. The MOSFET M_{17} is added in series with the decision circuit to increase the average voltage out of the decision circuit.

4. Clocked Comparators

Figure 5.7 shows clocked comparators used in analog-to-digital converter design. The use of a clock pulse can greatly improve comparator performance and speed. The use of a clock eliminates the need for an output buffer used in level shifting the output of the decision circuit. The circuit is based on the regenerative latch. The comparator is made up of three stages: pre-amp, positive-feedback decision circuit, and output buffers. The regenerative latch (cross-coupled inverters) is made up of M_1 through M_4. When *latch* is low, the *p*-channels, M_1 and M_2, of the latch are isolated from the *n*-channels, M_3 and M_4. In addition, the outputs of the decision circuit are pulled high, i.e., the outputs of the comparator are low, $Q = \overline{Q} = 0$. When *latch* signal transitions are high, the regenerative action of the latch combined with the pre-amplifier causes an imbalance in the decision circuit, forcing the outputs into a state determined by v_+ and v_-.

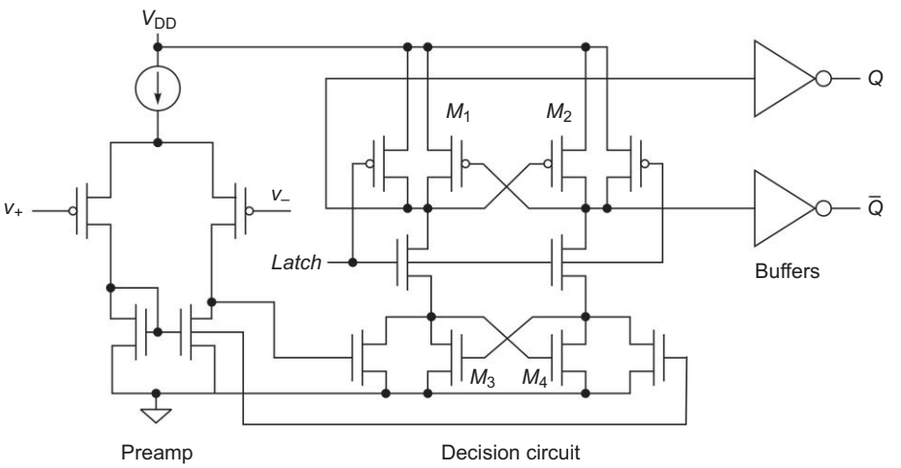


Fig. 5.7 Clocked comparator

5.3 ANALOG MULTIPLIERS

Analog multipliers have more applications in communication systems. Figure 5.8 shows the block diagram of an analog multiplier in which ideally, the output of the multiplier is related to the inputs by

$$v_{out} = K_m v_x v_y \tag{5.4}$$

where K_m is the multiplier gain with units of V^{-1}. In reality, the output of the multiplier has imperfections resulting in offsets and nonlinearities. The output of the multiplier can be written by including these offset and nonlinear terms.

$$v_{out} = K_m (v_x + V_{osx}).(v_y + V_{osy}) + V_{osout} + v_x^m + v_y^n \tag{5.5}$$

where, V_{osx}, V_{osy}, and V_{osout} are offset voltages for x and y input and output respectively. The terms v_x^m, v_y^n.

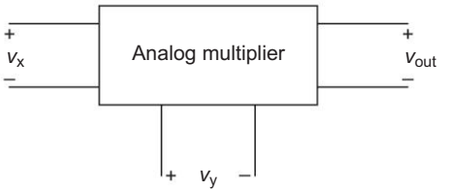


Fig. 5.8 Analog multiplier

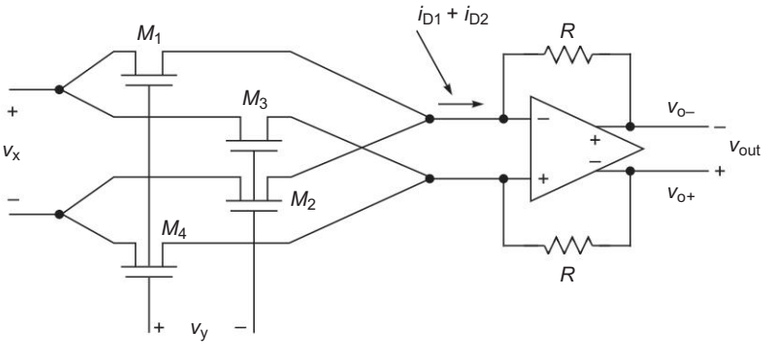


Fig. 5.9 CMOS analog multiplier without biasing

The Multiplying Quad

Figure 5.9 shows a CMOS multiplier (without biasing) having a multiplying quad ($M_1 - M_4$). The multiplying quad operates in the triode region, and thus MOSFETs $M_1 - M_4$ act as active resistors. The negative output voltage of the multiplier is written as

$$v_{0-} = -R(i_{D1} + i_{D2}) \tag{5.6}$$

where as the positive output voltage is written as

$$v_{0+} = -R(i_{D3} + i_{D4}) \tag{5.7}$$

The output voltage of the multiplier is written as

$$v_{out} = v_{0+} - v_{0-} = R(i_{D1} + i_{D2} - i_{D3} - i_{D4}) \tag{5.8}$$

Figure 5.10 shows the multiplying quad with biasing in which the op-amp inputs are at an ac virtual ground and at a dc voltage of V_{CM} (the op-amp output common-mode voltage). In order to minimize the dc input current on the *x*-axis inputs, the common-mode dc voltage on this input is set to V_{CM}. The dc biasing voltage on the *y*-input is set to a value large enough to keep the quad in triode. The input signals have been broken into two parts (e.g., $v_x/2$ and $-v_x/2$) for general analysis where the minus inputs can be connected directly to the bias voltages at the cost of large-signal linearity.

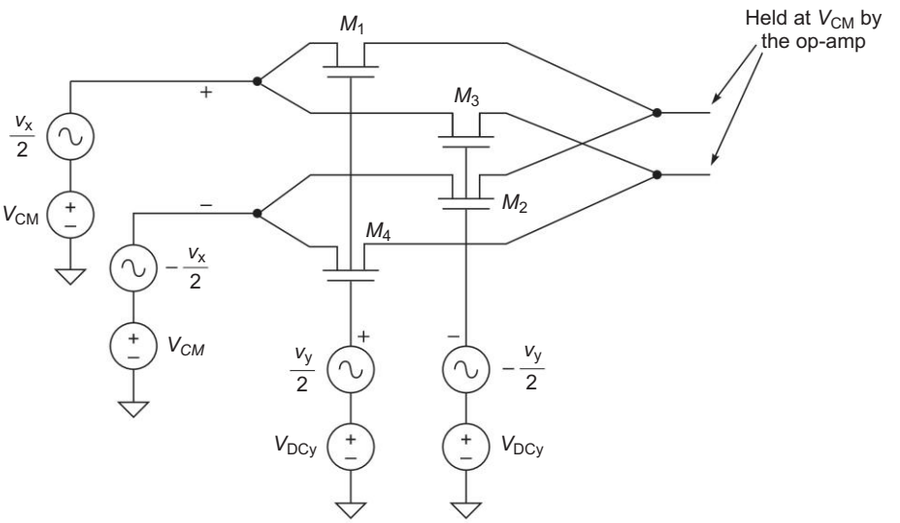


Fig. 5.10 CMOS analog multiplier

$$i_{D1} = \beta_1 \left[\left(V_{GS} + \frac{v_y}{2} - V_{THN} \right) . \left(\frac{v_x}{2} \right) - \frac{1}{2} \left(\frac{v_x}{2} \right)^2 \right] \tag{5.9}$$

$$i_{D2} = \beta_2 \left[\left(V_{GS} + \frac{v_y}{2} - V_{THN2} \right) . \left(-\frac{v_x}{2} \right) - \frac{1}{2} \left(-\frac{v_x}{2} \right)^2 \right] \tag{5.10}$$

$$i_{D3} = \beta_3 \left[\left(V_{GS} + \frac{v_y}{2} - V_{THN3} \right) . \left(\frac{v_x}{2} \right) - \frac{1}{2} \left(-\frac{v_x}{2} \right)^2 \right] \tag{5.11}$$

$$i_{D4} = \beta_4 \left[\left(V_{GS} + \frac{v_y}{2} - V_{THN4} \right) . \left(-\frac{v_x}{2} \right) - \frac{1}{2} \left(-\frac{v_x}{2} \right)^2 \right] \tag{5.12}$$

Using the above current equations in output voltage, we can write

$$v_{out} = R\beta \left(\frac{v_x}{2} \right) \left[\frac{v_y}{2} - V_{THN1} + \frac{v_y}{2} + V_{THN2} + \frac{v_y}{2} + V_{THN3} + \frac{v_y}{2} - V_{THN4} \right] \tag{5.13}$$

where $\beta_1 = \beta_2 = \beta_3 = \beta_4 = \beta$. Considering threshold voltages $V_{THN1} = V_{THN2} = V_{THN3} = V_{THN4}$, we can write

$$v_{out} = R\beta . v_x . v_y = K_m . v_x . v_y \tag{5.14}$$

When the sources of the MOSFETs are connected to the op-amp, all the MOSFETs in the multiplying quad have the same threshold voltage. Since the source of each MOSFET is tied to the same potential, the body effect changes each MOSFET's threshold voltage by the same amount.

5.4 □ LEVEL SHIFTING

Level-shifting stages are used to implement the biasing batteries for single-ended to differential conversion since they have many applications in single-supply chip design. Figure 5.11 shows the basic *p*-channel source-follower circuits for level shifting. The source-gate voltages of the *p*-channel MOSFETs are used to shift the input signals, which are referenced to ground, upward. This circuit can be used in implementing the *x*-input level shifter in our analog multiplier. The *x*-inputs can actually go negative by V_{THP} before M_1 or M_2 go into the triode region.

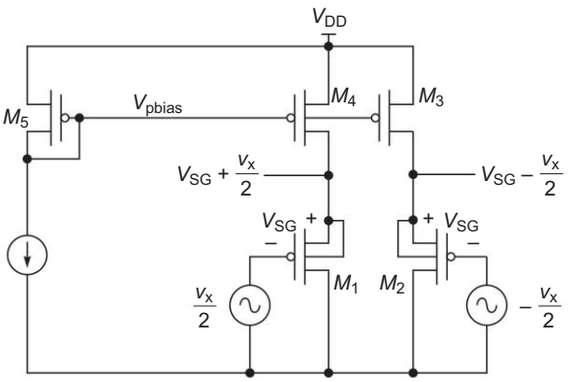


Fig. 5.11 Level shifting using *p*-channel source

The level-shifting circuit can be implemented for biasing in analog multiplier as shown in Fig. 5.11. This level-shifting configuration is wide-band since all MOSFETs are operated in the source-follower configuration. Because of less than unit gain of the source-followers, the overall gain of the multiplier is less.

5.5 □ DYNAMIC MIXED SIGNAL CIRCUIT

Dynamic CMOS mixed signal circuits are useful in storing information in gate capacitance of MOSFET. These circuits are sample-and-hold circuit, current mirrors, amplifiers, filters etc.

5.5.1 MOSFET Switch

A fundamental component of any dynamic circuit (analog or digital) is the switch. An important attribute of the switch in CMOS (shown in Fig. 5.12) is that under dc conditions the gate of the MOSFET does not draw a current. The benefits of using the CMOS transmission gate are seen from this figure, namely, lower overall resistance. Another benefit of using the CMOS TG is that it can pass a logic high or a low without

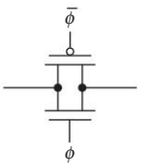


Fig. 5.12 CMOS transmission gate switch

a threshold voltage drop. The largest voltage for an *n*-channel switching is $V_{DD} - V_{THN}$, whereas the lowest voltage of switching for a *p*-channel switch is V_{THP}.

5.5.2 Sample-and-Hold Circuits

An important application of the switch is in the sample-and-hold (S/H) circuit. The sample-and-hold circuit is used in data-converter applications as a sampling gate. Figure 5.13 shows a simple sample-and-hold circuit. A narrow pulse is applied to the gate of the MOSFET which enable v_{in} to charge the hold capacitor, C_H. The width of the gate pulse should be such that it allows the capacitor to fully charge before being removed. In the figure, the op-amp acts as a unity gain buffer, isolating the hold capacitor from any external load. This circuit suffers from the clock feed through and charge injection problems.

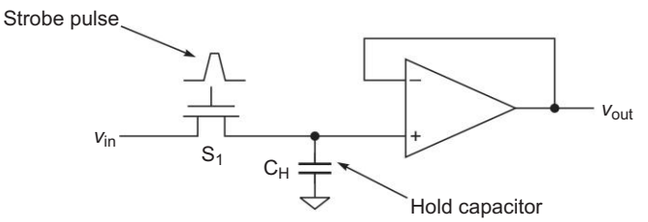


Fig. 5.13 Simple sample-and-hold circuit

Figure 5.14 shows a fully differential sample-and-hold circuit and associated clock waveforms that eliminate clock feed through and charge injection. The switches in this figure are closed when their controlling clock signals are high. The basic operation can be understood by considering the state of

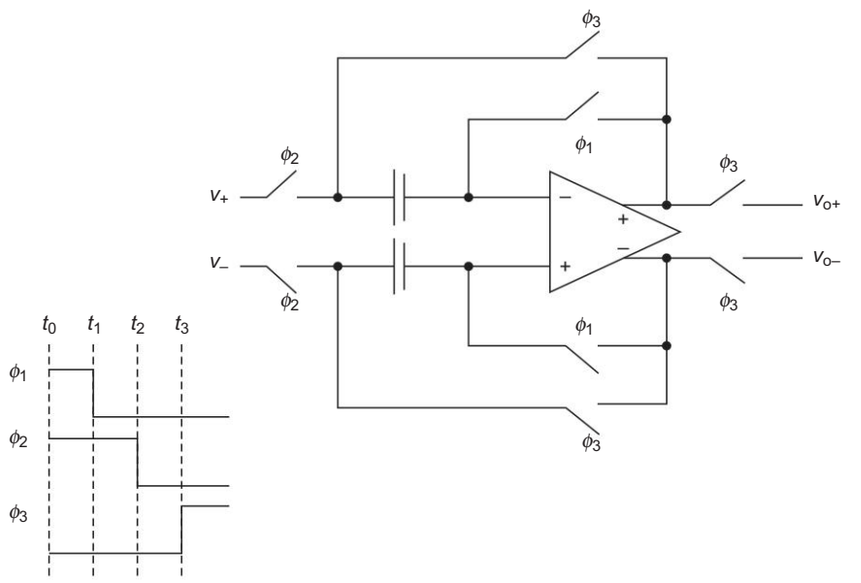


Fig. 5.14 Sample-and-hold using differential topology

the circuit at t_0. At this time, the input signals charge the sampling capacitors. The bottom plates of the capacitors (polyl) are tied directly to the input signals, for reasons that will be explained below. The op-amp is operating in a unity-follower configuration in which both inputs of the op-amp are held at V_{CM}. At this particular instance in time, prior to f_1, the amplifier is said to be operating in the sample mode of operation.

At t_1, the ϕ_1 switches turn off. The resulting charge injection and clock feed through appear as a common-mode signal on the inputs of the op-amp and are ideally rejected. Since the top plates of the hold capacitors (the inputs to the op-amp) are always at V_{CM}, at this point in time the charge injection and clock feed through are independent of the input signals. The result is an increase in the dynamic range of the sample-and-hold (the minimum measurable input signal decreases). The voltage on the inputs of the op-amp (the top plate of the capacitor) between t_1 and t_2 is $V_{0FFl} + V_{CM}$, a constant voltage. The op-amp is operating open loop at this time so that the time between t_1 and t_3 should be short.

At t_2, the ϕ_2 switches turn off. At this point, the voltages on the bottom plates of the sampling capacitors are v_+ and v_- for the + and – inputs of the circuit, respectively. The voltages on the top plates of the capacitors are $V_{OFF1} + V_{0FF2} + V_{CM}$ (assuming the storage capacitors are much larger than the input capacitance of the op-amp). The term V_{OFF2} is ideally a constant that results from the charge injection and capacitive feed through from the ϕ_2 switches turning off. The time between t_1 and t_2 should be short compared to variations in the input signals.

At time t_3, the ϕ_3 switches turn on and the op-amp behaves like a voltage follower, and the circuit is said to be in the hold mode of operation. The charge injection and clock feed through resulting from the φ_3 switches turn on causing the top plate of the capacitor to become $V_{OFF1} + V_{OFF2} + V_{OFF3} + V_{CM}$, again assuming that the storage capacitors are much larger than the input capacitance of the op-amp. The outputs of the sample-and-hold are v_+ and v_-, assuming infinite op-amp gain since these offsets appear as a common-mode voltage on the input of the op-amp. Note that the terms V_{0FF2} and V_{OFF} are dependent on the input signals.

Another improvement of the basic S/H circuit can be seen in Fig. 5.15. Here, two amplifiers buffer the input and the output. Notice that switch S_2 ensures that amplifier A_1 is stable while in hold mode. If the switch were not present, amplifier A_1 would be open loop during hold mode. During the next sampling mode, it would then be slew limited while going from the supply to the value of v_{in}. With the switch S_2, the output of amplifier A_1 tracks v_{in} even while in hold mode. The switch S_3 also disconnects A_1 from the output during hold mode. This S/H has its disadvantages, however. The capacitor is still subjected to charge injection and clock-feed through problems. In addition, during sample mode, the circuit may become unstable since there are now two amplifiers in the single-loop feedback structure. Although compensation capacitors can be added to stabilize its performance, the size and placement of the capacitors are purely dependent on the type and characteristics of the op-amps.

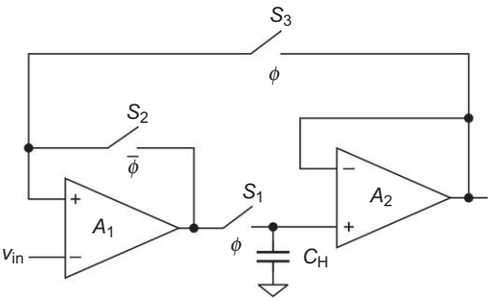


Fig. 5.15 Closed-loop S/H circuit with two op-amps

Figure 5.16 shows a S/H circuit using transconductance amplifier which removes the problems of previous S/H circuit. In the figure, the hold capacitor is actually in the feedback path of the amplifier, A_2, with one side connected to the output of the amplifier and the other connected to a virtual ground. When switch S_1 turns off, any charge injected onto the hold capacitor will result in a slight change in the output voltage. Since one side of the switch is at virtual ground, the change in voltage is no longer dependent on the threshold voltage of the switch itself. Therefore, the charge injection will be independent of the input signal and will result as a simple offset at the output.

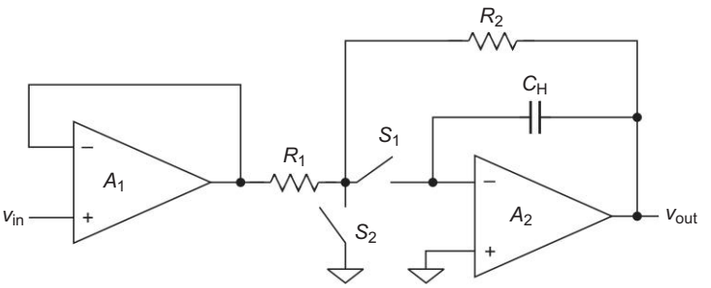


Fig. 5.16 A closed S/H circuit using transconductance amplifier

When sampling, S_1 is closed and S_2 is open, and the equivalent circuit is simply a low-pass filter with a buffered input and transfer function written as

$$\frac{v_{\text{out}}}{v_{\text{in}}} = \frac{R_2}{R_1} \cdot \frac{1}{(sR_2C_{\text{H}} + 1)} \tag{5.15}$$

This circuit acts as a low-pass filter function while sampling. The buffer op-amp A_1 can be eliminated when we desire a low-input impedance. Once hold mode commences, the output will stay constant at a value equal to v_{in}, while the switch S_2 isolates the input from the hold capacitor. During both sample mode and hold mode, there is only one op-amp in each feedback loop, so this S/H topology is much more stable than other closed-loop S/H circuit.

5.5.3 Switched-Capacitor Circuits

Figure 5.17 shows the dynamic circuit, named a switched-capacitor resistor. The clock signals ϕ_1 and ϕ_2 are two non-overlapping clock signals with frequency f_{clk} and period T. When ϕ_1 is high, the capacitor is charged to v_1 and can be written as $C.\text{v}_1$. Similarly while ϕ_2 is high, the capacitor is charged by $q_2 = C.\text{v}_2$. Due to non-overlapping of clocked signals, a charge difference $q_1 - q_2$ is transferred between v_1 and v_2 during time interval T. The average current transferred in time interval is written as

$$I_{\text{avg}} = \frac{C(v_1 - v_2)}{T} = \frac{v_1 - v_2}{R_{\text{sc}}} \tag{5.16}$$

Where switched capacitor resistance is

$$R_{\text{sc}} = \frac{T}{C}$$

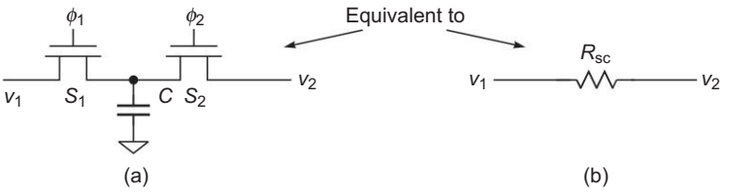


Fig. 5.17 Switched capacitor circuit

1. Switched-Capacitor Integrator

The switched-capacitor resistor is sensitive to parasitic capacitances and finds little use in many switched-capacitor circuits. One of the circuits is a switched-capacitor integrator which is shown in Fig. 5.18. The portion of the circuit consisting of switches S_1 through S_4 and C, forms a switched-capacitor resistor r with a value given by

$$R_{sc} = \frac{T}{C_1} \tag{5.17}$$

The transfer function of the switched-capacitor integrator is given by

$$\frac{v_{out}}{v_{in}} = \frac{1}{i\omega\left(\frac{T.C_F}{C_1.}\right)} \tag{5.18}$$

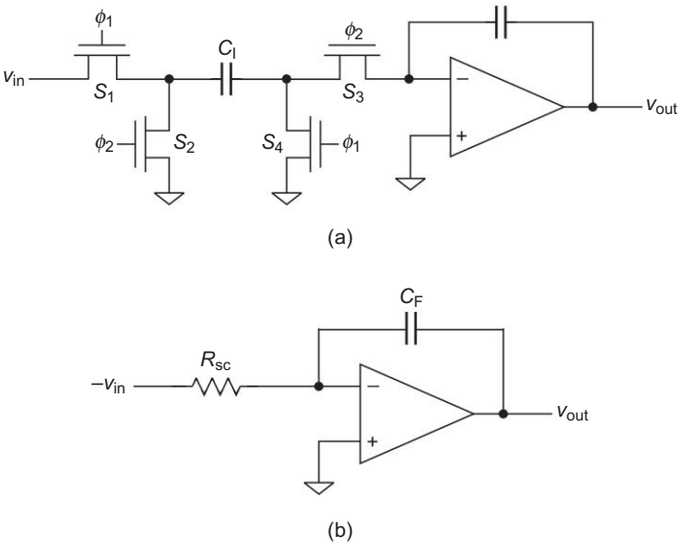


Fig. 5.18 Switched capacitor integrator

2. Switched-capacitor filter

Fig. 5.19 shows a switched-capacitor filter which is a lossy integrator. The output voltage at time nT is $V_{out}(n)$ whereas output voltage at time $(n + 1)T$ is $V_{out}(n + 1)$ which is written as

$$v_{out}(n+1) = v_{out}(n) + \frac{C_1}{C_F} v_{in}(n) \tag{5.19}$$

After taking Fourier transform of Eq. (5.19), we can write

$$\frac{v_{out}(j\omega)}{v_{in}(j\omega)} = \frac{C_1}{C_F}\left[\frac{1}{e^{j\omega T} - 1}\right] \tag{5.20}$$

$$e^{j\omega T} v_{out}(j\omega) = v_{out}(j\omega) + \frac{C_1}{C_F} v_{in}(j\omega) \tag{5.21}$$

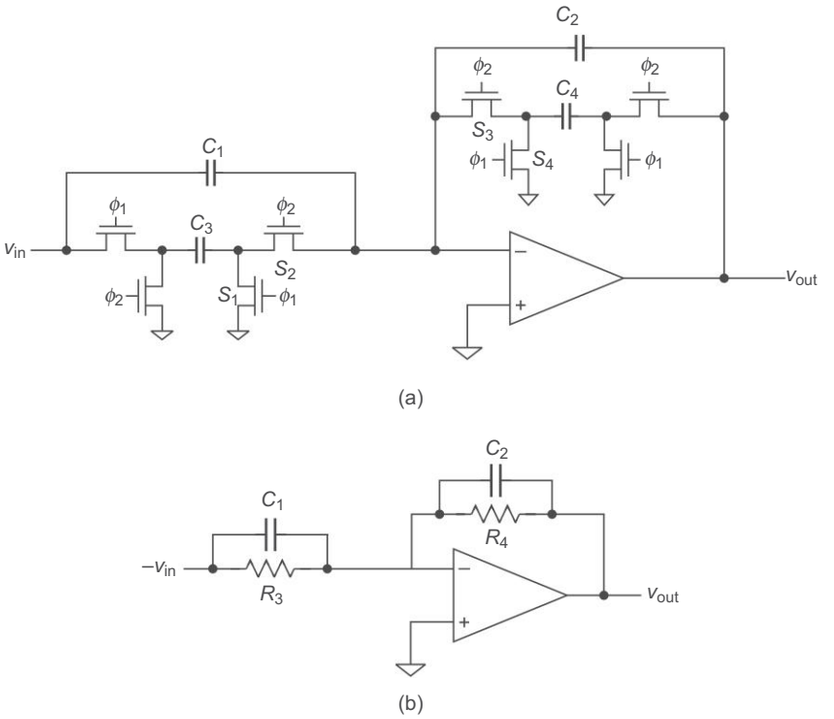


Fig. 5.19 Switched-capacitor filter

5.5.4 Dynamic Reduction Circuit for Offset Voltage

The elimination of op-amp offset voltage is done by adding a dc voltage in series with the non-inverting input of the op-amp. A capacitor is used to cancel offset voltage. It is charged to a voltage equal and opposite to the comparator offset voltage as shown in Fig. 5.20. The dynamic analog circuit shown in Fig. 5.21 is used to reduce the effects of the offset voltage. The clock signals ϕ_1, and ϕ_2 are the

non-overlapping clock signals which keeps switches S_1, S_2, and S_3 from being on at the same time as switches S_4 and S_5. The op-amp, via the negative feedback, forces its output to zero volts. Doing so, the capacitor is charged, in the polarity shown, to V_{os}. Under these conditions, the op-amp is removed from the inputs. When ϕ_2 is high and ϕ_1 is low, the op-amp functions normally, assuming the storage capacitance C is much larger than the input capacitance of the op-amp.

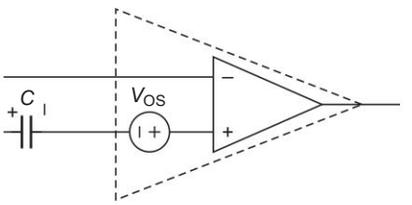


Fig. 5.20 Reduction of offset voltage with capacitor

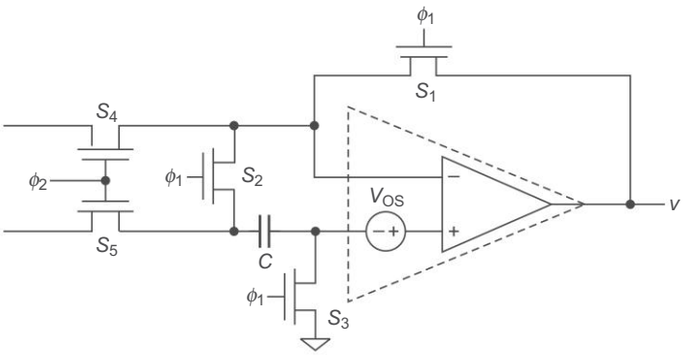


Fig. 5.21 Dynamic reduction of offset voltage

1. Dynamic Comparator

Figure 5.22 shows a dynamic comparator based on the inverter. When ϕ_1 is high, the voltage on the v_- input is connected to the node A, while the voltage on the node B is set via S_3 so that the input and output voltages of the inverter are equal. (The inverter is operating as a linear amplifier where both M_1

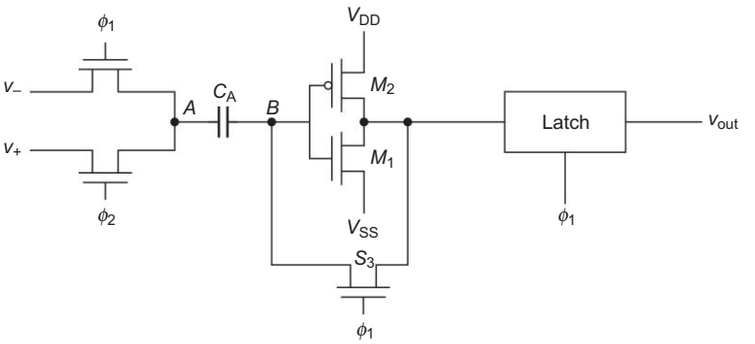


Fig. 5.22 Dynamic comparator

and M_2 are in the saturation regions.) When ϕ_1 become high and ϕ_1 is low due to no overlapping, the v_+ input is connected to node A. If C_A is much larger than the input capacitance of the inverter (C_B), then the voltage change on the input of the inverter (V_B) is

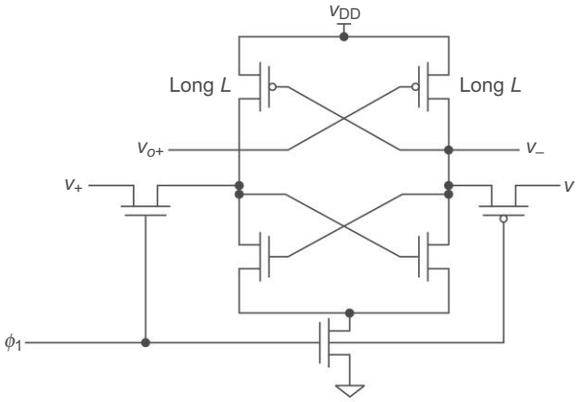


Fig. 5.23 Dynamic comparator based on CMOS latch

Figure 5.23 shows that dynamic comparator configuration is based on the dynamic CMOS latch. This latch is used as the positive feedback stage of the comparator. In the circuit, the offset-voltage of the comparator is reduced by using either input offset storage or Output Offset Storage (OOS) around the comparator preamp.

2. Dynamic Current Mirror

Figure 5.24 shows a dynamic current mirror circuit in which it is biased dynamically. When ϕ_1 is high, M_2 sinks current, and when ϕ_2 is high, M_1 sinks current. These circuits are useful in eliminating the mismatch effects, and thus the differences in the output currents, resulting from threshold voltage and transconductance parameter differences between devices. Since a single-reference current can be used to program the current in a string of current mirrors, only the finite output resistance of the mirrors will cause current differences $V_{in} = v_+ - v_-$.

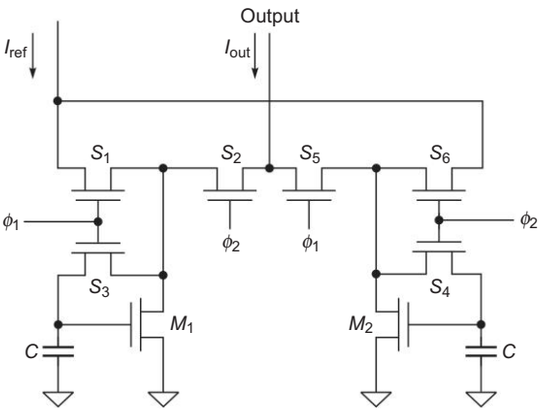


Fig. 5.24 Dynamic current mirror

3. Dynamic Amplifiers

Figure 5.25 shows a dynamic amplifier. The circuit amplifies when ϕ is low and dynamically biases M_1 and M_2, and it does not amplify, when ϕ is high. If C_1 and C_2 are large compared to the input capacitance of M_1 and M_2, then the input ac signal, v_{an} is applied to both gates. This biasing scheme makes the amplifier less sensitive to threshold and power supply variations. Other dynamic amplifier configurations exist, which have differential inputs and operate over both clock cycles.

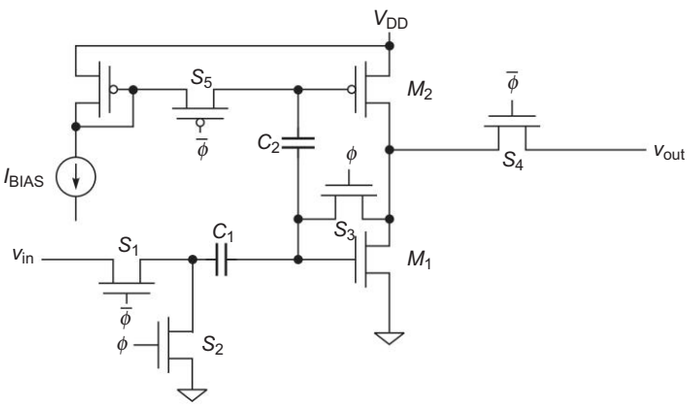


Fig. 5.25 Dynamic amplifier

5.6 ▫ DATA CONVERTER CIRCUITS

Data converters play an important role in the widespread electronics world. As the processing of signal can be done more accurately in the digital or discrete time domain, more sophisticated data converters must be required to translate analog to digital data and digital data to our inherent analog world as shown in Fig. 5.26. So there are two types of conversion—Analog-to-Digital Converter (ADC) which

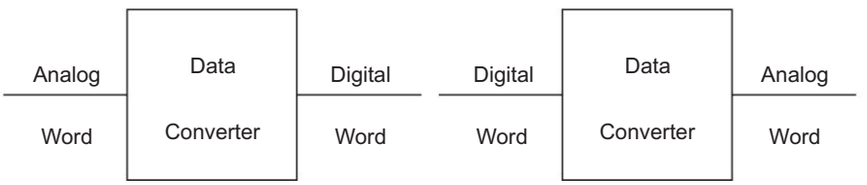


Fig. 5.26 ADC and DAC Converters

converts analog signals to discrete time or digital signal and Digital-to-Analog Converter (DAC) which makes the reverse operation. In order to discuss the functionality of these data converters, it is required to compare the characteristics of analog versus digital signals.

5.6.1 Analog versus Digital Signal

An analog signal is continuous and infinite valued, whereas the digital signal is discrete with respect to time and quantized amplitude. The term 'continuous-time signal' refers to a signal whose response with respect to time is uninterrupted. Simply stated, the signal has a continuous value for the entire segment of time for which the signal exists. The real-world physical quantities such as voltage, current, temperature, pressure, and time are in analog form. Although analog signals represent these quantities more accurately, it is difficult to process, store and transmit these analog form of signals.

The digital signal, on the other hand, is discrete with respect to time. This means that the signal is defined for only certain or discrete periods of time. A signal that is quantized can only have certain values (as opposed to an infinitely valued analog signal) for each discrete period. As mentioned earlier, it is convenient to represent these quantities in digital form for processing, transmission, and storage purpose. Figure 5.26 shows a typical ADC and DAC used between plant and processor for storage and processing of data.

5.6.2 Analog to Digital Converter (ADC)

We have already established the differences between analog and digital signals. In this section, we discuss how it is possible to convert from an analog signal to a digital signal. Figure 5.27 shows an ADC block which accepts input analog voltage v_{IN} and produces an output N-bit binary word D_0, D_1….$D_{N\text{-}1}$ of functional value

$$D = D_0 2^{-1} + D_1 2^{-2} + \cdots + D_{N-1}\, 2^{-N} \tag{5.22}$$

where D_0 = Most Significant Bit (MSB) and $d_{N\text{-}1}$ = Least Significant Bit (LSB).

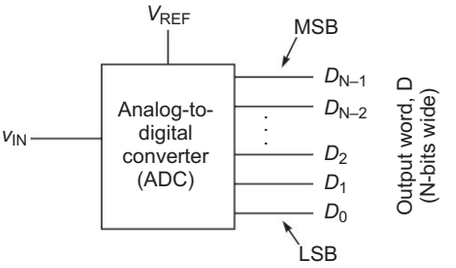


Fig. 5.27 Basic ADC block

A survey of ADC developments states that there are four different types of architectures: pipeline, flash-type, successive approximation, and oversampled ADCs. Each has benefits that are unique to that architecture and span the spectrum of high speed and resolution.

Since the ADC has a continuous, infinite-valued signal as its input, the important analog points on the transfer curve *x*-axis for an ADC are the ones that correspond to changes in the digital output world.

1. Flash-type ADC

Figure 5.28 shows a Flash-type ADC which utilizes one comparator per quantization level ($2^N - 1$) and 2^N resistors. Flash or parallel converters have the highest speed of any type of ADC. In the figure, the reference voltage is divided into 2^N values and each of divided reference value is fed into a comparator. The input voltage is compared with each reference voltage and represented in terms of a thermometer code at the output of the comparators. Table 5.1 shows the comparator output in terms of input analog voltage compared with divide reference voltage value V_d. A thermometer code will provide all zeros for each resistor level if the value of v_{IN} is less than the value on the resistor string, and ones, if v_{IN} is greater than or equal to voltage on the resistor. The $(2^N - 1) : N$ digital thermometer decoder circuit converts the compared data into an *N-bit* digital word. Each clock pulse generates an output digital word. The advantage of this converter is high speed but it has the doubling of area with each bit of increased resolution. Another disadvantage of the Flash ADC is power requirements of the $2^N - 1$ comparators. The speed is limited by the switching of the comparators and the digital logic.

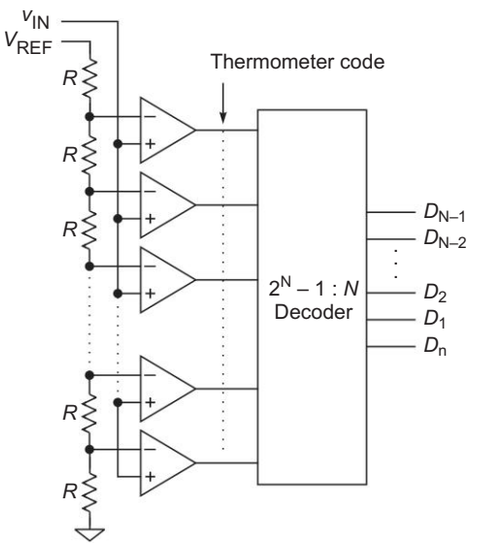


Table 5.1 Comparator input/output

Input voltage	Comparator output
$V_{IN} > V_d$	$X > 1$
$V_{IN} > V_d$	$X = 0$
$V_{IN} = V_d$	Previous

Fig. 5.28 Flash-type ADC

As for an example, Fig. 5.29 shows a 3-bit flash type ADC consisting of a resistive divider network, 8 op-amp comparators and an 8-line to 3-line encoder (3-bit priority decoder). Table 5.2 shows the truth table for the same.

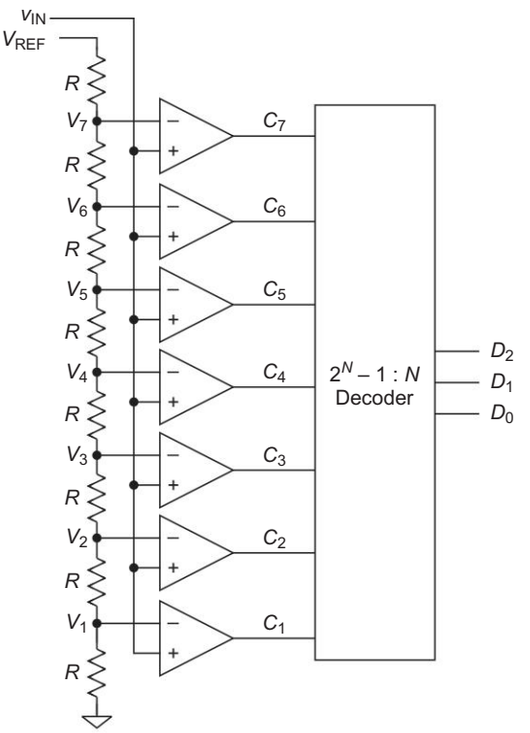


Fig. 5.29 3-bit flash type ADC

Table 5.2 Truth table of flash-type ADC

Input voltage (v_{IN})	$C_7\ C_6\ C_5\ C_4\ C_3\ C_2\ C_1\ C_0\ D_2\ D_1\ D_0$
0 to $V_{REF}/8$	0 0 0 0 0 0 0 1 0 0 0
$V_{REF}/8$ to $2V_{REF}/8$	0 0 0 0 0 0 1 1 0 0 1
$2V_{REF}/8$ to $3V_{REF}/8$	0 0 0 0 0 1 1 1 0 1 0
$3V_{REF}/8$ to $4V_{REF}/8$	0 0 0 0 1 1 1 1 0 1 1
$4V_{REF}/8$ to $5V_{REF}/8$	0 0 0 1 1 1 1 1 1 0 0
$5V_{REF}/8$ to $6V_{REF}/8$	0 0 1 1 1 1 1 1 1 0 1
$6V_{REF}/8$ to $7V_{REF}/8$	0 1 1 1 1 1 1 1 1 1 0
$7V_{REF}/8$ to $8V_{REF}/8$	1 1 1 1 1 1 1 1 1 1 1

Accuracy Analysis for the Flash ADC Accuracy is dependent on the matching of the resistor string and the input offset voltage of the comparators. The voltage at the i^{th} tap is found to be

$$V_i = V_i^{ideal} + \frac{V_{REF}}{2^N}\sum_{k=1}^{i}\frac{\Delta R_k}{R} \tag{5.23}$$

where $V_i^{ideal} = i\frac{V_{REF}}{2^N} = i^{th}$ tap ideal voltage and the term ΔR_e = error in resistance

where = i^{th} tap ideal voltage and the term ΔR_k = Resistance error.

The Integral Non Linearity (INL) is defined as the difference between the actual and ideal switching points. The worst case INL can be written as

$$\text{INL} = V_{SW,i} - V_i^{ideal} = \frac{V_{REF}}{2^N}\sum_{k=1}^{i}\frac{\Delta R_k}{R} + V_{os,i} \tag{5.24}$$

where $V_{Sw,i} = V_i + V_{os,i}$ = Switching voltage of the i^{th} comparator and $V_{os,i}$ = input referred offset voltage of i^{th} comparator.

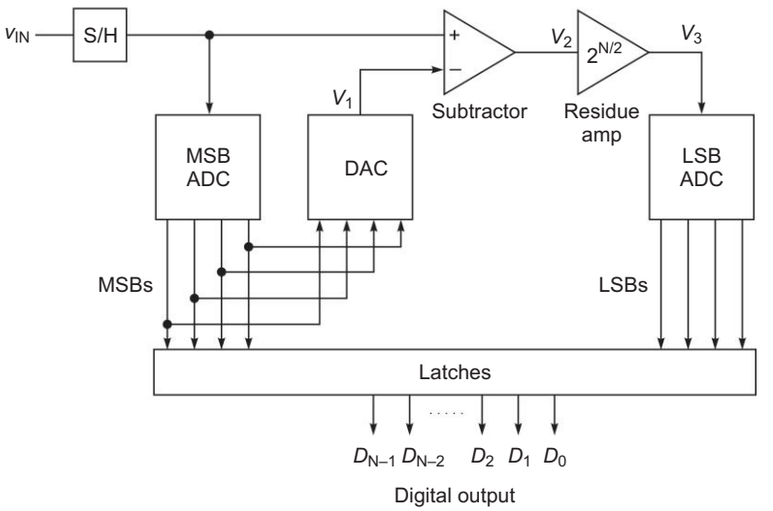


Fig. 5.30 Two-step Flash type

Two-Step Flash ADC Figure 5.30 shows the block diagram of a two-step Flash converter or a parallel, feed-forward ADC. The converter is separated into two complete Flash ADCs with feed-forward circuitry. The first converter generates a rough estimate of the value of the input, and the second converter performs a fine conversion. The advantages of this architecture are that the number of comparators is greatly reduced from that of the Flash converter—from $2^N - 1$ comparators to $2(2^{N/2} - 1)$ comparators. The conversion process is as follows:

(a) After the input is sampled, the most significant bits (MSBs) are converted by the first Flash ADC.
(b) The result is then converted back to an analog voltage with the DAC and subtracted with the original input.
(c) The result of the subtraction, known as the residue, is then multiplied by 2^m and input into the second ADC. The multiplication not only allows the two ADCs to be identical, but also increases the quantum level of the signal input into the second ADC.
(d) The second ADC produces the least significant bits through a Flash conversion.

Some architectures use the same set of comparators in order to perform both steps. The multiplication mentioned in Step (c) can be eliminated if the second converter is designed to handle very small input signals.

2. The Pipeline ADC

The pipeline ADC is an *N*-step converter, with 1-bit being converted per stage. Figure 5.31 shows a pipeline ADC consisting of *N* stages connected in series to achieve high resolution (10–13 bits) at relatively fast speeds. Each stage has a 1-bit ADC (a comparator), a sample-and-hold, a summer, and a gain of two amplifiers. Each stage of the converter performs the following operation:

(a) After the input signal has been sampled, it is compared to $\frac{V_{REF}}{2}$. The output of each comparator is the bit conversion for that stage.
(b) If $V_{IN} > \frac{V_{REF}}{2}$, comparator output is 1 and $\frac{V_{REF}}{2}$ is subtracted from the held signal and the result is passed to the amplifier. If $V_{IN} < \frac{V_{REF}}{2}$, the comparator output is o and the original input signal is passed to the amplifier. The output of each stage in the converter is referred as the residue.
(c) The result of the summation is multiplied by 2 and the result is passed to the sample and hold of the next stage.

The main advantage of the pipeline converter is its high throughput. After an initial delay of *N* clock cycles, one conversion will be completed per clock cycle. While the residue of the first stage is being operated on by the second stage, the first stage is free to operate on the next samples. Each stage operates on the residue passed down from the previous stage, thereby allowing for fast conversions. A slight error in the first stage propagates through the converter and results in a much larger error at the end of the conversion. Each succeeding stage requires less accuracy than the one before, so special care must be taken when considering the first several stages.

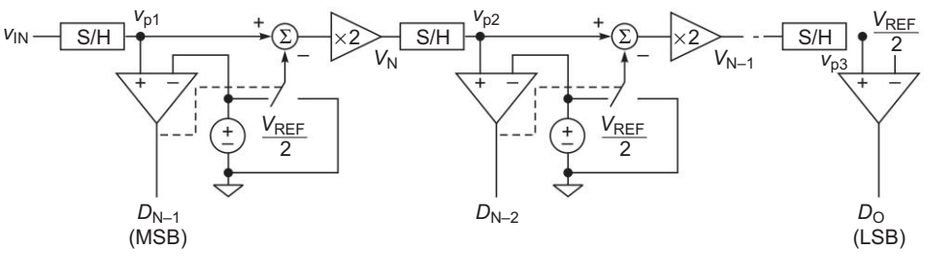


Fig. 5.31 Pipeline-type ADC

Accuracy in the Pipeline Converter The 1-bit per stage ADC can be analyzed by examining the switching point of each comparator for the ideal and non-ideal case. Since the comparators in different stages are pipelined, the error in the comparator in each stage is propagated to next stage. The integral nonlinearity (INL) is defined as the difference between the actual and ideal switching points. The worst-case INL after N^{th} stage can be written as

$$INL_{\mathrm{N}} = \frac{1}{2} D_{\mathrm{N-2}} . V_{\mathrm{REF}} . \left(\frac{1}{A} - \frac{1}{2} \right) + \frac{1}{2} D_{\mathrm{N-3}} . V_{\mathrm{REF}} . \left(\frac{1}{A^2} - \frac{1}{4} \right) + ... \frac{1}{2} . V_{\mathrm{REF}} . \left(\frac{1}{A^{N-1}} - \frac{1}{2^{N-1}} \right) + \frac{V_{\mathrm{cos,N}}}{A^{N-1}} - \sum_{k=1}^{N} \frac{V_{\mathrm{SOS,k}}}{A^{k-1}} \quad (5.25)$$

where $V_{\mathrm{cos,N}} = N^{th}$ comparator offset voltage, $V_{\mathrm{SOS,k}} = k^{th}$ sample-and-hold offset voltage comarator, and $V_{\mathrm{os,\,i}}$ = Input referred offset voltage of i^{th} comparator. $D_{\mathrm{N-1}}$, $D_{\mathrm{N-2}}$, ...$D_1 . D_0$ are output bits in 1^{st}, 2^{nd} …… stages respectively. A is gain of the residue amplifier.

3. Integrating ADC

Another type of ADC performs the conversion by integrating the input signal and correlating the integration time with a digital counter. There are two types of integrating ADC—single-slope and dual-slope architecture. These types of converters have high resolution but have relatively slow conversions. However, they are not very costly and are used in slow-speed, cost-conscious applications.

(a) Single-Slope Architecture Figure 5.32(a) shows the block diagram of a single-slope converter in which a counter determines the number of clock pulses that are required before the integrated value of a reference voltage is equal to the sampled input signal. The number of clock pulses is proportional to the actual value of the input, and the output of the counter is the actual digital representation of the analog voltage. The output of the integrator should start at zero and linearly increase with a slope that is dependent on the gain of the integrator as shown in Fig. 5.32(b). The reference voltage is negative because the output of the inverting integrator should be positive. When the output of the integrator surpasses the value of the S/H output, the comparator switches states, thus triggering the control logic to latch the value of the counter. The control logic also resets the system for the next sample. The conversion time, t_c, is dependent on the value of the input signal and can be described as

$$t_{\mathrm{c}} = \frac{v_{\mathrm{IN}}}{V_{\mathrm{REF}}} 2^N . T_{\mathrm{CLK}} \quad (5.26)$$

where T_{CLK} is the period of the clock. The sampling rate is inversely proportional to the conversion time and can be written as

$$f_{\mathrm{s}} = \frac{V_{\mathrm{REF}}}{V_{\mathrm{IN}} . 2^N} . f_{\mathrm{CLK}} \quad (5.27)$$

(b) Accuracy Issues Related to the Single-Slope ADC In this architecture, many errors mainly come from integrating circuit. At the end of the conversion, the voltage across the integrating capacitor, V_c, (assuming no initial condition), will be

$$V_{\mathrm{c}} = \frac{1}{CR} \int_0^{t_{\mathrm{c}}} . V_{\mathrm{REF}} dt = \frac{V_{\mathrm{REF}}}{RC} t_{\mathrm{c}} \quad (5.28)$$

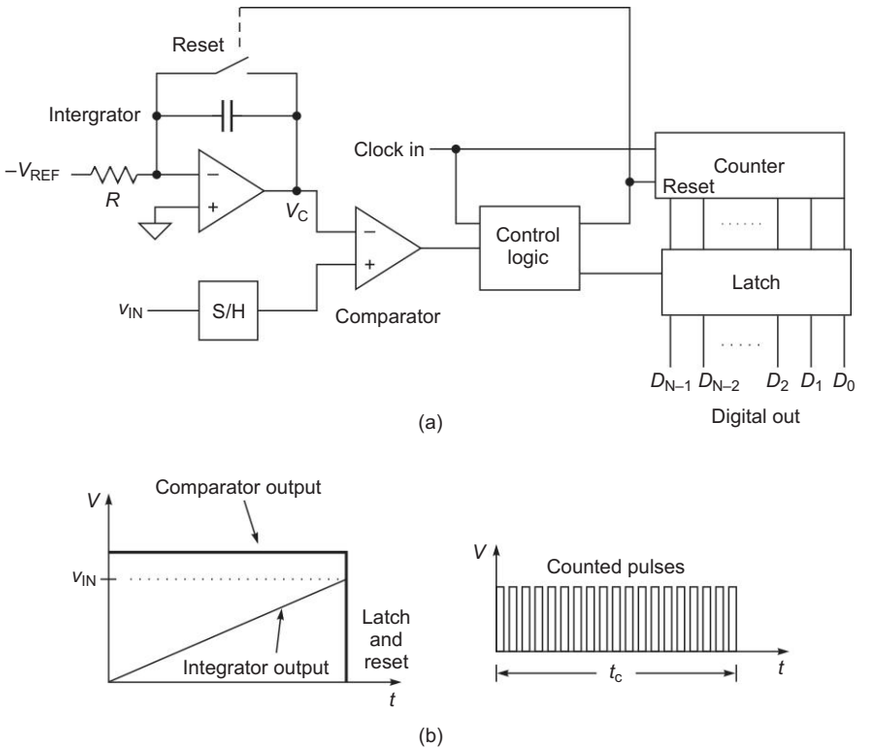


Fig. 5.32 Single-slope ADC: (a) Block diagram (b) Integration output and counter output

Apart from capacitance in integrator, resistance can limit the accuracy, since the resistor will be effectively nonlinear. The reference voltage must also stay constant within the accuracy of the converter.

(c) Dual-Slope Architecture Figure 5.33(a) shows a slightly more sophisticated dual-slope integrating ADC in which two integrations are performed—one on the input signal and one on V_{REF}. The input voltage in this case should be negative, so that the output of the inverting integrator results in a positive slope during the first integration. Figure 5.33(b) shows dual slope of integration—first slope time period is constant and second slope time period is variable. The first integration is of fixed length, determined by the counter, in which the sample-and-held signal is integrated, resulting in the first slope. After the counter overflows and is reset, the reference voltage is connected to the input of the integrator. Since v_{IN} was negative and the reference voltage is positive, the inverting integrator output will begin discharging back down to zero at a constant slope. A counter again measures the amount of time for the integrator to discharge, thus generating the digital output. In this ADC, the first slope varies according to the value of the input signal, while the second slope, dependent only on V_{REF}, is constant. Similarly, the time required to generate the first slope is constant, since it is limited by the size of the counter. However, the discharging period is variable and results in the digital representation of the input voltage.

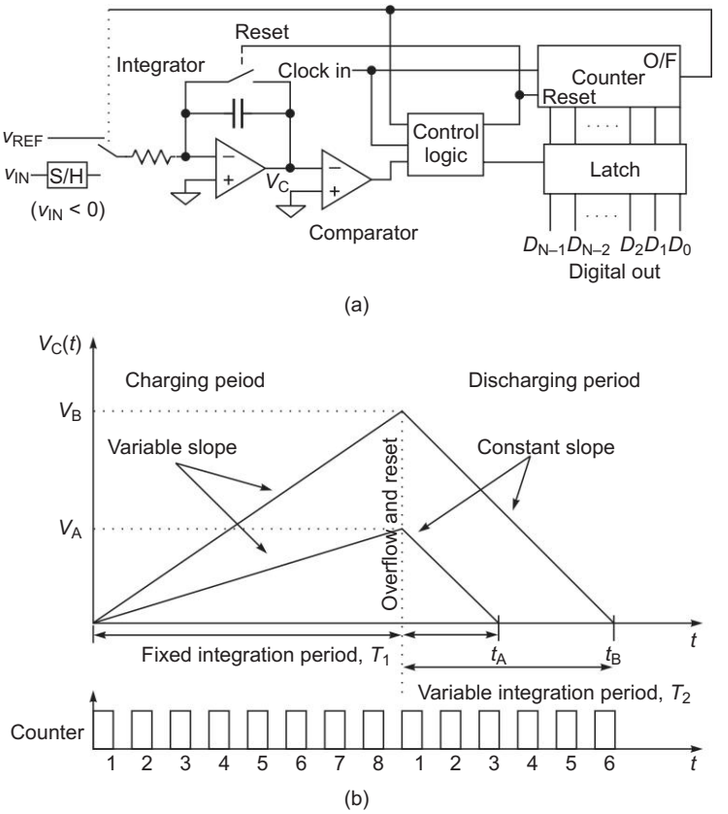


Fig. 5.33 Dual slope ADC: (a) Block diagram (b) Integration output and counter output

(d) Accuracy Issues Related to the Dual-Slope ADC The dual-slope converter is an improvement over the single-slope architecture because of a significantly longer conversion time. The first integration period requires a full 2^N clock cycle and cannot be decreased, because the second integration might require the full 2^N clock cycles to discharge if the maximum value of v_m is being converted. However, the dual slope is the preferred architecture because the same integrator and clock are used to produce both slopes. Therefore, any non-idealities will essentially be canceled. The output at the end of *T,* is positive since the input voltage is considered to be negative and the integrator is inverting. The V_C can be written as

$$V_c = \frac{1}{CR}\int_0^{T_1} |V_{IN}|.dt - \frac{1}{RC}\int_0^{T_2} V_{REF}\,dt \tag{5.29}$$

$$= \frac{1}{RC}\left[|V_{IN}|.T_1 - V_{REF}.T_2\right] \tag{5.30}$$

If the nonlinearity of integration is there in the integration circuit, it will be cancelled. For full cancellation,

$$|V_{IN}|.T_1 = V_{REF}.T_2 \tag{5.31}$$

4. Successive Approximation ADC

Figure 5.34 shows a successive approximation converter which performs a binary search through all possible quantization levels before converging on the final digital output. An *N*-bit register controls the timing of the conversion where *N* is the resolution of the ADC. V_{IN} is sampled and compared to the output of the DAC. The comparator output controls the direction of the binary search, and the output of the Successive Approximation Register (SAR) is the actual digital conversion. The steps of successive approximation are as follows.

(a) High logic level is applied to the input to the shift register. For each bit converted, the high is shifted to the right 1-bit position. $B_{N-1} = 1$ and B_{N-2} through $B_0 = 0$.

(b) The MSB of the SAR, D_{N-1} is initially set to 1, while the remaining bits, D_{N-2} through D_0, are set to 0.

(c) Since the SAR output controls the DAC and the SAR output is 100...0, the DAC output is set to $\frac{V_{REF}}{2}$.

(d) Next, v_m is compared to $\frac{V_{REF}}{2}$. If $\frac{V_{REF}}{2}$ is greater than v_{IN} then the comparator output is a 1 and the comparator resets D_{NA} to 0. If $\frac{V_{REF}}{2}$ is less than v_{IN}, then the comparator output is a 0 and the D_{N-1} remains a 1. D_{N-1} is the actual MSB of the final digital output.

(e) The 1 applied to the shift register is then shifted by one position so that $B_{N-2} = 1$, while the remaining bits are all 0.

(f) D_{N-2} is set to a 1, D_{N-3} through D_0 remains 0, while D_{N-1} remains the value from the MSB conversion. The output of the DAC will now be either equal to $\frac{V_{REF}}{2}$ (if $D_{N-1} = 0$) or $\frac{3V_{REF}}{4}$ (if $D_{N-1} = 1$).

(g) Next, v_{IN} is compared to the output of the DAC. If the DAC output is greater than v_{in} of the comparator, the D_{N-2} is reset to 0. If v_{IN} is less than the DAC output, D_{N-2} remains a 1.

(h) The process repeats until the output of the DAC converges to the value of v_{IN} within the resolution of the converter.

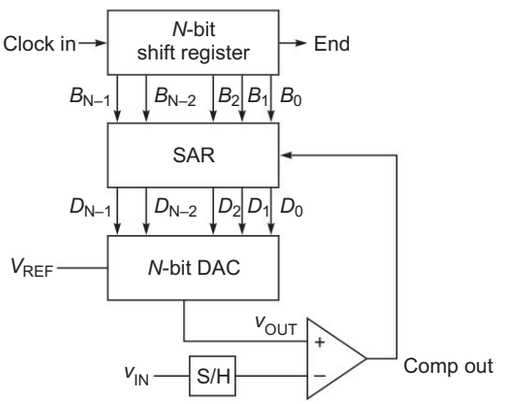


Fig. 5.34 Block diagram of the successive approximation ADC

In an example of the 8-bit successive approximation ADC converter in which initially MSB value of SAR is set to 1, the code becomes 10000000. The output of DAC is compared with sampled input analog voltage and if input voltage is greater than DAC output then 10000000 is less than corrected representation. It is repeated till SAR, DAC output is equal to input signal and after getting the same, it becomes the end of operation.

Figure 5.35 shows charge-redistribution successive-approximation ADC in which the binary-weighted capacitor array is used as its DAC. The binary-weighted capacitor array of the converter samples the input signal and then performs the binary search based on the amount of charge on each of the DAC capacitors. The comparator is replaced by a unity-gain amplifier.

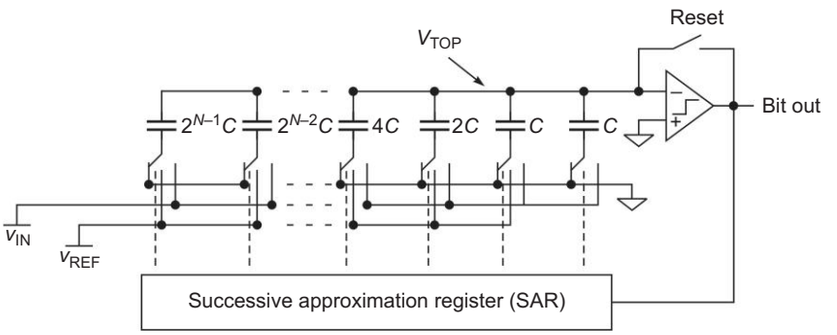


Fig. 5.35 Charge redistribution successive approximation ADC

The simplicity of the design allows for both high speed and high resolution while maintaining relatively small area. The limit to the ADC's accuracy is dependent mainly on the accuracy of the DAC. If the DAC does not produce the correct analog voltage with which to compare the input voltage, the entire converter output will contain an error.

The conversion process begins by discharging capacitor array, via the reset switch. Although this may appear to be an insignificant action, the converter is also performing automatic offset cancellation. Once the reset switch is closed, the comparator acts as a unity-gain buffer. Thus, the capacitor array charges to the offset voltage of the comparator. This requires that the comparator be designed so as to be unity gain stable, which means that internal compensation may have to be switched in during the reset period. The input voltage, v_{IN} is sampled onto the capacitor array. The equivalent circuit is seen in Fig. 5.36. The conversion process begins by switching the bottom plate of the MSB capacitor to V_{REF} (Fig. 5.36c). If the output of the comparator is high, the bottom plate of the MSB capacitor remains connected to V_{REF}. If the comparator output is low, the bottom plate of the MSB is connected back to ground. The output of the comparator is D_{N-1}. The voltage at the top of the capacitor array, V_{T0P}, is written as

$$V_{Top} = -v_{IN} + V_{os} + D_{N-1} \cdot \frac{V_{REF}}{2} \tag{5.32}$$

The next largest capacitor is tested in the same manner as seen in Fig. 5.36(d). The voltage at the top plate of the capacitor after the second capacitor becomes

$$V_{\text{Top}} = -v_{\text{IN}} + V_{\text{os}} + D_{\text{N}-1} \cdot \frac{V_{\text{REF}}}{2} + D_{\text{N}-2} \cdot \frac{V_{\text{REF}}}{4} \tag{5.33}$$

The conversion process continues on with the remaining capacitors so that the voltage on the top plate of the array, V_{T0P}, converges to the value of the offset voltage, V_{os} (within the resolution of the converter).

$$V_{\text{T0P}} = v_{\text{IN}} + V_{\text{os}} + D_{\text{N}-1} \cdot \frac{V_{\text{REF}}}{2} + D_{\text{N}-2} \cdot \frac{V_{\text{REF}}}{4} + \ldots\ldots + D_0 \cdot \frac{V_{\text{REF}}}{2^{\text{N}-1}} \approx V_{\text{os}} \tag{5.34}$$

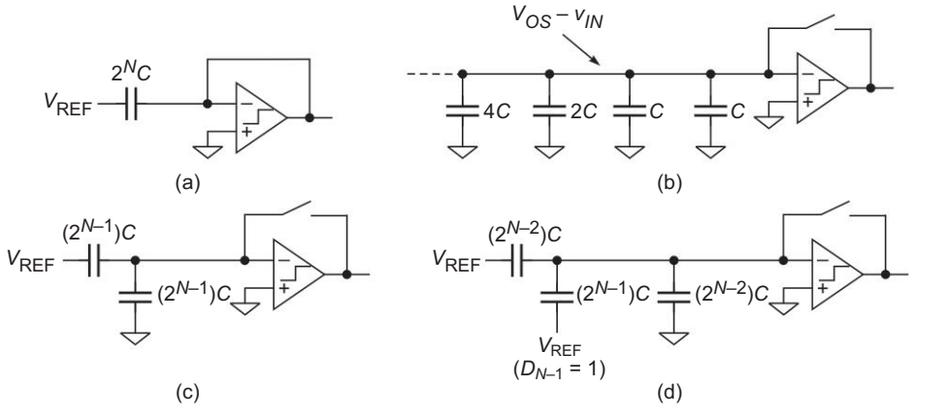


Fig. 5.36 The charge redistribution process: (a) Sampling the input while autozeroing the offset (b) Voltage at the top plate after sampling (c) Equivalent circuit while converting the MSB (d) Equivalent circuit while converting the next largest capacitor with the MSB result equal to one.

Accuracy of the Charge Redistribution Successive Approximation ADC Accuracy of this architecture is limited due to the capacitor mismatching. The mismatch is analyzed in the same manner as the binary-weighted current source array. Integrated nonlinearity for capacitance mismatch in current source I can be written as

$$\left|INL\right|_{\max} = 2^{N-1}(C + \left|\Delta C\right|_{\max}) - 2^{N-1}C = 2^{N-1}\left|\Delta C\right|_{\max} \tag{5.35}$$

5.6.3 DAC Architectures

Figure 5.37 shows digital-to-analog converter in which input is D_0, D_1……$D_{\text{N-2}}$, $D_{\text{N-1}}$ and compared with V_{REF} to give analog output v_{OUT} which is written as

$$v_{\text{OUT}} = KV_{\text{FS}}(D_0 2^{-1} + D_1 2^{-2} + \ldots\ldots\ldots + D_{\text{N}-1} 2^{N}) \tag{5.36}$$

Where V_{FS} = Full-scale output voltage, K = Scaling factor, D_0 = Most significant bit and $D_{\text{N}-1}$ = least significant bit. There are different types of DAC—weighted resistor DAC, register-string based DAC, R-2R ladder based DAC, and charge scaling DAC. Each, of course, has its own merits. Some use voltage division, whereas others employ current steering and even charge scaling to map the digital value into an analog quantity.

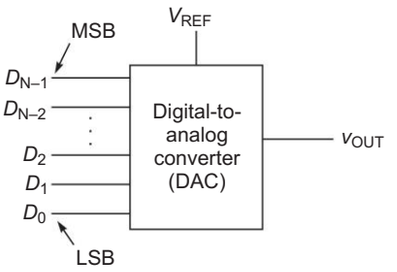


Fig. 5.37 Basic DAC block

1. Weighted Register DAC

Figure 5.38 shows weighted register DAC consisting of a summing amplifier with binary resistor network consisting of input resistances 2^1R, 2^2R, 2^3R……$2^{-N}R$ and feedback resistance R_f. It has N single-pole double-throw type electronic switches D_0, D_1…….D_{N-1} controlled by binary input word. In the figure, the output current is given by

$$I_{out} = I_0 + I_1 + \ldots\ldots\ldots + I_{N-1}$$

$$= \frac{V_R}{R}(D_0 2^{-1} + D_1 2^{-2} + \ldots\ldots\ldots + D_{N-1} 2^{-N})$$

$$v_{OUT} = I_{out} R_f = V_R \frac{R_f}{R}(D_0 2^{-1} + D_1 2^{-2} + \ldots\ldots\ldots + D_{N-1} 2^{N}) \quad (5.37)$$

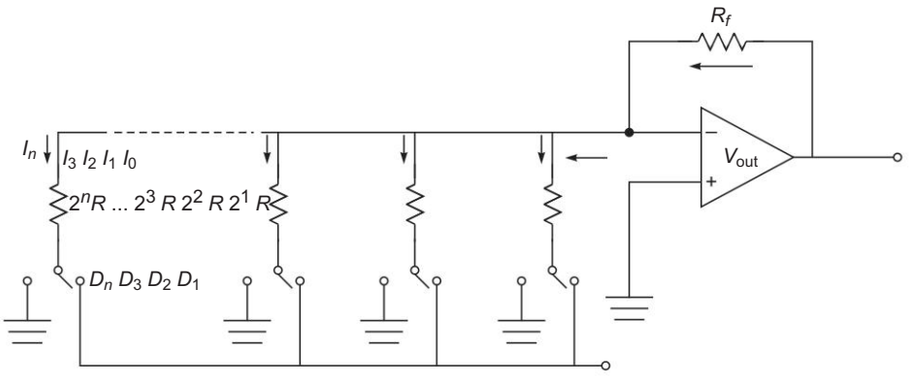


Fig. 5.38 Binary-weighted register DAC

2. Resistor String

Figure 5.39 shows basic DAC consisting of a simple resistor string of 2^N identical resistors\and an array of $2^N - 1$ switches. The analog output is simply the voltage division of the resistors at the selected tap. But an N:2^N decoder will be required to provide the 2^N signals controlling the switches. This archi-

tecture typically has good accuracy because no output current is required provided that the values of the resistors are within the specified error tolerance of the converter. One big advantage of a resistor string is that the output will always be guaranteed to be monotonic.

Here, a binary switch array ensures that the output is connected to at most N switches that are on and N switches that are off, thus increasing the conversion speed. The input to this switch array is a binary word since the decoding is inherent in the binary-tree arrangement of the switches.

The problem with the resistor string is that an integrated form of this converter occupies a large chip area for higher bit resolutions because of the large number of passive components needed. Active resistors such as the N-well resistor can be used for low-resolution applications. However, as the resolution increases, the relative accuracy of the resistors becomes an important factor. Although the value of R could always be made small to minimize the chip area required, power dissipation would then become the critical issue as current flows through the resistor string at all times.

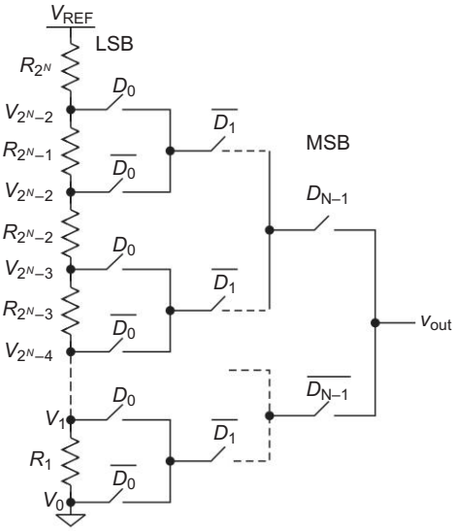


Fig. 5.39 Register-string based DAC using binary switch array

The value of the output analog voltage at the TPA associated with i^{th} resistance is written as

$$V_{out}(i) = \frac{(i)\, V_{REF}}{2^N}$$

where, $[i = 0, 1, 2\ldots\ldots 2^N - 1]$.

Mismatch Errors Related to the Resistor String DAC The accuracy of the resistor string is obviously related to matching between the resistors, which ultimately determines the INL and DNL for the entire DAC. We consider that the i^{th} resistor has a mismatch error associated with it so that

$$Ri = R + \Delta R_t \tag{5.38}$$

where R is the ideal value of the resistor and ΔR_t is the mismatch error. Due to the mismatch in resistance, the actual value of the i^{th} voltage will be the sum of all the resistances up to and including resistor i, divided by the sum of all the resistances in the string. This can be represented by

$$V_{\text{out}}^{\text{mis}}(i) = V_{\text{REF}} \frac{\sum_{\text{k}=1}^{i} (R + \Delta R_{\text{k}})}{2^{\text{N}} R} = \frac{(i) V_{\text{REF}}}{2^{\text{N}}} + \frac{V_{\text{REF}}}{2^{\text{N}}} \sum_{k=1}^{i} \frac{\Delta R_{\text{k}}}{R} \tag{5.39}$$

Integral nonlinearity (INL) is defined as the difference between the actual and ideal switching points.

$$INL = \frac{V_{\text{REF}}}{2^{\text{N}}} \sum_{k=1}^{i} \frac{\Delta R_{\text{k}}}{R} \tag{5.40}$$

Resistor-string matching is not as critical when determining the DNL. The definition of DNL is simply the actual height of the stair-step in the DAC transfer curve minus the ideal step height. So we can write this in terms of the voltages at the taps of adjacent resistors on the string.

$$DNL_{\text{i}} = \left| V_{\text{i}} - V_{\text{i}-1} \right| = \frac{V_{\text{REF}}}{2^{N}} \frac{\Delta R_{\text{i}}}{R} \tag{5.41}$$

3. R-2R Ladder Networks

We know that a wide range of registers are required for both binary-weighted register DAC and register-string DAC architecture. To avoid these registers, it is required to have a DAC that incorporates fewer resistors. Figure 5.40 shows an *N*-bit *R*-2*R* ladder network DAC that has fewer resistances. This configuration consists of a network of resistors alternating in value of *R* and 2*R.* In the figure, starting at the right end of the network, it is seen that the resistance looking to the right of any node to ground is 2*R.* The digital input determines whether each resistor is switched to ground (non-inverting input) or to the inverting input of the op-amp. Each node voltage is related to V_{REF}, by a binary-weighted relationship caused by the voltage division of the ladder network. The total current flowing from V_{REF} is constant, since the potential at the bottom of each switched resistor is always zero volts (either ground or virtual ground). Therefore, the node voltages will remain constant for any value of the digital input.

The output voltage, v_{out}, dependent on currents flowing through the feedback resistor, R_{F} is written as

$$v_{\text{OUT}} = -i_{\text{tot}} . R_{\text{F}} = \sum_{k=1}^{N-1} D_{\text{k}} \frac{V_{\text{REF}}}{2^{\text{N}-\text{k}}} \frac{R_{\text{F}}}{2R} \tag{5.42}$$

where i_{tot} is the sum of the currents selected by the digital input and D_{k} is the k^{th} bit of the input word with a value that is either a 1 or a 0.

Mismatch Error This architecture, like the resistor-string architecture, requires matching within the resolution of the converter. Therefore, the switch resistance must be negligible, or a small voltage drop will occur across each switch, resulting in an error. One way to eliminate this problem is to add dummy switches. The total resistance of any horizontal branch, $R^{/}$ is

$$R^{/} = R + \Delta R/2 \tag{5.43}$$

The resistance of any vertical branch is $2R + \Delta R$, which is twice the value of the horizontal branch. To avoid this mismatch in resistances, a $R^{/} — 2R^{/}$ relationship is also maintained.

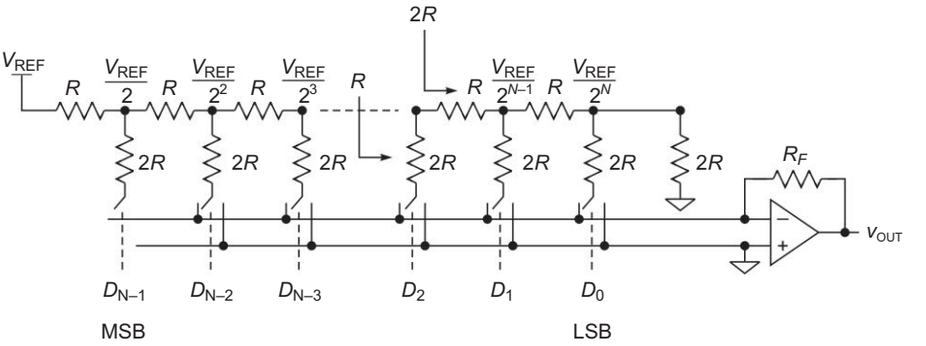


Fig. 5.40 *R*-2*R* ladder networks

4. Charge Scaling DACs

Figure 5.41(a) shows a charge scaling or charge-distributed DAC consisting of a parallel array of binary-weighted capacitors, totaling $2^N C$ connected to an op-amp. After initially being discharged, the digital signal switches each capacitor to either V_{REF} or ground, causing the output voltage, v_{out}, to be a function of the voltage division between the capacitors. Since the capacitor array totals $2^N C$, if the MSB is high and the remaining bits are low then a voltage divider occurs between the MSB capacitor and the rest of the array. The analog output voltage, v_{out}, becomes

$$v_{out} = V_{REF} \cdot \frac{2^{N-1}C}{2^{N-1}C + 2^{N-2}C + 2^{N-3}C + ... + 2C + C. + C} = \frac{V_{REF}}{2} \tag{5.44}$$

It is confirmed from the fact that the MSB changes the output of a DAC by $\frac{V_{REF}}{2}$. Figure 5.41(b) shows the equivalent circuit under this condition. If it is assumed that the k^{th} bit, D_k, is 1 and all other bits are zero, the ratio between v_{out} and V_{REF} due to each capacitor can be written in general form for k^{th} node as

$$v_{out}(k) = \frac{2^k}{2^N} V_{REF} = \frac{V_{REF}}{2^{N-k}} \tag{5.45}$$

From the superposition value of v_{out} for digital input word, D_0, D_1,……D_k …..D_{N-1} can be written as

$$v_{out} = \sum_{k=0}^{N-1} D_k \frac{V_{REF}}{2^{N-k}} \tag{5.46}$$

One limitation of this architecture is the existence of a parasitic capacitance at the top plate of the capacitor array due to the op-amp.

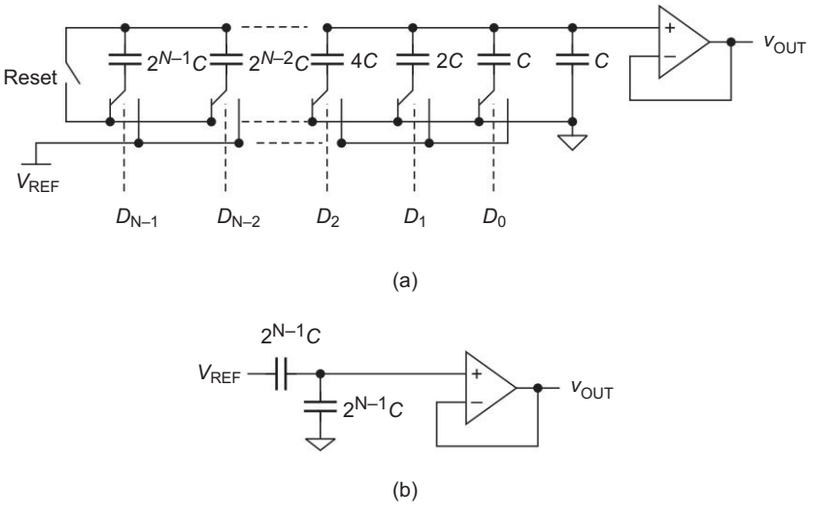


Fig. 5.41 Charge scaling DAC: (a) Block diagram (b) Equivalent circuit

5. Pipeline DAC

Figure 5.42 shows pipeline DAC consisting of an N stages cyclic converter where each stage performs one bit of the conversion. Here, the signal is passed down the "pipeline," and as each stage works on one conversion, the previous stage can begin processing another. Therefore, an initial N clock cycle delay is experienced as the signal makes its way down the pipeline the very first time. However, after the N clock cycle delays, a conversion takes place at every clock cycle. Besides the N clock cycle delay, this architecture can be very fast. However, the amplifier gains must be very accurate to produce high resolutions. The output voltage of the n^{th} stage in the converter can be written as

$$v_{out}(k) = [D_{k-1}.V_{REF} + v_{OUT}(k-1)]\frac{1}{2} \tag{5.47}$$

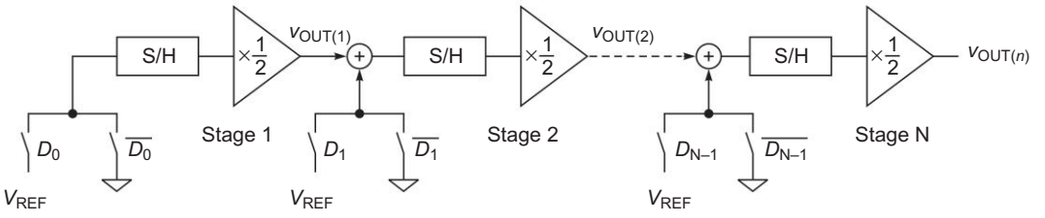


Fig. 5.42 Pipeline DAC using cyclic converter

5.7 □ BIT SYNCHRONIZATION/DATA RECOVERY CIRCUIT

A data-recovery circuit is a mixed signal circuit which performs an important task of bit synchronization in high-speed communication. The circuit uses either a Phase-Locked Loop (PLL) or Delay-Locked Loop (DLL). In a PLL, a clock signal is generated to lock or synchronize with incoming signal

whereas in a DLL, the input data is delayed through a Voltage Variable Delay Line (VVDL) until it is synchronized with the clock signal which is available at the correct frequency. Since in a DLL, no clock signal synthesis is required, it offers better stability and faster lock speed than that of PLL.

5.7.1 Phase-Locked Loop-Based Data Recovery Circuit

Phase-locked loop circuit is used as a bit synchronization or data/clock recovery circuit in communication system. It performs the function of generating a clocked signal which is locked or in synchronization with the incoming signal. The generating clocked signal is used in the receiver to clock the shift register and to recover data. Figure 5.43 shows the basic block diagram of phase-locked loop (PLL) consisting of Phase Detector (PD), loop filter, Voltage-Controlled Oscillator (VCO), and divide by N counter. The PD [shown in Fig. 5.45] generates an output signal proportional to the time difference between the data in and the divided-down clock (dclock). This signal is filtered by a loop filter and filtered output is connected to the input of VCO which generates a synchronized clock out.

The PD is normally XOR which gives voltage output V_{PDout} as follows:

$$V_{PDout} = V_{DD}\frac{\Delta\phi}{\pi} = K_{PD}.\Delta\phi \tag{5.48}$$

where $\Delta\phi = \phi_{data} - \phi_{dclock} = 2\pi\Delta t / T_{dcLK}$. Figure 5.45(a) shows the PD with loop filter.

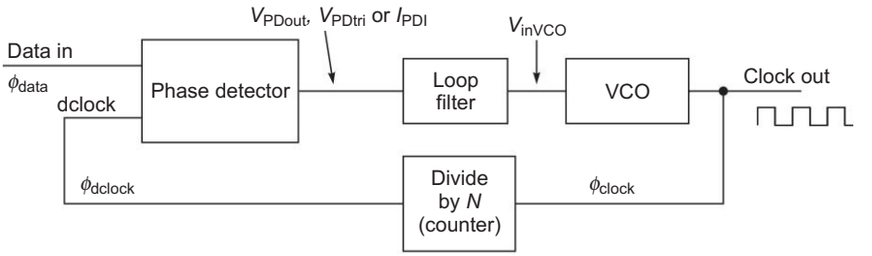


Fig. 5.43 Phase-locked loop

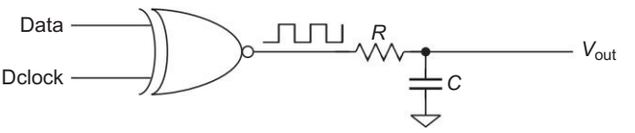


Fig. 5.44 PD with loop filter

Figure 5.44 shows phase detector (PD) with loop filter whereas Fig. 5.45 shows voltage controlled oscillator based CMOS. In Fig. 5.45, MOSFETS M_5 and M_6 behave as constant-current sources sinking a current $I_{D.}$ whereas M_1 and M_2 operate as switches. If M_1 is off and M_2 is on, the drain of M_1 is pulled to V_{DD}-V_{THN} by M_3 and is held at the same voltage till M_1 turns on and M_2 is off. This process goes on alternatively with oscillating frequency of VCO. Figure 5.46 shows PLL using XOR detector. The phase transfer function can be written as

$$H(s) = \frac{\phi_{\text{clock}}}{\phi_{\text{dclock}}} = \frac{K_{\text{PD}}K_{\text{F}}.K_{\text{VCO}}}{s + \beta K_{\text{PD}}K_{\text{F}}.K_{\text{VCO}}} \tag{5.49}$$

where K_{PD} = Gain of phase detector, K_{VCO} = VCO gain and K_{F} = Gain of filter and $S = jw$.

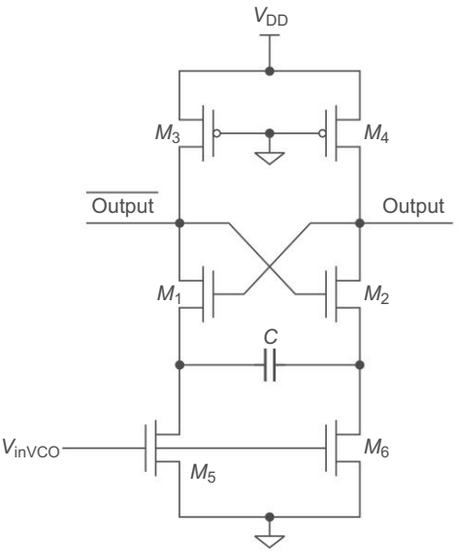


Fig. 5.45 Source-coupled VCO

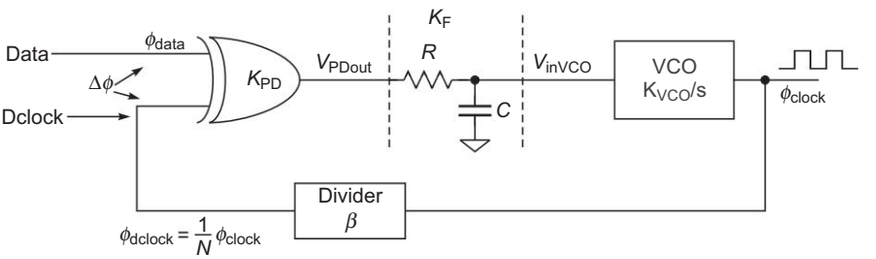


Fig. 5.46 PLL with XOR

5.7.2 Delay-Locked Loop-Based Data Recovery Circuit

Figure 5.47 shows the block diagram of a data recovery circuit consisting of a DLL circuit, clock multiplier, SAW filter, frequency divider, and sample/hold circuit. The synchronized clock (Syn Clk) signal is extracted from the reference clock (Ref Clk) and the incoming NRZ signal by the DLL. The frequency divider converts the clock signal to the centered frequency (f_c) of the SAW filter. The clock multiplier converts the output signal of the SAW filter into the high-frequency clock signal. Lastly, the sample-and-hold circuit, triggered on the Syn Clk obtained from clock multiplier, samples the input data and holds on its last sampled value until the next Clk pulse reaches to it. The main components for

jitter generation are the DLL circuit and clock multiplier. The jitter generation of DLL is decreased by adjusting the loop gain of DLL. The jitter generation also depends on the multiplication ratio (m) of the clock multiplier. The multiplication ratio should be set lower than 16 to get the jitter generation below 3.6 mUI rms. Here, we have kept the multiplication ratio below 16.

Delay-locked Loop

Architecture of the DLL circuit is shown in Fig. 5.47 (b). Synchronous clock (Syn Clk) is extracted from a reference clock (Ref Clk) by Voltage Variable Delay Line (VVDL) which is controlled by a feedback loop. The loop regulates the phase between Clk and data close to zero with the following basic principle. The data and Clk signals drive a charged pumped Phase Detector (PD) whose output is filtered by first-order loop filter to generate a stable loop control voltage (V_c).

The output of the charge pumped phase-detector circuit is given by

$$I_{PDI} = K_{PDI}\,(I_{pump})\,\Delta\Phi \tag{5.50}$$

where, $\Delta\Phi$ = Phase difference of clock and NRZ data, $K_{PDI}\,(I_{pump})$ = Charge pumped PD gain. The output of the loop-filter circuit is given by

$$V_c = K_F\,(s)\,K_{PDI}(I_{pump})\,\Delta\Phi \tag{5.51}$$

where, $s = j\omega$ = Complex frequency and $K_F(s)$ = Loop-fitter gain. Here, the loop filter is a simple capacitor (C) that gives $K_F(s) = \dfrac{1}{SC}$. The charge-pumped loop circuit regulates V_c controlled by high-frequency PD signals. Figure 5.47 shows the variation of V_c with C/I_{pump}. It is evident from the figure that for the lower value C/I_{pump}, V_c decreases nonlinearly and for higher C/I_{pump}, it decreases slowly and linearly with C/I_{pump}. V_c is almost saturated at C/I_{pump} ~12.5 pF/mA. The jitter increases with the reduction of C/I_{pump} because V_c has larger effect on time delay created by VVDL. In order to decrease the jitter, we choose higher C/I_{pump} for which V_c decreases slowly.

VVDL is an important part of DLL circuit. It consists of a multistage adjustable delay inverter as shown in Fig. 5.48. The VVDL does not generate any signal, rather it delays the Ref Clk signal by a time given by

$$t_o = K_V(V_c, N).\,V_c \tag{5.52}$$

where, $K_V\,(V_c, N)$ has units of seconds/V and N is the number of stages of delay inverter in VVDL. It is seen that $K_V\,(V_c, N)$ remains constant for a certain range of V_c ($V_{max} < V_c < V_{min}$) and for V_c below V_{min} and V_c above V_{max} increases with V_c. This range decreases with increase of N. Moreover, if the VVDL circuit produces long delay, the rising clk edge would arrive late at PD related to data edge. So

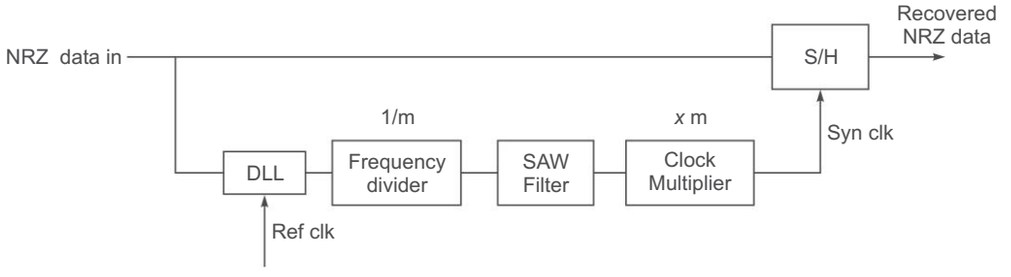


Fig. 5.47 (a) Data recovery circuit based on DLL

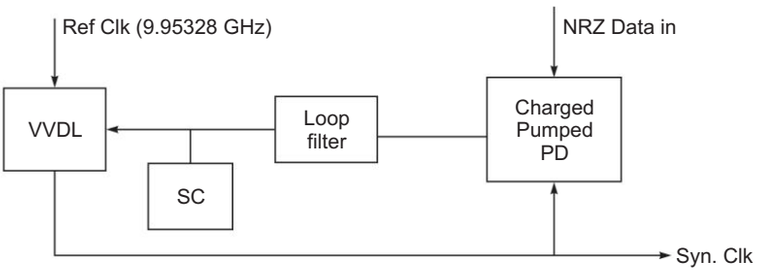


Fig. 5.47 (b) Delay locked loop (DLL)

the time delay produced by VVDL has to be restricted by V_c value which should be kept in between V_{max} and $V_{min.}$ This can be performed by Self-Correcting (SC) circuit, that compares V_c with predefined voltages V_{max} and $V_{min.}$ The jitter increases with N since K_V (V_c, N) increases with N. Further, taking less number of delay inverters gives less power consumption.

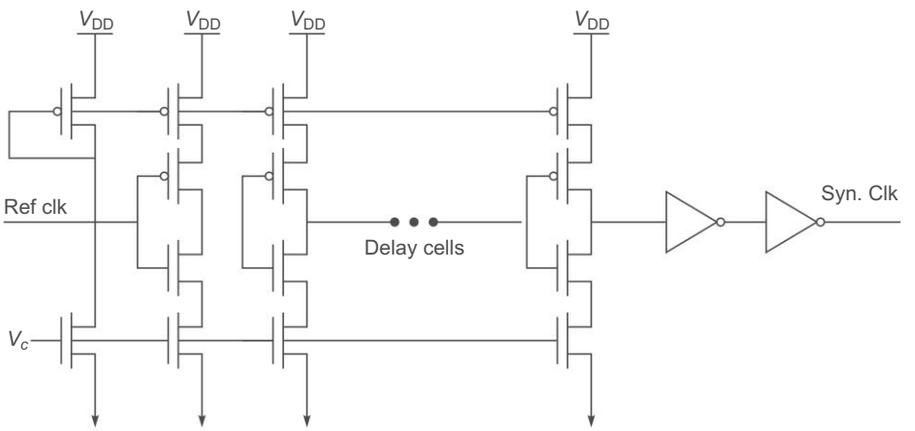


Fig. 5.48 Voltage variable delay line based on cascaded multistage current-starved delay inverter

5.8 □ SPREAD SPECTRUM SIGNALING

Spread spectrum involves spreading the desired signal over a bandwidth much larger than the minimum bandwidth necessary to send the signal. It was originally developed by the military as a method of communications that is less sensitive to intentional interference or jamming by third parties, but has become very popular in the era of personal communications recently. Spread spectrum methods can be combined with Code Division Multiple Access (CDMA) methods to create multi-user communications systems with very good interference performance. It can be used to provide multipath rejection in good ground-based mobile radio environment. Secret messaging system can employ spread spectrum to avoid detection by other persons. For example, the operator of an enemy receiver may attempt to begin transmitting an interference signal to block communication between the transmitter and receiver.

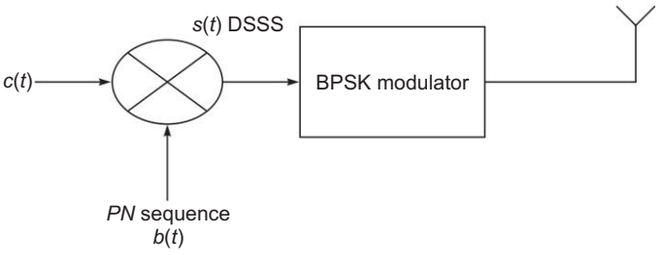
It is used in mobile communication and local area networks. Here again, spread spectrum acts to reduce the effective power of interference so that interference can proceed with least interference. With the emergence of home entertainment automation and information devices that are capable of being interconnected in home networks, there is an increasing demand in the use of wireless communication in the era of latest communication. In this direction the latest communication system should be immune to noise and intentional impairment to the system with least bit error and bandwidth efficiency. In this technique, analog data or digital data can be transmitted using analog signals. There are two types of spread spectrum—Direct-Sequence Spread Spectrum (DSSS) and Frequency-Hopped Spread Spectrum (FHSS)

5.8.1 Direct Sequence Spread Spectrum (DSSS)

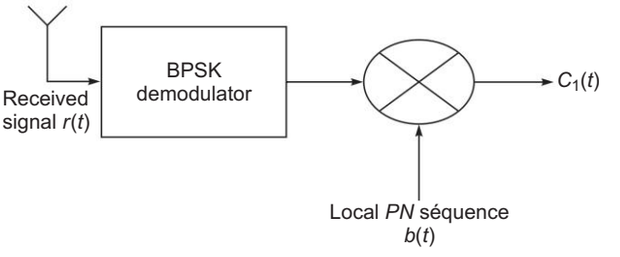
The two major spread-spectrum methods differ mainly in the way they encode the data with the *PN* sequence. In DSSS, the carrier (data signal) is modulated by the *PN* code sequence, which is of a much higher frequency than the desired data rate version of spread.

The DSSS signal is obtained by multiplying the data bit with the *PN* signal. The resultant signal will have a spectrum that is nearly the same as the wide-band *PN* signal. Figure 5.49 shows the signals, the data signal for one pulse width, and the *PN* sequence over the same time and resultant signal. We may express the transmitted DSSS signal as message signal $C(t)$ multiplying with *PN* sequence $b(t)$ using exclusive or

$$s(t) = c(t) \oplus b(t) \tag{5.53}$$



(a) Block diagram of DSSS transmitter



(b) Block diagram of DSSS receiver

Fig. 5.49 Block diagram of transceiver of DSSS system

The DSSS signal $s(t)$ is modulated with Bipolar Phase-Shift Keying (BPSK).

At the receiving end, the received signal $r(t)$ is demodulated by a BPSK demodulator. It is then multiplied with the locally generated *PN* sequences in the multiplier stage. The message signal is obtained as

$$c_1(t) = s_1(t) \oplus b(t) \tag{5.54}$$

where $s_1(t)$ = Signal after BPSK demodulation. Here, it is assumed that there is perfect synchronization between the transmitter and receiver. The *PN* sequences used at the receiver are the same as used at the transmitter. Also, there is perfect synchronization between the data received and local *PN* sequences.

The block diagram of a four channel transmitter-and-receiver based code phase-shift keying is given below in Fig. 5.50(a) and Fig. 5.50(b) respectively. In this scheme, the data from each channel is grouped into a 4-bit data word called one symbol. With the help of a *PN* generator circuit, equal to total symbols, say *M*, *PN* sequences are generated [2]. Different *PN* sequences are generated from single *PN* sequence with the help of a phase-shift network. Each sequence of 4-bits data word selects one *PN* sequence with the help of a code selector, which is basically 16:1 multiplexer and it is then modulated as BPSK signal and transmitted.

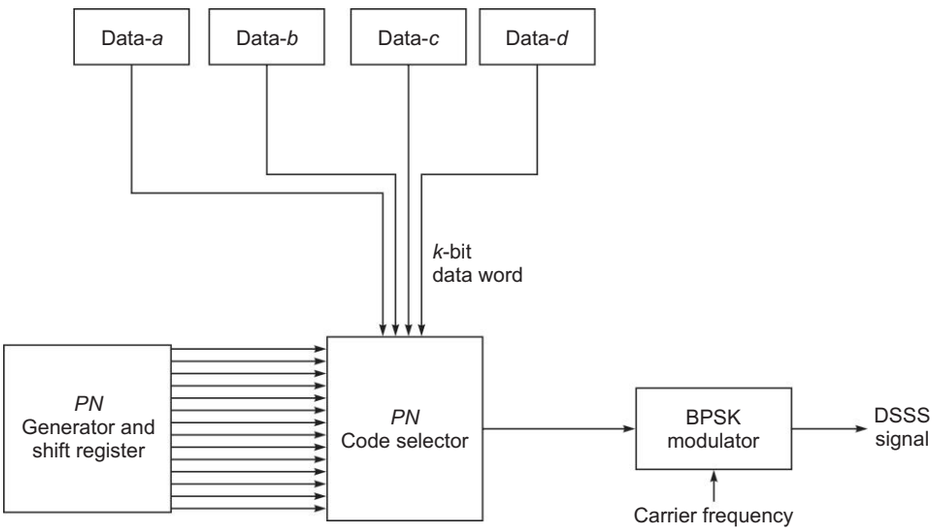


Fig. 5.50 (a) Block diagram of 4-channel CPSK-based DSSS transmitter

In the receiver, the received BPSK signal is demodulated and correlated with the help of locally generated *PN* sequences. So there are *M* correlators. The output of all the correlators are fed to decision device, which selects the largest output fed to the decoder stage. The decoder decodes this largest output in *k*-bit binary data. Then each bit of this *k*-bits data are routed to the respective channels.

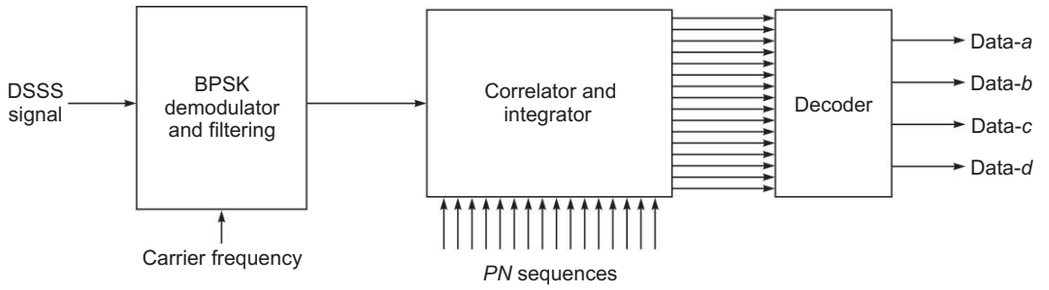


Fig. 5.50 (b) Block diagram of 4-channel CPSK based DSSS receiver

1. PN Generator and Shift Register

A pseudo-noise (*PN*) sequence is a periodic binary sequence with a noise-like waveform, which is generated using a feedback shift register. Here, maximum length *PN* sequence is used for CPSK. Figure 5.51 shows a 15 *PN* sequence generator and shift register block diagram, which consists of 4 *D*-flip flops and two inputs EX_OR gate. The first flip-flop of the *PN* generator is set to 1 with the preset control and the remaining 3 flip-flops are set to 0 with the help of clear control. With each clock pulse, the output of the first flip-flop keeps on shifting to the next stage. The *PN* sequence generated at the output of each flip-flop is repeated after 15 clock pulses.

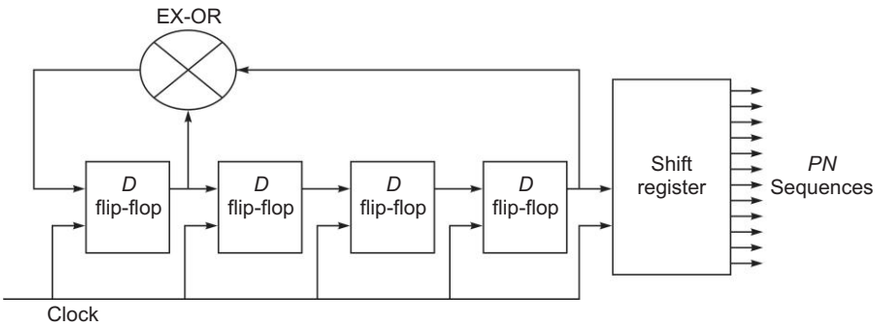


Fig. 5.51 Block diagram of *PN* sequence generator and shift register

The *PN* sequence obtained from the first flip-flop is applied to the shift register network, which consists of 15 flip-flops. Again, all 15 flip-flops are set to the first *PN* sequence with the help of an RC

circuit (**1 1 1 1 0 1 0 1 1 0 0 1 0 0 0**). With each triggering edge of clock, the *PN* sequences keep on shifting to the next stage and it repeats after 15 clock pulses. The 15 phase-shifted *PN* sequences are taped from the output of 15 flip-flops. Table 5.3 below shows the output of a *PN* generator and shifter registers.

So output of the first flip-flop will be **1 1 1 1 0 1 0 1 1 0 0 1 0 0 0** which is one of the *PN* sequence, which repeats after every 15 clock pulses. This *PN* sequence is applied to the shifting network. The shifting network consists of 15 *D* flip-flops connected in series and triggered with the clock of a *PN* generator's circuit to keep them in the same phase. Also all the 15 flip-flops are set to first *PN* sequence with the help of preset and clear controls, i.e**., 0 0 0 1 0 0 1 1 0 1 0 1 1 1 1**.

Table 5.3 Output of shifting network

Clock	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
Initial state	0	0	0	1	0	0	1	1	0	1	0	1	1	1	1
1	1	0	0	0	1	0	0	1	1	0	0	0	1	1	1
2	1	1	0	0	0	1	0	0	1	1	0	0	0	1	1
3	1	1	1	0	0	0	1	0	0	1	1	0	0	0	1
4	1	1	1	1	0	0	0	1	0	0	1	1	0	0	0
5	0	1	1	1	1	0	0	0	1	0	0	1	1	0	0
6	1	0	1	1	1	1	0	0	0	1	0	0	1	1	0
7	0	1	0	1	1	1	1	0	0	0	1	0	0	1	1
8	1	0	1	0	1	1	1	1	0	0	0	1	0	0	1
9	1	1	0	1	0	1	1	1	1	0	0	0	1	0	0
10	0	1	1	0	1	0	1	1	1	1	0	0	0	1	0
11	0	0	1	1	0	1	0	1	1	1	1	0	0	0	1
12	1	0	0	1	1	0	1	0	1	1	1	1	0	0	0
13	0	1	0	0	1	1	0	1	0	1	1	1	1	0	0
14	0	0	1	0	0	1	1	0	1	0	1	1	1	1	0
15	0	0	0	1	0	0	1	1	0	1	0	1	1	1	1

2. Data Word Generator

The purpose of data word generator circuit is to generate 4-bit data word. This 4-bit data word resembles the 4 channels. The circuit diagram of a data word generator consist of 4 JK flip-flops, which are connected in serial-in–serial-out fashion. The clock driving the data-word generator is obtained from the clock *PN* generator circuit by diving it by a divide-by-15 counter. So the *PN* sequences generated

by *PN* generating circuit and data-word generating circuits remains in same phase. The output of the first flip-flop triggers the second flip-flop and the output of the second flip-flop triggers the third flip-flop, and so on. So taking the output from the output of each flip-flop, we get the 4-bit output as given by Table. 5.4. This 4-bit data word is the address word of the code selector module. The code selector module is a multiplexer 16:1. Table 5.4 shows the output of a data-word generator circuit.

Table 5.4 Output of data-word generator

Clock pulses	Code sequence
1	0000
2	0001
3	0010
4	0011
5	0100
6	0101
7	0110
8	0111
9	1000
10	1001
11	1010
12	1011
13	1100
14	1101
15	1110
16	1111

3. Code Sequence Selector and Modulator

Figure 5.52 shows the block diagram of a *PN* sequence selector and modulator. It selects the particular *PN* sequence corresponding to each data word for getting DSSS signal. This is achieved using a 16:1 multiplexer. The data word is connected to the address pins of the multiplexer as address bits and the *PN* sequences are connected to the input pins. The clock pulse duration of the data-word generator is 15 times the clock duration of a *PN* generator circuit. Hence, for each state of the data-word generator, the multiplexer selects one *PN* sequence. The *PN* sequence coming out of the multiplexer is 0 and +5 volt level for low and high level respectively which is converted into –5 volt and +5 volt level by a bi-level shifter. This bi-level shifter is a high-speed op amplifier AD817, working in the comparator configuration. This bi-leveled *PN* coded signal is modulated with the carrier frequency ω_c of 200 kHz to generate DSSS signal. The modulator is multiplier, which multiplies the carrier frequency and bi-level signal.

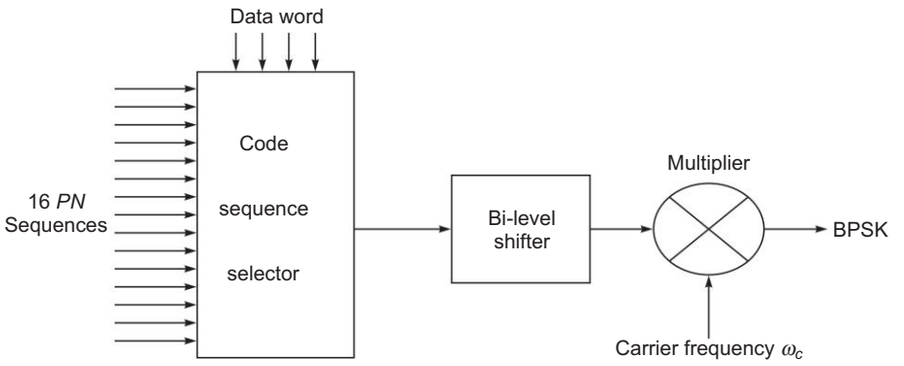


Fig. 5.52 Block diagram of sequence selector and modulator

4. Demodulation and Filtering

Figure 5.53 shows the block diagram of a demodulator and filter. In the figure, the received signal is mixed with carrier frequency in the demodulator. The demodulator is a multiplier, which multiplies both, the received signal and carrier frequency ω_c to detect the received *PN* coded signal. Second-order Butterworth low-pass filter with a cut-off frequency depending upon the carrier frequency is used to filter out the output of multiplier and filter out the spread signal that was initially transmitted. The Sallen–Key filter is designed using high-speed amplifier AD 817. The receiver is tested with different carrier frequencies.

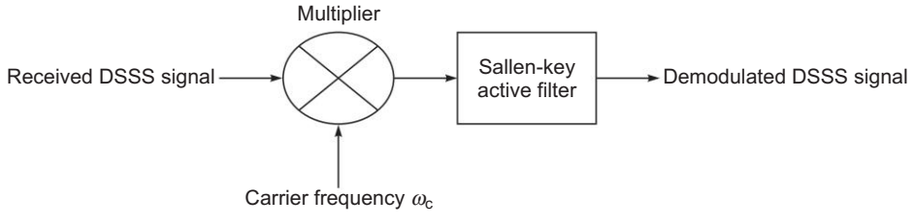


Fig. 5.53 Block diagram demodulator with filter

5. Correlation and Integration

The correlator correlates demodulated signal with the locally generated *PN* sequences (*PN*-1, 2, 3, …16) with the help of an EX-NOR gate. In a 4-channel transmitter and receiver, there are 16 states corresponding to 4-bit address word. So there are 16 correlators and integrators. The received *PN* sequence is fed to all the 16 correlators (one input of EX-NOR gate) and at the other input, the locally generated *PN* sequence is connected. So when the received *PN* sequence is matched with one of the local *PN* sequences, the corresponding EX-OR gate gives the highest output during that *PN* sequence duration. Whereas the output of other correlators will not be high continuously. After correlation, the signal is integrated with the help of integrator circuits. The integrating circuit consists of an *RC* circuit. The capacitor or each integrating circuit keeps on charging during the *PN* cycle duration. In between the

correlator and integrator circuit, one resetting circuit has been introduced. This resetting circuit resets the output of each correlator to zero at the end of integration cycle. Resetting to zero of each capacitor is obtained with the help of switching transistors triggered by a clock. Also, it is seen that there is a phase difference between the received *PN* sequence and locally generated *PN* sequences. This delay is introduced due to time taken by the signal to process through different modules and components. This phase difference has been removed using a delay device, which introduces an appropriate delay in the locally generated *PN* sequence to match with the received *PN* sequences. So the received *PN* sequences are correlated with all the 16 locally generated *PN* sequences and all 16 outputs of the correlator circuit are integrated simultaneously. The magnitude comparator compares the 16 inputs and selects the largest one. The selected largest signal is fed to decoder.

6. Decoder

The decoder circuit decodes this largest output of correlator and integrator circuit into four-bit data words as it was transmitted. So it separates the decoded signals into four channels. This circuit receives all the 16 outputs from the magnitude comparator and largest select module, simultaneously and decodes in four-bit data, as shown in Table 5.5. This is achieved with the help of 4 eight-input OR gates. When any of the input to 8-input OR gate is high, the output of the corresponding gate will be high. Table 5.5 shows the connection of the different outputs from correlator and integrator module to the 4 gates of the decoder.

Table 5.5 Different inputs and output to 8-inputs OR gate

Inputs	8-I*n*puts OR-Gate	Outputs
$A_1, A_3, A_5, A_7, A_9, A_{11}, A_{13}, A_{15}$	OR Gate for channel-1	Y_0
$A_2, A_3, A_6, A_7, A_{10}, A_{11}, A_{14}, A_{15}$	OR Gate for channel-2	Y_1
$A_4, A_5, A_6, A_7, A_{12}, A_{13}, A_{14}, A_{15}$	OR Gate for channel-3	Y_2
$A_8, A_9, A_{10}, A_{11}, A_{12}, A_{13}, A_{14}, A_{15}$	OR Gate for channel-4	Y_3

5.8.2 Frequency Hopping Spread Spectrum (FHSS)

Frequency hopping is a form of spread spectrum in which spreading takes place by hopping from frequency to frequency over a wide band. A hopping table generated with the help of a pseudo-noise code sequence determines the specific order in which the hopping occurs. The rate of hopping is a function of the information rate. The order of frequencies that is selected by the receiver is a function of pseudo-noise sequences. Here, the transmitted spectrum of a frequency-hopping signal is quite different from that of a direct sequence signal. It is sufficient to note that the data is spread out over a signal band larger than is necessary to carry it. The block diagram of a frequency-hopping transmitter and receiver is shown in Fig. 5.54. In the transmitter shown in Fig. 5.54(a), the data signal $d(t)$ consiting of binary data are applied to an M-ary FSK modulator. The resulting modulated wave and the output from a digital frequency synthesizer which is controlled by *PN* sequences are mixed via a mixer that

consists of a multiplier followed by a band-pass filter. The filter is designed to select the sum frequency component resulting from the multiplication process as the transmitted signal. In particular, successive k-bit segments of a *PN* sequence drive the frequency synthesizer, which enables the carrier frequency to hop over ${}_2k$ distinct values. On a single hop, the bandwidth of the transmitted signal is the same as that resulting from the use of a conventional MFSK with an alphabet of $M = {}_2k$ orthogonal signals. However, for a complete range of ${}_2k$ frequency hops, the transmitted FH/MFSK signal occupies a much larger bandwidth.

In the receiver shown in Fig. 5.54(b), the frequency hopping is first removed by mixing the received signal with the output of a local frequency synthesizer that is synchronously controlled in the same manner as that in the transmitter. The resulting output is then band-pass filtered, and subsequently processed by a noncoherent M-ary FSK demodulator. There are two types of frequency-hop spread spectrum—slow frequency hopping and fast frequency hopping.

In the slow frequency-hopping scheme, the several symbols are transmitted on each frequency hop, so the signal stays in a particular sub-band for a long time relative to the data rate. The hop rate is less than the base-band message bit rate. As shown in Fig. 5.54, during each hop, three bits (symbols) are transmitted.

In the fast frequency-hopping scheme, the carrier frequency will change several times during the transmission of one symbol. Here, chipping rate is greater than the base-band data rate. In this case, one message bit is transmitted by two or more frequency-hopped RF signals. This technique is used to defeat the smart jammer.

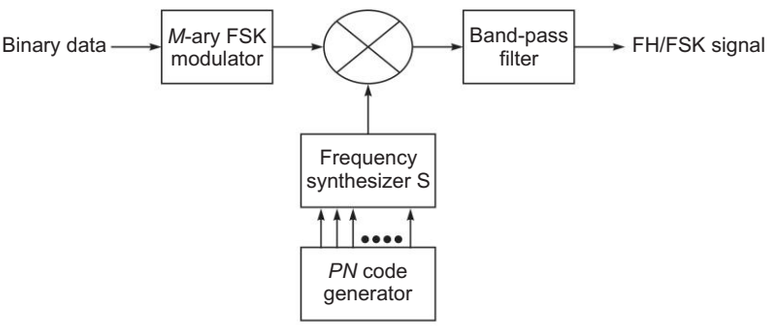


Fig. 5.54 (a) Block diagram of FHSS transmitter

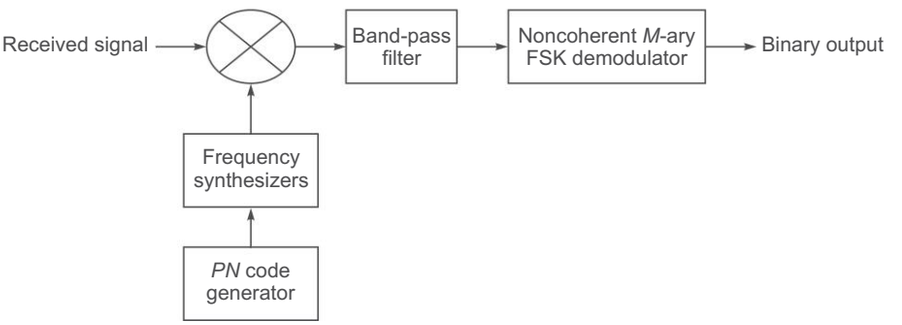


Fig. 5.54 (b) Block diagram of FHSS receiver

FHSS based on Code M-ary Frequency Shift Keying Technique

The code *M*-ary frequency shift keying technique is based on generating different frequencies and coding each frequency with the suitable code and transmitting it. The transmitted signal is hopping from one frequency to another as in case of simple frequency hopping spread spectrum. This scheme is suitable for multiple channels. This technique is suitable for wireless communication, which results in higher throughput. Figure 5.55 (a) shows the block diagram of the transmitter of this scheme, whereas Fig. 5.55 (b) shows the receiver of this scheme. In the transmitter, the data from each channel are grouped into *K*-data called asymbol which has 2^K combinations. The 2^K *PN* sequences are generated by *PN* sequence generator. Corresponding to each symbol, one *PN* code is selected with the help of code selector and converted into analog voltage with the help of D/A converter. 2^K different analog voltages are generated with the D/A converter, which are applied to the voltage-controlled oscillator (VCO). The VCO generates 2^K different frequencies corresponding to each analog level.

Code *M*-ary frequency-shift-keying technique codes at the transmitting end for each sequence (symbol). The output of VCO is amplified and transmitted as simple sinusoidal signal. At the receiving end, reverse operation takes place. The signal mixed with noise is received, amplified and fed to 2^K band-pass filters. Each filter is tuned to one of the VCO frequencies. The band-pass filters separate each of the frequencies which is then fed to 2^K magnitude comparators and selected largest device. The output of the selected largest device is decoded *K*-bit data word in to Ch-1, Ch-2…Ch-*k*. Finally, each bit of the data word is routed to respective channels. *PN* sequence generator, comparator and decoder have already been discussed in DSSS.

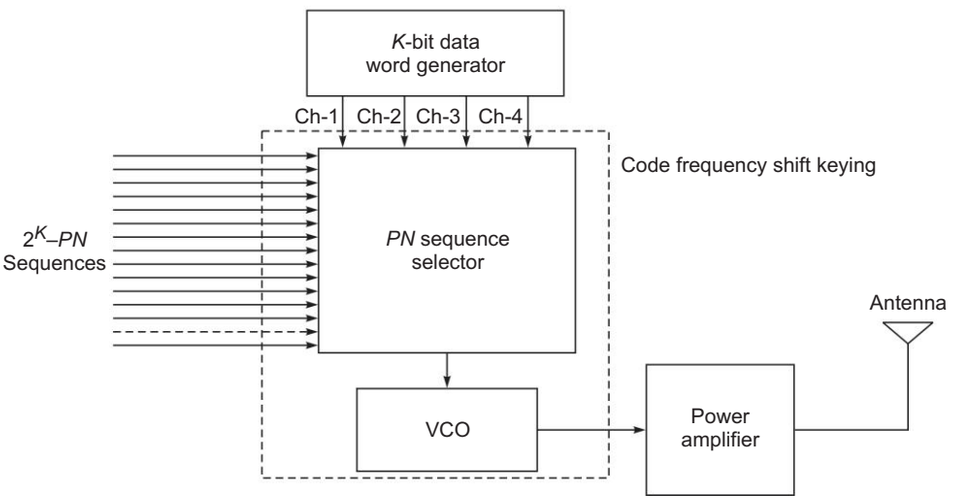


Fig. 5.55 (a) Block diagram of CFSK based FH transmitter

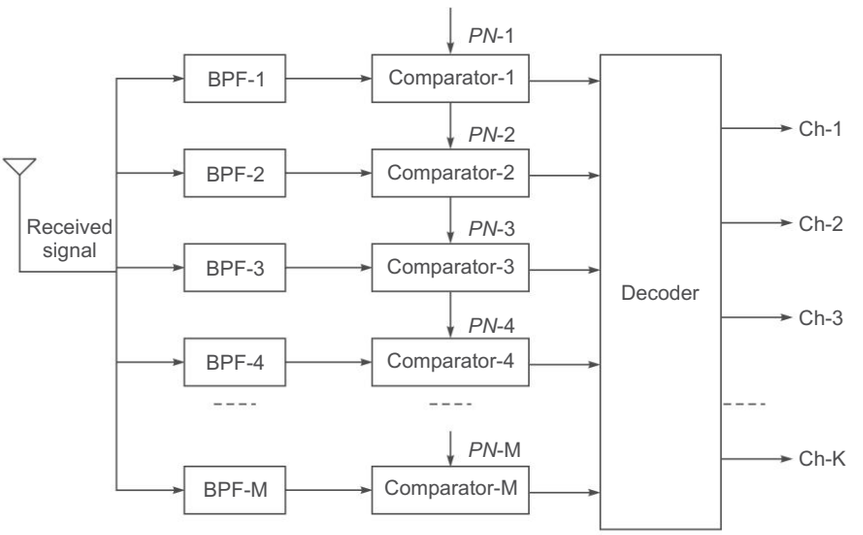


Fig. 5.55 (b) Block diagram of frequency-hopped receiver

□ REFERENCES

5.1. D.J. Allstot, "A Precision Variable-Supply CMOS Comparator," *IEEE Journal of Solid-State Circuits,* Vol. SC-17, No. 6, pp. 1080–1087, December 1982.

5.2. M. Bazes, "Two Novel Full Complementary Self-Biased CMOS Differential Amplifiers," *IEEE Journal of Solid-State Circuits,* Vol. 26, No. 2, pp. 165–168, February 1991.

5.3. B.S. Song, S. Lee, and M.F. Tompsett, "A 10-b 15 MHz CMOS Recycling Two-Step A/D Converter," *IEEE Journal of Solid-State Circuits,* Vol. 25, No. 6, pp. 1328–1338, December 1990.

5.4. M.G. Degrauwe, J. Rijmenants, E.A. Vittoz, and H.J. DeMan, "Adaptive Biasing CMOS Amplifiers," *IEEE Journal of Solid-State Circuits,* Vol. SC-17, No. 3, pp. 522–528, June 1982.

5.5. E.A. Vittoz, "Micropower Techniques," Chapter 3 in J.E. Franca and Y. Tsividis (eds.) *Design of Analog-Digital VLSI Circuits for Telecommunications and Signal Processing,* 2nd ed., Prentice Hall, 1994, ISBN 0-13-203639-8.

5.6. S. Soclof, *Applications of Analog Integrated Circuits,* Prentice Hall, 1985, ISBN 0-13-039173-5.

5.7. M. Ismail, S.C. Huang, and S. Sakurai, "Continuous-Time Signal Processing," Chapter 3 in M. Ismail and T. Fiez (eds.), *Analog VLSI: Signal and Information Processing,* McGraw Hill, 1994, ISBN 0-07-032386-0.

5.8. R. Gregorian and G.C. Temes, *Analog MOS Integrated Circuits for Signal Processing,* John Wiley and Sons, 1986, ISBN 0-471-09797-7.

5.9. H.J. Song and C.K. Kim, "An MOS Four-Quadrant Analog Multiplier Using Simple Two-Input Squaring Circuits with Source Followers," *IEEE Journal of Solid-State Circuits,* Vol. 25, No. 3, pp. 841–848, June 1990.

5.10. D.J. Allstot and W.C. Black, "Technology Design Considerations for Monolithic MOS Switched-Capacitor Filtering Systems," *Proceedings of the IEEE,* Vol. 71, No. 8, pp. 967–986, August 1983.

5.11. J. Shieh, M. Patil, and B. Sheu, "Measurement and Analysis of Charge Injection in MOS Analog Switches," *IEEE Journal of Solid State Circuits,* Vol. 22, No. 2, pp. 277–281, April 1987.

5.13. G. Wegmann, E. Vittoz, and F. Rahali, "Charge Injection in Analog MOS Switches," *IEEE Journal of Solid State Circuits,* Vol. 22, No. 6, pp. 1091–1097, December 1987.

5.14. C. Eichenberger and W. Guggenbuhl, "On Charge Injection in Analog MOS Switches and Dummy Switch Compensation Techniques," *IEEE Transactions on Circuits and Systems,* Vol. 37, No. 2, pp. 256–264, February 1990.

5.15. J. McCreary and P.R. Gray, "All MOS Charge Redistribution Analog-to-Digital Conversion Techniques—Part 1," *IEEE Journal of Solid State Circuits,* Vol. 10, pp. 371–379, December 1975.

5.16. P.W. Li, M.J. Chin, P.R. Gray, and R. Castello, "A Ratio-Independent Algorithmic Analog-to-Digital Conversion Technique," *IEEE Journal of Solid-State Circuits,* Vol. SC-19, No. 6, pp. 828–836, December 1984.

5.17. E.J. Kennedy, *Operational Amplifier Circuits: Theory and Applications,* Holt, Rinehart and Winston, New York, 1988.

5.18. R.W. Broderson, P.R. Gray and D.A. Hodges, "MOS Switched-Capacitor Filters," *Proceedings of the IEEE,* Vol. 67, No. 1, pp. 212–226, January 1979.

5.19. K. Martin, "Improved Circuits for the Realization of Switched-Capacitor Filters," *IEEE Transactions on Circuits and Systems,* Vol. CAS-27, No. 4, pp. 237–244, April 1980.

5.20. R. Gregorian, K.W. Martin, and G. Temes, "Switched-Capacitor Circuit Design," *Proceedings of the IEEE,* Vol. 71, No. 8, pp. 941–966, August 1983.

5.21. A.G. Dingwall and V. Zazzu, "An 8-MHz subranging 8-bit A/D Converter," *IEEE Journal of Solid-State Circuits,* Vol. SC-20, No. 6, pp. 1138–1143, December 1992.

5.22. B. Razavi and B. A. Wooley, "Design Techniques for High-Speed, High-Resolution Comparators," *IEEE Journal of Solid-State Circuits,* Vol. 27, No. 12, pp. 1916–1926, December 1992.

5.23. S. Masuda, Y. Kitamura, S. Ohya, and M. Kikuchi, "CMOS Sampled Differential Push-Pull Cascode Operational Amplifier," *IEEE International Symposium on Circuits and Systems,* Vol. 3, pp. 1211–1214, 1983.

5.24. R.L. Geiger, P.E. Allen, and N.R. Strader, *VLSI—Design Techniques for Analog and Digital Circuits,* McGraw-Hill Publishing Co., 1990.

5.25. R.E. Suarez, P.R. Gray, and D.A. Hodges, "All-MOS Charge Redistribution Analog-to-Digital Conversion Techniques—Part II," *IEEE Journal of Solid State Circuits,* Vol. 10, No. 6, pp. 379–385, December 1975.

5.26. M.J.M. Pelgrom et. al, "25-Ms/s 8-bit CMOS A/D Converter for Embedded Application," *IEEE Journal of Solid-State Circuits,* Vol. 29, No. 8, pp. 879–886, August 1994.

5.27. N. Shiwaku, "A Rail-to-Rail Video-band Full Nyquist 8-bit A/D Converter," *Proceedings of the 1991 Custom Integrated Circuits Conference.*

5.28. B. Razavi and B.A. Wooley, "A 12-b, 5-MSample/s Two-Step CMOS A/D Converter," *IEEE Journal of Solid State Circuits,* Vol. 27, No. 12, pp. 1667–1678, December 1992.

5.29. J. Dornberg, P.R. Gray, and D.A. Hodges, "A 10-bit, 5M sample/s CMOS Two-Step Flash ADC," *IEEE Journal of Solid State Circuits,* Vol. 24, No. 2, pp. 241–249, April 1989.

5.30. T. Shimizu, et al., "A 10-bit, 20 MHz Two-Step Parallel A/D Converter with Internal S/H," *IEEE Journal of Solid State Circuits,* Vol. 24, No. 1, pp. 13–20, February 1989.

5.31. B.S. Song, S.H. Lee, and M.F. Tompsett, "A 10-bit 15 MHz CMOS Recycling Two-Step A/D Converter," *IEEE Journal of Solid State Circuits,* Vol. 25, No. 12, pp. 1328–1338, December 1990.

5.32. B.S. Song, M.F. Tompsett, and K.R. Lakshmikumar, "A 12-bit, 1M Sample/s Capacitor Error-Averaging Pipelined A/D Converter," *IEEE Journal of Solid State Circuits,* Vol. 23, No. 6, pp. 1324–1333, December 1988.

5.33. S. Sutarja and P.R. Gray, "A Pipelined 13-bit, 250-ks/s, 5-V Analog-to-Digital Converter," *IEEE Journal of Solid State Circuits,* Vol. 23, No. 6, pp. 1316–1323, December 1988.

5.34. P. Vorenkamp and J.P.M. Verdaasdonk, "A 10 b 50 Ms/s Pipelined ADC," *IEEE ISSCC Digest of Technical Papers,* pp. 34–35, February 1992.

5.35. J.L. McCreary and P.R. Gray, "All-MOS Charge Redistribution Analog-to-Digital Conversion Techniques—Part I," *IEEE Journal of Solid State Circuits,* Vol. 10, No. 6, pp. 371–379, December 1975.

5.36. K. Bacrania, "A 12 Bit Successive-Approximation ADC with Digital Error Correction," *IEEE Journal of Solid State Circuits,* Vol. 21, No. 6, pp. 1016–1025, December 1986.

5.37. Y. Matsuya, K. Uchimura, et al, "A 16-bit Oversampling A/D Conversion Technology Using Triple Integration Noise Shaping," *IEEE Journal of Solid State Circuits,* Vol. 22, No. 6, pp. 921–929, December 1987.

5.38. P.P. Sahu, " Improvement of Jitter characteristics of a 9.95328 Gb/s Data recovery DLL using SAW filter", *Computers & Electrical Engineering Journal* Elsevier, Vol-33(2), pp 127–132, 2007.

5.39. P.P. Sahu and M. Singh, "Multichannel frequency hopping spread spectrum signaling using code *M*-ary frequency shift keying" *Computers & Electrical Engineering Journal,* Elseiver,. Vol-34(4), pp 338–345, 2008.

5.40. P.P. Sahu and M. Singh, "Multichannel Direct Sequence Spread Spectrum Signaling using Code Phase Shift Keying," *Computer & Electrical Engineering*, Elsevier, Vol-35(1), pp 218–226, 2009.

5.41. M. Singh and P.P. Sahu, "4-channel transmitter and receiver using CPSK based direct sequence spread spectrum," *International Journal HIT trans ECCN*, vol-1(1), pp 63–69, 2006.

EXERCISES

5.1 A very important component of a comparator is its offset voltage. The offset voltage of a comparator can be modeled as a dc voltage source in series with the gate of the MOSFET used in the input diff-pair (Fig. P5.1). Find the output offset voltage.

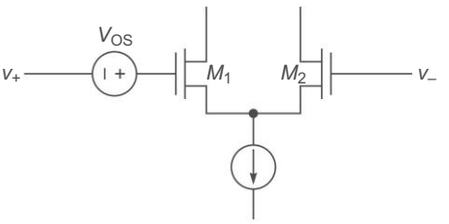


Fig. P5.1

5.2 Can the self-biased comparator be used as a wide-swing op-amp? If so, how would the op-amp be compensated?

5.3 Sketch the schematic of an adaptive voltage follower that can source or sink current.

5.4 Draw the single-ended (output) version of the sample-and-hold amplifier and describe, using timing diagrams, the operation of the circuit.

5.5 Show that the switched-capacitor circuits shown in Fig. P5.2 behave like resistors and find the value of resistance.

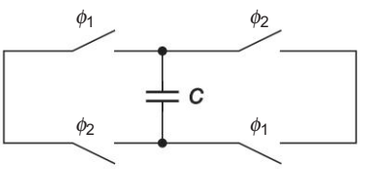


Fig. P5.2

5.6 Draw the fully differential switched-capacitor integrator made using a differential input/output op-amp and find the transfer function of this topology.

5.7 Suppose the op-amp in Fig. P5.3 is used with a feedback factor of 0.5. Estimate the minimum unity gain frequency, f_u, that the op-amp must possess.

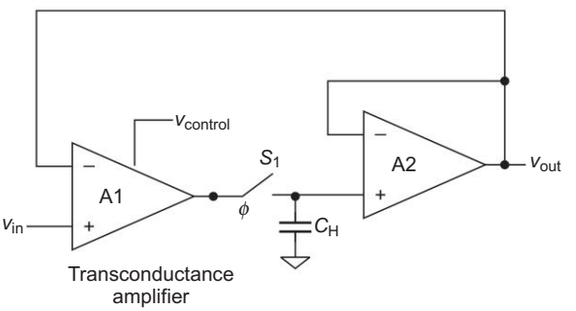


Fig. P5.3

5.8 A 3-bit resistor string DAC similiar to the one shown in Fig. P5.4 was designed with a desired resistor of 500 Ω. After fabrication, mismatch caused the actual value of the resistors to be R_1 = 500, R_2 = 480, R_3 = 470, R_4 = 520, R_5 = 510, R_6 = 490, R_1 = 530, and determine the maximum INL and DNL for the DAC assuming V_{REF} = 5 V.

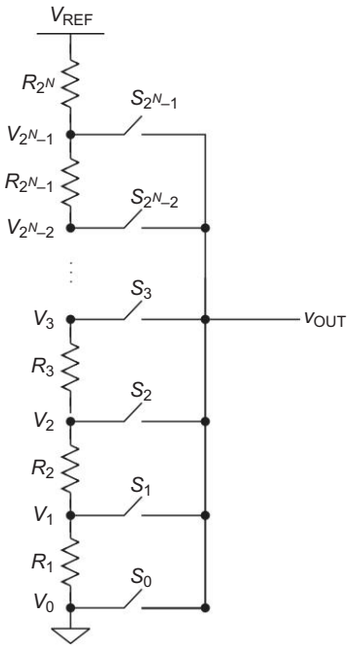


Fig. P5.4

5.9 Compare the digital input codes necessary to generate all eight output values for a 3-bit resistor string DAC similiar to those shown in Fig. 5.39. Design a digital circuit that will allow a 3-bit binary digital input code to be used for the DAC in Fig. P5.4. Discuss the advantages and disadvantages of both architectures.

5.10 Suppose we have 4-bit *R*-2*R* DAC contained resistors that were perfectly matched and that R = 1 kQ and V_{REF} = 5 V. Determine the maximum switch resistance that can be tolerated for which the converter will still have 3-bit resolution. What are the values of INL and DNL?

5.11 Design a 3-bit current steering DAC using the generic current steering DAC shown in Fig. P5.5 Assume that each current source, /, is 5 mA, and find the total output current for each input code.

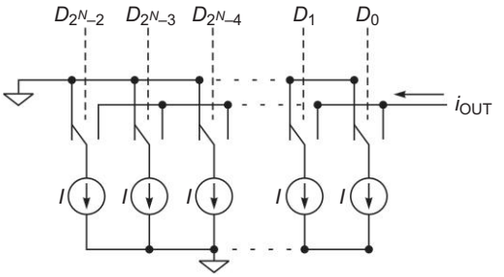


Fig. P5.5

Design an 8-bit current steering DAC using binary-weighted current sources. Assume that the smallest current source will have a value of 1 μA. What is the range of values that the current source corresponding to the MSB can have while maintaining an INL of Y_i LSB? Repeat for a DNL less than or equal to V_i LSB.

5.12 A certain process is able to fabricate matched current sources within 0.05 percent. Determine the maximum resolution that a current steering (nonbinary weighted) DAC can attain using this process.

5.13 Prove that the 3-bit charge scaling DAC used in Fig. 5.42 has the same output voltage increments as the R-2R DAC for V_{REF} = 5 V and C = 0.5 pF.

Design a 4-bit charge scaling DAC using a split array. Assume that V_{REF} = 5 V and that C = 0.5 pF. Draw the equivalent circuit for each of the following input words and determine the value of the output voltage: D = 0001, 0010, 0100, 1000. Assuming the capacitor associated with the MSB had a mismatch of 4 percent, calculate the INL and DNL.

5.14 Design a 3-bit pipeline DAC using V_{REF} = 5 V. (a) Determine the maximum and minimum gain values for the first-stage amplifier for the DAC to have less than $\pm V_i$ LSBs of DNL assuming the rest of the circuit is ideal. (b) Repeat for the second-stage amplifier. (c) Repeat for the last-stage amplifier. Using the same DAC designed in Problem 5.14, (a) Determine the overall error (offset, DNL, and INL) for the DAC designed in Fig. 5.43 if the S/H amplifier in the first stage produces an offset at its output of 0.25 V. Assume that all the remaining components are ideal. (b) Repeat for the second-stage S/H. (c) Repeat for the last-stage S/H.

5.15 Design a 3-bit Flash ADC with its quantization error centered about zero LSBs. Determine the worst-case DNL and INL if resistor matching is known to be 5 percent. Assume that V_{REF} = 5 V. Using the ADC designed in Fig. 5.28, determine maximum offset which can be tolerated if all the comparators had the same magnitude of offset, but with different polarities, to attain a DNL of less than or equal to $\pm V_i$ LSB.

5.16 A 4-bit Flash ADC converter has a resistor string with mismatch as shown in Table P5.1. Determine the DNL and INL of the converter. How many bits of resolution does this converter possess? V_{REF} = 5 V.

Table P5.1

Resistor	Mismatch (%)
1	2
2	1.5
3	0
4	–1
5	–0.5
6	1
7	1.5
8	2
9	2.5
10	1
11	–0.5
12	–1.5
13	–2
14	0
15	1
16	1

5.17 Determine the open-loop gain required for the residue amplifier of a two-step ADC necessary to keep the converter to within V_i LSB of accuracy with resolutions of (a) 4 bits, (b) 8 bits, and (c) 10 bits.

5.18 Assume that a 4-bit, two-step flash ADC uses two separate Flash converters for the MSB and LSB ADCs. Assuming that all other components are ideal, show that the first Flash converter needs to be more accurate than the second converter. Assume that $V_{REF} = 5$ V.

5.19 Assume that an 8-bit pipeline ADC was fabricated and that all the amplifiers had a gain of 2.1 V/V instead of 2 V/V. If $V_{IN} = 3$ V and $V_{REF} = 5$ V, what would be the resulting digital output if the remaining components were considered to be ideal? What are the DNL and INL for this converter?

5.20 Show that the first-stage accuracy is the most critical for a 3-bit, 1-bit per stage pipeline ADC by generating a transfer curve and determining DNL and INL for the ADC for three cases: (1) The gain of the first-stage residue amplifier set equal to 2.2 V/V. (2) The second-stage residue amplifier set equal to 2.2 V/V (3) The third-stage residue amplifier set equal to 2.2 V/V. For each case, assume that the remaining components are ideal. Assume that $V_{REF} - 5$ V.

5.21 An 8-bit single-slope ADC with a 5 V reference is used to convert a slow-moving analog signal. What is the maximum conversion time assuming that the clock frequency is 1 MHz? What is the maximum frequency of the analog signal? What is the maximum value of the analog signal which can be converted?

5.22 An 8-bit single-slope ADC with a 5 V reference uses a clock frequency of 1 MHz. Assuming all other components to be ideal, what is the limitation on the value of *RC*? What is the tolerance of the clock frequency which will ensure less than 0.5 LSB of INL?

5.23 An 8-bit dual-slope ADC (Fig. 5.33) with a 5 V reference is used to convert the same analog signal in Fig. 5.32. What is the maximum conversion time assuming that the clock frequency is 1 MHz? What is the mimimum conversion time that can be attained? If the analog signal is 2.5 V, what will be the total conversion time?

5.24 Discuss the advantages and disadvantages of using a dual-slope versus a single-slope ADC architecture.

5.25 Design a 3-bit, charge redistribution ADC and determine the voltage on the top plate of the capacitor array throughout the conversion process for $v_{IN} = 2$, 3, and 4 V, assuming that $V_{REF} = 5$ V. Assume that all components are ideal. Draw the equivalent circuit for each bit decision.

5.26 Show that the charge redistribution ADC is immune to comparator offset by assuming an initial offset voltage of 0.3 and determining the conversion for $v_{IN} = 2$ V.

5.27 Discuss the differences between Nyquist rate ADCs and oversampling ADCs.

6

BiCMOS Circuit

BiCMOS is made by using CMOS and bipolar junction transistors (BJT). CMOS is used because of its small layout size and ease of implementing logic while BJTs are used for their high-current capability. To achieve high-speed, high-current-driving bipolar transistors and low-power, high-impedance CMOS devices, it is required to possess CMOS and BJT in the same substrate. This process is called a BiCMOS process. Gate propagation delays of 2 μm BiCMOS process is proportional to few hundred picoseconds which is much smaller than CMOS technology. BiCMOS technology has advantages and disadvantages associated with each. Bipolar device capabilities have been added to some CMOS processes to improve speed, while CMOS device capabilities have been added to some bipolar processes to minimize power dissipation. Microprocessors are particularly well suited for BiCMOS technology. Typically, three generic categories limit microprocessor performance: (1) instructions per task, (2) cycles per instruction, and (3) time per cycle. The third category can be greatly improved by increasing the speed critical blocks. A PC microprocessor was developed using a bipolar-based BiCMOS process. A microprocessor operating at 533 MHz is an example of BiCMOS technology.

6.1 ◻ MODELING OF *npn* BJT

The junction-isolated *npn* bipolar transistor operation is very similar to normal BJT operation, with large parasitic resistances associated with the base and collector. To develop a digital model for the BJT which is similar to the model (as shown in Fig. 6.1) we developed for the MOSFET, we can define the variable R_{npn} by

$$R_{npn} = R_C \tag{6.1}$$

where R_C = Parasitic collector resistance.

The input resistance of the lateral BJT can be estimated by

$$R_{in} = R_b \tag{6.2}$$

where R_b = Parasitic base resistance.

The BJT capacitances result from the depletion capacitances of the implant regions and from the forward-biased base-emitter junction (the storage capacitance). The storage capacitance associated with the base-emitter forward-biased diode is given by

$$C_{storage} = \frac{I_E}{V_T}\tau_L \tag{6.3}$$

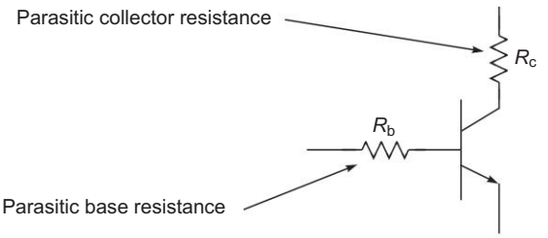


Fig. 6.1 BJT model

where τ_L = Minority carrier lifetime of the base-emitter junction, I_E is the dc emitter current and V_T is the thermal voltage (kT/q). As the emitter current increases, the storage capacitance increases.

6.2 □ THE BICMOS INVERTER

Figure 6.2 shows a BiCMOS inverter consisting of two bipolar transistors T_1 and T_2 with one *n*MOS, M_4 and *p*MOS, M_3 which are in enhancement modes. The operation is straightforward and given below.

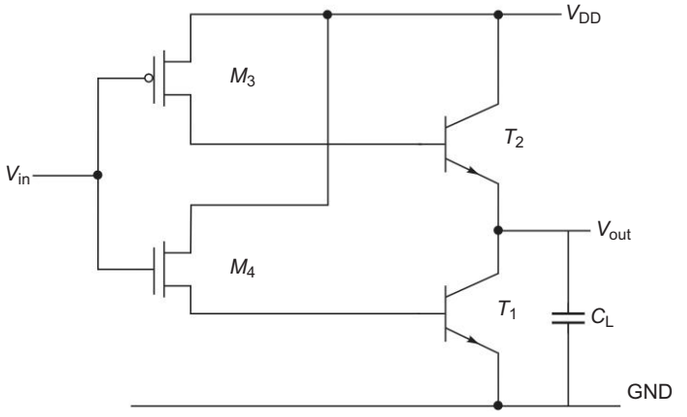


Fig 6.2 BiCMOS inverter

Case 1 With $V_{in} = 0$ (ground), M_4 is off and T_1 is nonconducting but M_3 is on and T_2 is conducting and acts as current source to charge load capacitance to get V_{out} to be V_{DD}.

Case 2 With $V_{in} = V_{DD} = 5$ volts, M_4 is on and T_1 is conducting but M_3 is off and T_2 is not conducting and since T_1 is conducting, the load capacitance discharges through T_1 to make V_{out} to become 0 volt.

So in Case 1, input is low and output is high whereas in Case 2, input is high and output is low. The BiCMOS has the following advantages : (1) low output resistance and high input resistance (2) high current capability and (3) high load current sinking. The main disadvantage is lowering the noise

margins of the logic. The maximum output voltage is approximately $V_{DD} - 0.7$ V, while the minimum logic output voltage is approximately 0.7 V. The 0.7 V drop for the high and low side comes from the base-emitter voltage drop of Q_2 and Q_1, respectively. Caution should be exercised when using the output of BiCMOS gates with CMOS logic. The low-output voltage of 0.7 V is very close to the threshold voltage of the *n*-channel transistor. CMOS gates with switching point voltages close to the threshold voltage are susceptible to noise.

(a) Switching Characteristics The delay associated with the BiCMOS inverter discharging a capacitance, C_L, consists of two parts: the delay in T_1 turning on and the delay once C_L discharging is through T_1. The delay associated with discharging C_L is given by

$$t_L = R_{npn} - C_L \tag{6.4}$$

The low-to-high delay time can be estimated in much the same way as the high-to-low delay. The delay in charging C_L is given by

$$t_H = R_{npn}\, C_L = t_L \tag{6.5}$$

(b) Wide-Swing BiCMOS Inverters Figure 6.3 show a wide-swing BiCMOS inverter. When the input is grounded, MOSFETs M_2 and M_4 are off while MOSFET M_5 is on. MOSFETs M_1 and M_3 can be thought of as resistors. Since M_5 is on, the base of Q_2 is pulled to V_{DD}. The transistor Q_2 is on and pulls the output to $V_{DD} - 0.7$. MOSFET M_3, which behaves like a resistor, then pulls the output up to V_{DD}. When the input is high, M_2 and M_4 are on and M_5 is off. This pulls the base of Q_2 to ground, turning it off. At the same time M_2 turns on, with the output high, causing Q_1 to turn on. Q_1 pulls the output down to 0.7 V. From there, M_1—which behaves like a resistor–pulls the output down to ground. If M_1 or M_3 does not have a large effective resistance (long L), the circuit will not operate correctly.

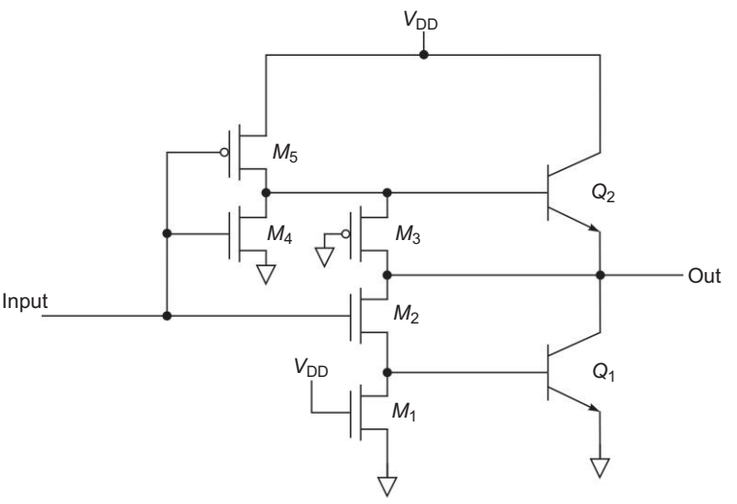


Fig 6.3 Wide-swing BiCMOS inverter

6.3 □ BiCMOS NAND GATE

Figure 6.4 shows BiCMOS-based NAND gate consisting of CMOS devices and BJTs. The operation is straightforward and given below.

Case 1 When $A = 0$ (ground) and $B = 0$ (ground), M_3 and M_4 are on and T_2 is conducting but M_1 and M_2 are off and T_1 is nonconducting. So it acts as a current source to charge load capacitance C_L to get V_{out} to be V_{DD}.

Case 2 When $A = 1$ and $B = 0$ (ground), M_3 is off and M_4 is on and T_2 is conducting but M_1 is on and M_2 is off and T_1 is nonconducting. So it acts as a current source to charge load capacitance C_L to get V_{out} to be V_{DD}.

Case 3 When $B = 1$ and $A = 0$ (ground), M_3 is on and M_4 is off and T_2 is conducting but M_1 is off and M_2 is on and T_1 is nonconducting. So it acts as a current source to charge load capacitance C_L to get V_{out} to be V_{DD}.

Case 4 When $A = 1$ and $B = 1$, M_3 is off and M_4 is off and T_2 is nonconducting but M_1 is on and M_2 is on and T_1 is conducting. So load capacitance C_L discharges through T_1 to get V_{out} to be zero.

The circuit in the figure follows the truth table of NAND gate and so it acts as a NAND gate. The switching analysis is very similar to BiCMOS inverter gate. The delay time is dominated by the time necessary to charge and discharge the load capacitance to get V_{out} to be high and low respectively. The delay associated with the BiCMOS NAND discharging a capacitance, C_L, consists of two parts: the delay in T_1 turning on and the delay once C_L discharging is through T_1. The delay associated with discharging C_L is given by

$$t_L = R_{npn} C_L \tag{6.6}$$

The low-to-high delay time can be estimated in much the same way as the high-to-low delay. The delay in charging C_L is given by

$$t_H = R_{npn}\ C_L = \ t_L \tag{6.7}$$

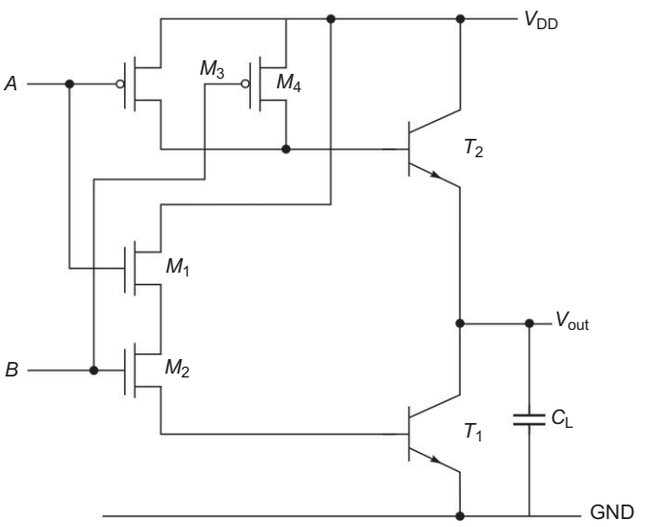


Fig 6.4 BiCMOS NAND gate

6.4 BICMOS NOR GATE

Figure 6.5 shows BiCMOS-based NOR gate consisting of CMOS devices and BJTs. The operation is given below.

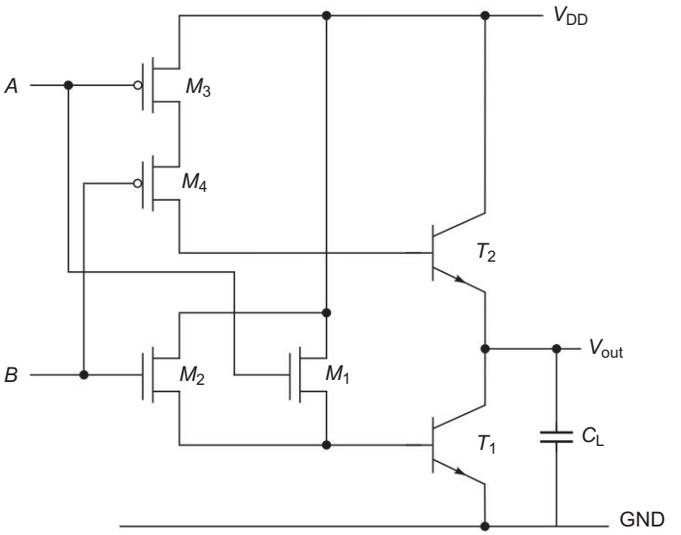


Fig 6.5 BiCMOS NOR gate

Case 1 When $A = 0$ (ground) and $B = 0$ (ground), M_3 and M_4 are on and T_2 is conducting but M_1 and M_2 are off and T_1 is nonconducting. So it acts as a current source to charge load capacitance C_L to get V_{out} to be V_{DD}.

Case 2 When $A = 1$ and $B = 0$ (ground), M_3 is off and M_4 is on and T_2 is non conducting but M_1 is on and M_2 is off and T_1 is conducting. So the load capacitance C_L discharges through T_1 to get V_{out} to be zero.

Case 3 When $B = 1$ and $A = 0$ (ground), M_3 is on and M_4 is off and T_2 is non conducting but M_1 is off and M_2 is on and T_1 is conducting. So the load capacitance C_L discharges through T_1 to get V_{out} to be zero.

Case 4 When $A = 1$ and $B = 1$, M_3 is off and M_4 is off and T_2 is non conducting but M_1 is on and M_2 is on and T_1 is conducting. So load capacitance C_L discharges through T_1 to get V_{out} to be zero.

The circuit in the figure follows truth table of NOR gate and so it acts as a NOR gate. The switching analysis is very similar to BiCMOS NAND gate. The delay time is dominated by the time necessary to charge and discharge the load capacitance to get V_{out} to be high and low respectively.

6.5 CMOS AND ECL CONVERSIONS USING BICMOS

BiCMOS has the ability to convert Emitter Coupled Logic (ECL) to CMOS logic and CMOS logic to ECL circuit. One advantage of using ECL circuits is that the bipolar transistors can double their output current for every 25 mV of change in the base-emitter voltage. This is simply because the collector current, I_c, through a BJT, can be described as

$$I_C = I_S \exp(-v_{BE}/V_T) \tag{6.8}$$

where I_s is the saturation current, V_T is the thermal voltage, and v_{BE} is the instantaneous base-emitter voltage. The expression for the transconductance, which relates the amount of drive current to the input voltage, is

$$g_m(B) = \frac{I_C}{V_T} = \frac{I_S}{V_T} \exp(-v_{BE}/V_T) \tag{6.9}$$

Since it is exponential, the BJTs can sink or source large amounts of load currents with very small input voltage swings. The transconductance for the MOSFET is

$$g_m(M) = \beta(V_{GS} - V_{THN}) \tag{6.10}$$

which is linear with respect to the input voltage. The amount of input voltage necessary to switch an output from a low to a high or a high to a low is much greater than the BJT case.

CMOS logic typically swings between V_{DD} and V_{SS} whereas ECL logic has smaller signal swing defining the logic levels. To increase speed of the switch, interface circuits are needed that convert ECL to CMOS logic levels and from CMOS logic levels to ECL logic levels. Figure 6.6 shows an ECL to CMOS converter circuit in which, the ECL input signal is level shifted by 2 V_{BE} drops to a Current-Mode Logic (CML) circuit that drives the CMOS output shifter stage. The stepped-down ECL input causes the CML circuit to become imbalanced so that one collector is considered high, while the other collector is considered low. The critical issues in minimizing the delay time are the output swing of the CML stage and the sizes chosen for the CMOS shifter. However, the two specifications are inversely proportional, for increasing the output swing of the CML stage decreases the delay through the CMOS shifter but increases the delay through the CML stage.

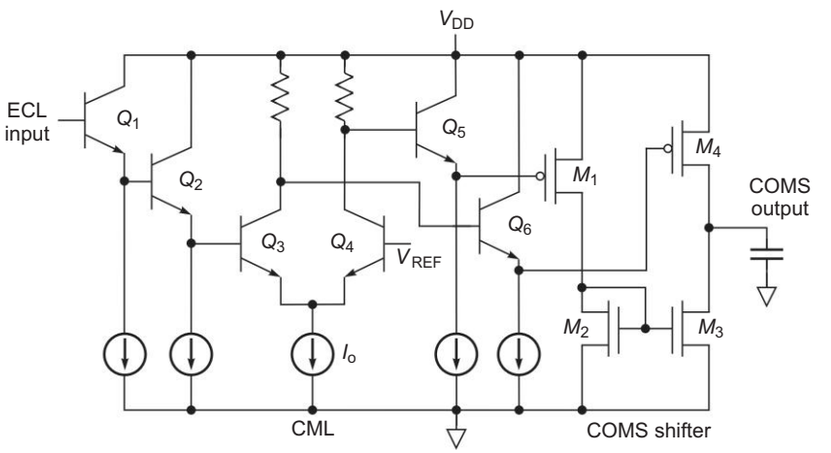


Fig 6.6 ECL to CMOS converter circuit

Figure 6.7 shows another conversion circuit from CMOS logic levels to EC logic levels in which the circuit translates CMOS signals to ECL logic levels and requires a complemented CMOS input. The input signal causes an imbalance in the source coupled pair, since the current, I_0, is constant. The output swing of the source coupled pair appearing at nodes *A* and *B* can be adjusted by wisely choosing the resistance values and input MOS device sizes.

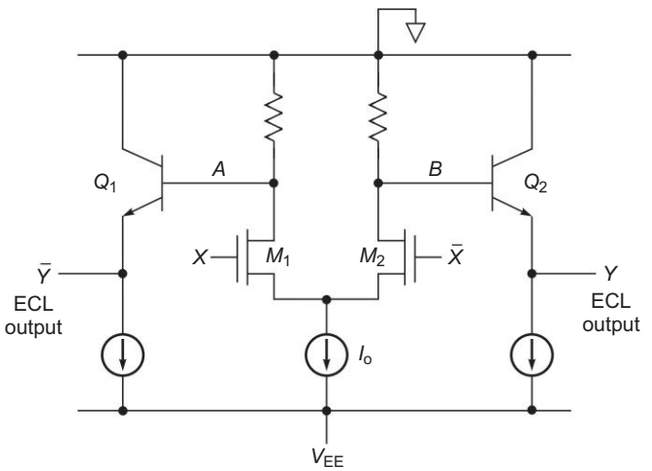


Fig 6.7 CML to ECL converter circuit

REFERENCES

6.1. M. Kubo, I. Masuda, K. Miyata, and K. Ogiue, "Perspective on BiCMOS VLSI's," *IEEE Journal of Solid State Circuits,* vol. 23, no. 1, pp. 5–11, February 1988.

6.2. M.I. Elmasry, "Introduction to BiCMOS Integrated Circuits: A Tutorial," *IEEE BiCMOS Integrated Circuit Design,* IEEE Press, 1994, ISBN 0-7803-0430-6.

6.3. M.I. Elmasry, *BiCMOS Integrated Circuit Design,* IEEE Press, 1992, ISBN 0-7803-0430-6, IEEE order number: PC0346-7.

6.4. S.H.K. Embabi, A. Bellaouar, and M.I. Elmasry, "Analysis and Optimization of BiCMOS Digital Circuit Structures," *IEEE Journal of Solid State Circuits,* vol. 26, no. 4, pp. 676–679, April 1991.

6.5. M. Rau and H.J. Pfleiderer, "An ECL to CMOS Level Converter with Complementary Bipolar Output Stage," *IEEE Journal of Solid-State Circuits,* vol. 30, no. 7, pp. 781–787, July 1995.

EXERCISES

6.1 Design a full-swing BiCMOS output buffer that has an input capacitance of 100 fF or less and will drive 10 pF with a $t_{PHL} + t_{PLH}$ less than 15 ns.

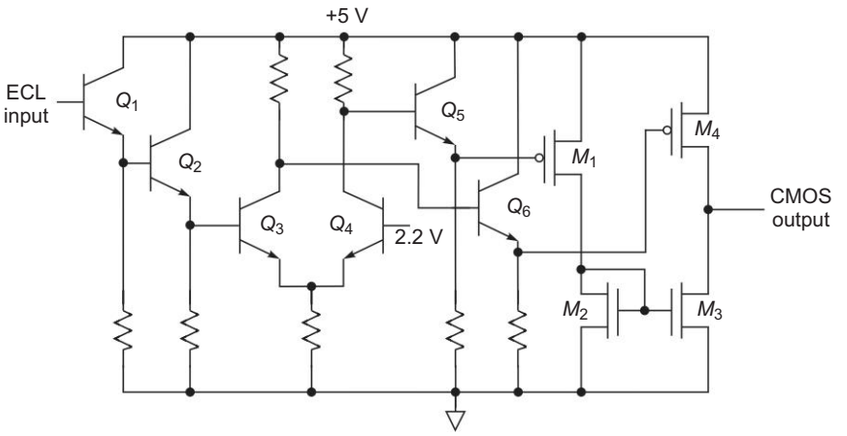


Fig. P6.1

6.2 Design and describe the operation of an ECL to CMOS converter based on the circuit topology shown in Fig. P6.1. Assume that the ECL input varies from 4.2 V (a logic high) down to 3.4 V (a logic low).

6.3 Design 13-to-1 multiplexer using BiCMOS.

6.4 Design 3-to-1 four-bit word multiplexer.

6.5 Design XOR and XNOR by using BiCMOS.

6.6 Design a half adder and full adder by using BiCMOS.

6.7 Design a 4 :1 multiplexer and demultiplexer by using BiCMOS.

6.8 Design an edge-triggered *D* flip-flop and a *J K* master-slave flip-flop by using BiCMOS.

7

Design of Testability

The testing of a chip is an operation in which the chip under test is exercised with carefully selected test patterns (stimuli). The responses of the chip to these test patterns are captured and analyzed to determine if it works correctly. A faulty chip is one that does not behave correctly. The incorrect operation of a chip may be caused by design errors, fabrication errors, and physical failures, which are referred to as faults. Table 7.1 lists a few sample faults in each of these three categories.

Table 7.1 Sample faults found in integrated circuits

Errors	Incorrect chip operation
Design errors	**Incomplete specifications**
	Incorrect logic implementations
	Incorrect wiring
	Design rule violations
	Excessive delays
	Glitches or hazards
	Slow rise/fall times
	Improper noise margins
	Improper timing margins
Fabrication errors	Shorts
	Opens
	Improper doping profiles
	Mask misalignments
	Incorrect transistor threshold voltages
Physical failures	Electron migration
	Cosmic radiation and α-participles

In some cases, we are only interested in whether the chip under test behaves correctly. For example, chips that have been fully debugged and put in production normally require only a pass or fail test. The chips that fail the test are simply discarded. We refer to this type of testing as **fault detection**. In order to certify a prototype chip for production, the test must be more extensive in nature to exercise the circuit

as much as possible. The test of a prototype also requires a more thorough test procedure called **fault location**. If incorrect behaviors are detected, the causes of the errors must be identified and corrected.

An important problem in testing is **test generation**, which is the selection of test patterns. A common assumption in test generation is that the chip under test is nonredundant. A circuit is nonredundant if there is at least one test pattern that can distinguish a faulty chip from a fault-free one.

A nonredundant combinational circuit with n inputs is fault free if and if it responds to all $2n$ input patterns correctly. Testing a chip by exercising it with all its possible input patterns is called an exhaustive test. This test scheme has an exponential time complexity so it is impractical except for very small circuits.

For example, 4.3×10^9 test patterns are needed to exhaustively test a 32-input combinational circuit. Assume that we have a piece of Automatic Test Equipment (ATE) that can feed the circuit with test patterns and analyze its response at the rate of 10^9 patterns per second (1 GHz). The test will take only 4.3 seconds to complete which is long but may be acceptable. However, the time required for an exhaustive test quickly grows as the number of inputs increases. A 64-input combinational circuit needs 1.8×10^{19} test patterns to be exhaustively tested. The same piece of test equipment would need 570 years to go over all these test patterns.

The testing of sequential circuits is even more difficult than combinational circuits. Since the response of a sequential circuit is determined by its operating history, a sequence of test patterns rather than a single test pattern would be required to detect the presence of a fault. There are also other problems in the testing of a sequential circuit, such as the problem of bringing the circuit into a known state and the problem of timing verification.

The first challenge in testing is thus to determine the smallest set of test patterns that allows a chip to be fully tested. For chips that behave incorrectly, the second challenge is to diagnose, or locate, the cause of the bad response. This operation is difficult because many faults in a chip are equivalent; so they are indistinguishable by output inspection. For example, if the output of an inverter incorrectly produces a 0, the following faults are indistinguishable without a further investigation:

- There is a short between the inverter output and V_{SS}.
- There is a short between the inverter input and V_{DD}.
- The pull-down transistor (nMOS) of the inverter is permanently turned on.

Fortunately, a truly exhaustive test is rarely needed. In addition, it is often sufficient to determine that a functional block, instead of an individual signal line or transistor, is the cause of an error. We begin with the discussion of popular fault models that allow practical test procedures to be developed.

7.1 □ FAULT MODELS

As noted above except for very small circuits, it is impractical to pursue an exhaustive test. Instead, a test should consist of a set of test patterns that can be applied in a reasonable amount of time. This test should provide the user with the confidence that the chip under test is very likely to be fault free if it passes the test. An important issue in the development of a test procedure is thus to evaluate the effectiveness of a test. The quality of a test can be judged by an index called **fault coverage**. Fault coverage is defined as the ratio between the number of faults a test detects and the total number of possible faults. This usually is determined by means of a simulated test experiment. The experiment, which is called a **fault simulation**, uses a software model of the chip to determine its response to the test when faults are present. A fault is detected by a test pattern if the circuit response is different from the expected fault-free response.

Fault models are created to facilitate the generation of test patterns. A fault model represents a subset of the faults that may occur in the chip under test. Several fault models have been developed for representing faults in CMOS circuits. These models can be divided into logic fault models, delay fault models, and current-based fault models.

The most widely used fault models are the stuck-open fault, and bridging-fault models. Delay-fault models incorporate the concept of timing into fault models. Examples of delay-fault models are the transition delay and path delay fault models. The current-based fault models were developed by recognizing the very low leakage current of a CMOS circuit. Many defects such as opens, shorts, and bridging, result in a significantly larger current flow in the circuit.

Many Automatic Test-Pattern Generation (ATPG) algorithms are based on the **single-fault assumption**. This assumption assumes that at most one fault exists at any time so the test generation complexity can be significantly reduced. We will revisit this subject later in this chapter. Brief introductions to these fault models are given below. We will discuss some of these models in detail.

1. Stuck-at Fault Model

The stuck-at-fault model assumes that a design error or a fabrication defect will cause a signal line to act as if it were shorted to V_{SS} or V_{DD}. If a line is shorted to V_{SS}, it is a constant 0 and is named a stuck-at-0 (s-a-0) fault. On the other hand, if a line is shorted to V_{DD}, it is a constant 1 and is called a stuck-at-1 (s-a-1) fault. The stuck-at-fault model is most effective if it is used at the inputs and outputs of a logic unit such as a logic gate, a full adder, etc. The application of this fault model in a test is to force a signal line to 1 for a s-a-1 fault. The response circuit is then analyzed.

2. Stuck-Open Fault Model

The stuck-open (s-op) fault model attempts to model the behaviors of a circuit with transistors that are permanently turned off. The result of having transistors that would not be turned on is unique to CMOS circuits. An s-op fault changes a CMOS combinational circuit into a sequential circuit. A two–step test is thus required. The first step brings the signal line being tested to an initial value. The second step then carries out the test in a way similar to the testing of stuck-at faults.

3. Bridging-Fault Model

A bridging-fault model represents the accidental connection of two or more signal lines from wired logic so the original logic function is changed. It is also possible that the circuit becomes unstable if there is an unwanted feedback in the circuit. Bridging faults can be tested by applying opposite values to the signal lines being tested.

4. Transition-Delay Fault Model

The transition-delay fault model is based on the stuck-at model. The application of this fault model is similar to the stuck-at model except that the s-a-0 and s-a-1 values are replaced by 0-to-1 and 1-to-0 transitions. An observed signal line is forced to a known value to some time before it is observed. A transition is then applied and the signal line is observed along with a time assessment. Possible faults

are slow-to-rise (0-to-1 transition) faults and slow-to-fall (1-to-0 transition) faults. This fault model is commonly used to model a functional block that has an excessive delay.

5. Path-Delay Fault Model

The path-delay fault model considers the total transition delay of a circuit path. The only difference between the path-delay fault model and the transition-delay model is that an entire path instead of a single signal node is targeted. The same operation of assigning an initial value and observing a transition is used.

6. Current-Based Fault Model

Some faults cannot be detected by either the logical fault model or the delay-fault model. These defects, which do not cause a logical error but represent a reliability problem, may cause a high leakage current. This defect-induced leakage current may be an order of magnitude larger than the normal CMOS quiescent current. The current-based fault model is only applicable when the chip is in a static state.

Our discussion begins with the stuck-at and stuck-open fault models. Figure 7.1 shows a two-input NAND gate that is used to explain the modeling of physical defects as stuck-at and stuck-open faults. We label the four transistors in this circuit as Q_{pA}, Q_{pB}, Q_{nA}, and Q_{nB}, respectively.

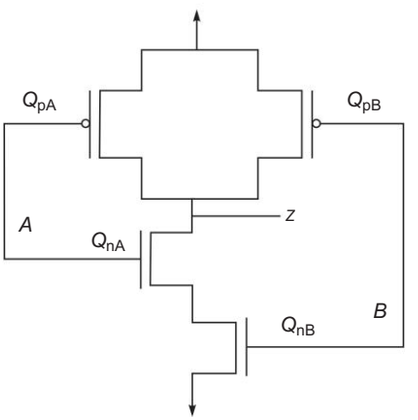


Fig. 7.1 CMOS NAND gate

The stuck-at fault model is often used along with the further assumption that there is at most one fault in the circuit, which is known as the single-fault assumption. The stuck-at fault model is often applied to the inputs and outputs of a logic block. For a logic block with n inputs and m outputs, $2(n + m)$ single stuck-at faults are possible. If we consider only the inputs and output of this NAND gate, six single stuck-at faults are possible.

- ***A***: s-a-0
- ***A***: s-a-1

- ***B***: s-a-0
- ***B***: s-a-1
- ***Z***: s-a-0
- ***Z***: s-a-1

By definition, if input ***A***: s-a-1, the NAND gate sees ***A*** = 1, regardless of the value placed on it. Many stuck-at faults of a circuit are equivalent and thus indistinguishable. For example, input ***A***: s-a-0 is equivalent to and thus indistinguishable from output ***Z***: s-a-1. When equivalent stuck-at faults are taken into consideration, the number of single stuck-at faults in the NAND gate is reduced to four: ***A***: s-a-1, ***B***:s-a-1, ***Z***:s-a-1, and ***Z***:s-a-0.

A stuck-open fault occurs if transistor $\boldsymbol{Q}_{\mathbf{nA}}$ is permanently opened. A possible cause of this stuck-open fault is that either its source or drain is disconnected from the rest of the circuit. In the presence of $\boldsymbol{Q}_{\mathbf{nA}}$: s-op, when ***AB*** = 11, output ***Z*** becomes a high impedance state (*HiZ*) since it is connected neither to V_{SS} nor to V_{DD}. Output ***Z*** thus retains its previous logic value; so the circuit acts like a sequential circuit. The length of time before the value at ***Z*** changes is determined by the capacitance of the node ***Z*** and the leakage current.

The stuck-at and stuck-open fault models do not cover all physical faults in a CMOS circuit. For example, consider a bridging between the source and drain of transistor $\boldsymbol{Q}_{\mathbf{nA}}$. This is equivalent to a situation in which transistor $\boldsymbol{Q}_{\mathbf{nA}}$ is turned on all the time. When $AB = 01$, a voltage divider is formed by transistors $\boldsymbol{Q}_{\mathbf{pA}}$ and $\boldsymbol{Q}_{\mathbf{nB}}$, which drives output Z to an intermediate value between V_{DD} and V_{SS}. The value perceived at Z is determined by the logic threshold of the stage accepting it as input. Bridging faults are often difficult to detect. We use a slightly more complicated logic function to demonstrate this problem. Figure 7.2 shows a CMOS implementation of $Z = AB + CD$. A bridging between nodes 1 and 2 causes the Z output to be intermediate value as 1, the circuit continues to function as expected with a substantially smaller noise margin. Otherwise, if Z is below the logic threshold of the next stage, the bridging fault has converted the function to $Z = \overline{A(B + D) + D(B + C)}$. Table 7.2 compares the NAND gate (Fig 7.1) output at a few example faulty conditions with their corresponding fault-free outputs. The last column ($\boldsymbol{Q}_{\mathbf{nA}}$: bridged) depicts the output when the source and drain of transistor $\boldsymbol{Q}_{\mathbf{nA}}$ are shorted. In the next section, we will show that test patterns can be generated by comparing the faulty and fault-free outputs of a circuit.

7.2 □ TEST GENERATION (STUCK-AT FAULTS)

Test generation deals with the selection of input combinations that can be used to verify the correct operation of a chip. Consider a circuit with a function $F(X) = F(x_1 \ldots\ldots x_n)$, where X is an input vector representing n inputs $x_1 \ldots\ldots x_n$. Suppose we would like to find a test pattern to detect stuck-at fault occurring at an internal circuit node k(i.e., $K \notin X$). The first observation is that-node k must be set to 0 in order to detect k: s-a-1 or 1 to detect k: s-a-0. The second observation is that a test pattern X_k qualified to detect the specific k stuck-at fault must satisfy the following condition. When X_k is applied to the circuit, the fault-free response $F(X_k)$ must be different from the incorrect output $F(X_k)$ caused by the stuck-at fault at k. This is the basic principle of test generation.

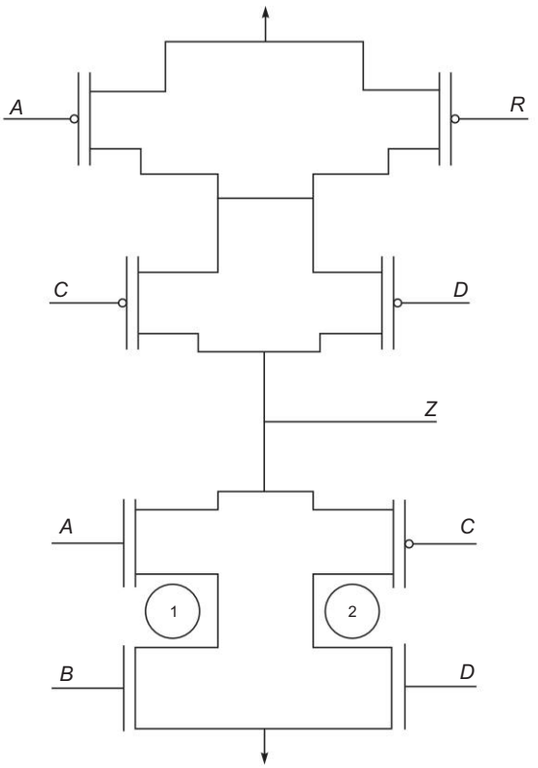


Fig 7.2 CMOS implementation of $Z = \overline{AB + CD}$

Table 7.2 A few possible faults in a CMOS NAND gate

AB	*Z* (fault-free)	*Z* (*A*: s-a-1)	*Z* (*A*: s-a-0)	*Z* (Q_{nA} : s-op)	*Z* (Q_{nA} : bridged)
00	1	**1**	**1**	**1**	**1**
01	1	**0**	**1**	**1**	***X***
10	1	**1**	**1**	**1**	**1**
11	0	**0**	**1**	**HiZ**	**0**

Normally, it is impossible to directly inject a value at an internal node of a chip. It is thus necessary to find an input combination X_K that can set K to the desired value. If we can set the value of a node of a chip, either directly in the case of an input node, or indirectly in the case of an internal node, the node is said to be controllable.

Unlike a board-based design, it is impartial to physically probe the internal nodes of a chip for their values. In order to observe an internal node, some path must be chosen to sensitize a path from the node under test to an observable output. If the value of a node can be determined, either directly in the case of an output, or indirectly in the case of an internal node, it is said to be observable.

Now we formalize the requirement of a test pattern that detects a stuck-at fault at an input X_i. X_i is a test vector for detecting X_i. *S-a*-1 if and only if

$$\bar{x}_i \bullet (F(X_i) \oplus (F^{/}(X_i)) = 1 \tag{7.1}$$

and a test vector for detecting x_i: *s-a*-0 if and only if

$$x_i \bullet (F(X_i) \oplus (F^{/}(X_i)) = 1 \tag{7.2}$$

where $F(X_i) = F(x_{1,\ldots..}, x_{1\ldots\ldots..}, x_n)$ and $F(X_i) = F(x_{1,\ldots..}, x_{1\ldots\ldots..}, x_n)$.

In Eq. (7.1), the term $\bar{x}_i$ ensures that x_i is set to 0. Similarly in (7.2), the term x_i ensures that x_i is set to 1. The exclusive or term used in (7.1) and (7.2) are called the Boolean difference of $F(X)$ with respect to its input x_i and can be written as

$$\frac{dF(X)}{dx_i} = F(x_1,\ldots\ldots\ldots, x_i,\ldots\ldots\ldots x_1,) \oplus F(x_1,\ldots\ldots\ldots, \bar{x}_i,\ldots\ldots\ldots x_n) \tag{7.3}$$

which specifies that the variables other than x_i must be assigned values so that the output is sensitive to a change of x_i.

The principles specified in (7.1) and (7.2) can be generalized to specify test patterns for an internal node of a combinational circuit. This can be easily done by rewriting $F(x_i,\ldots..x_n)$ as $F(x_1,\ldots\ldots\ldots,x_n, k)$ in which k is the internal node for which a test pattern is to be determined. The test pattern requirements are then generalized as follows.

X_k is a test pattern for detecting *k*:*s-a*-1 if and only if

$$\bar{K}\frac{dF(X)}{dk} = 1 \tag{7.4}$$

And a test pattern for detecting *k*:*s-a*-0 if and only if

$$\bar{K}\frac{dF(X)}{dk} = 1 \tag{7.5}$$

As an example, consider a logic function $F = x_1x_2 + x_3x_4$. Assume that $k = x_1x_2$ is an internal node of the circuit. We can rewrite the function as $F = k + x_3x_4$ and $k = x_1x_2$. The tests for *k*:*s-a*-1 are found by considering.

$$\bar{K}\frac{dF}{dk} = \overline{x_1x_2}((K + x_3x_4) \oplus (\bar{K} + x_3x_4)) \tag{7.6}$$

$$= \overline{x_1x_2}\ \overline{(x_3x_4)}$$

$$= \bar{x}_1\bar{x}_3 + \bar{x}_2\bar{x}_3 + \bar{x}_1\bar{x}_4 + \bar{x}_2\bar{x}_4 \tag{7.7}$$

$$= 1$$

The following test patterns, $x_1x_2x_3x_4 = 0 - 0,\text{-}00\text{-},0\text{-}0,\text{-}0\text{-}0$, in which the '–' indicates a don't care value, satisfy (7.6) and are thus the tests for *k*:*s-a*-1.

$$K\frac{dF}{dk} = x_1x_2((K + x_3x_4) \oplus (\bar{K} + x_3x_4))$$

$$= x_1x_2\overline{(x_3x_4)}$$

$$= x_1x_2x_3 + x_1x_2\bar{x}_4$$

$$= 1$$

An analysis of (7.7) yields test patterns $x_1x_2x_3x_4 =$ 110- and 11 -0 for *k*: s-a-0.

The above test generation principles have been implemented in various approaches. All these approaches are based on the assumption that the circuit–under–test is nonredundant and has at most a

single stuck-at fault. The single-fault assumption may be justifiable for a fully debugged chip coming out of a production line. This assumption does not apply to a prototype chip which may have more than one fault caused by design errors or fabrication defects. However, most automatic test-pattern-generation algorithms still adopt the single-fault assumption since the determination of test patterns can be significantly simplified. In practice, many multiple faults will also be detected by a test set generated under the single-fault assumption. With the exception of stuck-open faults, a test set is generated by faults. Faults that are not detected in a fault simulation can be considered individually so that their test patterns can be generated to enhance the test set.

7.3 □ PATH SENSITIZATION

Test generation involves two steps: fault activation and error propagation. **Fault activation** requires setting the circuit primary inputs so that a *s-a-v* line has a value $\overline{u}$. **Error propagation** seeks primary input values to propagate the resulting error to a primary output. Path sensitization is a direct implementation of (7.4) and (7.5). If the fault locates at an internal node of the circuit, a difference at the node being tested must be created. For example, a test vector that attempts to detect *k*:s-a-0 must set *k* to 1. A sensitized path must be found to propagate the difference from its origin to the output. The necessary conditions to create the difference at the tested node and to propagate the fault along the sensitized path are then established.

Path sensitization can be applied as a manual approach to identify test vectors for small circuits. The next section explains a computer-aided test-generation algorithm that implements the concept of path sensitization.

7.4 □ *D*-ALGORITHM

The *D*-algorithm is the pioneer of many computer-aided test-generation methods. The *D*-algorithm uses symbols D and $\overline{D}$ to represent errors. If we use D to denote a 0/1 error (0 is the expected value and 1 is the observed value) then $\overline{D}$ denotes a 1/0 error (1 is expected value and 0 is the observed value). The meaning of D and $\overline{D}$ can be exchanged as long as their uses are consistent throughout a chip-under-test. Error-free values 0/0 and 1/1 are simply denoted by 0 and 1, respectively. Adding an unspecified (don't care) value X, *D*-algorithm performs test generation by carrying out 5-valued logic operations in the chip-under-test. The 5-valued logic operations are shown in Table 7.3.

Table 7.3 5-valued logic operations in *D*-algorithm.

AND	0	1	D	$\overline{D}$	X
0	0	0	0	0	0
1	0	1	D	$\overline{D}$	X
D	0	D	D	0	X
$\overline{D}$	0	$\overline{D}$	0	$\overline{D}$	X
X	0	X	X	X	X

OR	0	1	D	$\overline{D}$	X
0	0	1	D	$\overline{D}$	X
1	1	1	1	1	1
D	D	1	D	1	X
$\overline{D}$	$\overline{D}$	1	1	$\overline{D}$	X
X	X	1	X	X	X

Consider the problem of generating a test of c: s-a-0 in the 2-input gates shown in Fig 7.3. The behavior of this faulty NAND gate is represented by the truth table of Fig. 7.4, in which the X's indicate don't care values. This truth table simply says that output c remains 0 regardless of the values of a and b.

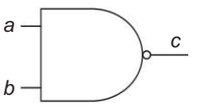


Fig 7.3 2-input NAND gate

In order to detect c: s-a-0, we need to set c at 1 to create a D (or $\overline{D}$, as long as it is consistent throughout the circuit). The input pattern (ab) can be easily determined by selecting one from the NAND gate's fault-free truth table that produces $c = 1$. Three patterns (ab = 00, 01, and 10) are possible.

Table 7.4 Truth table of a NAND gate with its output s-a-0.

a	*b*	*c:s-t-0*
X	X	0

Compact truth tables, called **singular covers**, are used in the D-algorithm. The truth table of a logic gate can be simplified by incorporating the don't care value (X). A singular cover of a logic gate can be generated by inspecting any two rows in the original truth table with identical outputs. In this inspection, any input on which the output does not depend is marked as a don't care (X). The results of these inspections are collected to form the gate's singular cover. Table 7.5 shows the singular cover for a two-input NAND gate. According to the singular cover of a two-input NAND gate, the input patterns that set $c = 1$ are $0X$ and $X0$.

Table 7.5 Singular cover for two-input NAND gate

a	*b*	$\overline{ab}$
0	X	1
X	0	1
1	1	0

A pattern formed by an input combination of a logic circuit and the logic circuit's response to this input combination is called a cube. For example, the rows (0*X*1, *X*01, 110, 001, etc.) in the singular covers shown in Table 7.5 are cubes. A primitive *D*-cube of a fault is a cube that brings the effect of a fault to the output of the logic circuit. It is used to generate a difference (i.e., *D*) at the faulty node to be tested.

In the ongoing example of determining a test pattern for *c*:*s*-*a*-0 (Fig. 7.3), if we set the inputs of the NAND gate to *ab* = 0*X* or *X*0, *c* = *D*. The primitive *D*-cubes for *c*:*s*-*a*-0 are thus 0*XD* and *X*0*D*.

A primitive *D*-cube for a logic function can be constructed by selecting one cube from the fault-free singular and one cube from the singular cover of the faulty circuit, which should have different output values. These two cubes are then intersected according to the intersecting rules given in Table 7.6, which describes the result of intersecting two values in corresponding positions of two cubes.

Table 7.6 Intersecting rules

Intersect (Λ)	0	1	*X*
0	0	$\bar{D}$	0
1	*D*	1	1
X	0	1	*X*

Apply the intersection operation to ***XX*0** (a cube from the faulty NAND gate with *c*:*s*-*s*-0, see Table 7.4), 0X1 (a cube from the faulty NAND gate with *c*:*s*-*a*-0 see Table 7.4), and 0*XI* (a cube from the fault-free NAND gate, see Table 7.5). We have primitive *D*-cube 0*X* $\bar{D}$ (or X0D). Similarly, intersecting *XX0* and *X01* produces primitive *D* cube *X*0*D* (or *X*0*D).* This result is consistent with the one found by observation.

A primitive *D*-cube can also be found for a faulty input of a logic function. We would like to find a primitive *D*-cube for *b*:*s*-*a*-0 for the 2-input NOR gate (Fig. 7.4) as shown in Table 7.7. The singular cover of the fault-free NOR gate is shown in Table 7.8.

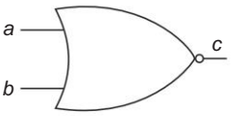


Fig 7.4 Two-input NOR gate

Table 7.7 Singular cover for NOR gate with *b*: s-a-0.

a	*b*	*c*
0	*X*	1
1	*X*	0

Table 7.8 Singular cover for fault-free NOR gate

A	*b*	*c*
0	0	1
1	*X*	0
X	1	0

The primitive *D*-cube for *b*: *s-a*-0 is generated as follows. Since *b*: *s-a*-0, it must be set to 1 to create a difference at *b*. Cube *X*10 fits with this description and is selected. It is then intersected with cube 0*X*1 from the faulty gate's singular cover. This produces the primitive *D*-cube 01*D*.

The *D*-algorithm uses propagation *D*-cubes to sensitize a path which propagates the difference *D* or $\bar{D}$ caused by a fault to a primitive output. Propagation *D*-cubes can be found by inspecting a gate's singular cover. All cubes that cause the output to depend only on one or more of its inputs are propagation *D*-cubes.

The propagation *D*-cubes of a logic function can be systematically constructed by intersecting cubes with different output values in its singular cover. For example, the propagation *D*-cubes of a two-input NAND gate (Fig. 7.3) is $abc = 1D\,\bar{D}$, $D1\,\bar{D}$ and $DD\,\bar{D}$.

The use of *D*-algorithm to determine test patterns follows the steps shown below.

1. Select a primitive *D*-cube for the fault of which test vectors are to be determined.
2. Select propagation *D*-cubes from the logic gates in the path from the faulty node to the output. This allows the difference (*D* or $\bar{D}$) to be propagated to the output so that it can be observed. This is called the **forward trace operation**.
3. For all other logic blocks that are not involved with the sensitized path, try to match the cubes in their singular cover with the values determined so far. A consistent set of input values is the valid test vector. If a consistent set of input values cannot be found, no test vector can be found for this fault (e.g., the circuit is redundant).

An example is used here to demonstrate the use of *D*-algorithm to identity test vectors.

Example 7.1 *Use the D-algorithm to generate test patterns for g: s-a-1 in the circuit shown in Fig. 7.5*

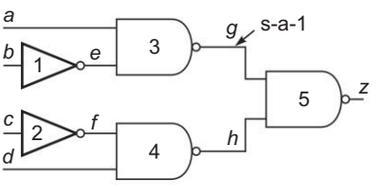


Fig 7.5 Sample circuit for *D*-algorithm

Solution: The signal line *g* is the output of a two-input NAND gate (gate 3). The primitive *D*-cube for gate 3 is thus selected to be *aeg* = 11*D*. The *D* at *g* must be propagated to the primary output *Z* through gate 5. Gate 5 has propagation *D*-cubes $ghZ = 1D\bar{D}$, $D1\,\bar{D}$, etc. We select $D1\,\bar{D}$ as the propagation *D*-cube of gate 5 to match with the primitive *D*-cube of gate 3. The rest of the signals are selected from the singular covers of gates 1, 2, 4 to be consistent with the signals determined so far. The steps of the

D-algorithm are shown in Table 7.9. Notice that the selection of a cube in each step must be consistent with the values selected in previous steps. The test patterns are found to be 101*X* (i.e., 1010 and 1011). Other test patterns can be found by selecting a different singular cover cube for gate 4.

Table 7.9 *D*-algorithm

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*z*
Primitive *D*-cube (gate 3)	1				1		*D*		
Propagation *D*-cube (gate 5)							*D*	1	$\bar{D}$
Singular cover (gate 1)		0			1				
Singular cover (gate 4)				*X*		0		1	
Singular cover (gate 2)			1			0			

7.5 □ TEST GENERATION FOR OTHER FAULT MODELS

1. Stuck-Open Faults

Recall that a stuck-open fault transforms a CMOS combinational circuit into a sequential circuit. In order to detect a stuck-open fault, the observable node must be first driven to a known initial value. Consider finding the test sequence for Q_{na}: s-op in the NAND gate in Fig 7.1. Setting AB = 00, 01 or 10 will drive output Z to an initial value of 1 when a second test vector of AB = 11 is applied. A fault-free circuit produces Z = 0 in response to this test sequence. On the other hand, Z = 1 when Q_{na} *s*-op.

2. Bridging Faults

When two normally unconnected signal lines are shorted, we have a bridging fault. A general model for a bridging fault between two lines *a* and *b* is shown in Fig. 7.6. Once a bridging fault occurs between signals *a* and *b*, these values become unobservable. We consider the values of *a* and *b* in the model—their driven values, not observed values.

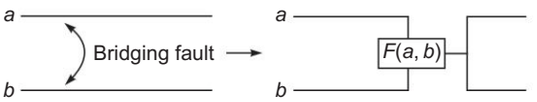


Fig 7.6 Bridging fault model

If *a* and *b* are identical, the function $F(a, b)$ assumes the same value. When *a* and *b* have opposite values, the value $F(a, b)$ is indeterminate.[4] This situation can be verified by considering two inverters with their outputs tied together. Indeterminate signal values are very difficult to detect since its value may depend on the following stage's logic threshold. If there exists at least one path between the

bridged lines, the short causes a feedback bridging fault. A combination circuit can be converted into a sequential circuit by the presence of a feedback bridging fault and thus requires a test sequence to detect the fault.

However, it is easy to show that a bridged signal driven by opposite signals causes an abnormal current to flow through the circuit, which can be detected by a current-based test.

7.6 ☐ TEST GENERATION EXAMPLE

The test is to generate a set of test patterns for the full adder shown in Fig 7.7

Fig 7.7 Full adder

The full-adder circuit has three inputs (*a*, *b* and *c*) and two outputs (*d* and *e*). Outputs *d* and *e* are its carry and sum outputs, respectively. Since it has three inputs, it can be exhaustively tested by all eight possible input combinations from 000 to 111. Assume that the single stuck-at fault model is used to determine a set of test patterns.

- *a*: s-a-1
- *b*: s-a-1
- *c*: s-a-1
- *d*: s-a-1
- *e*: s-a-1
- *a*: s-a-0
- *b*: s-a-0
- *c*: s-a-0
- *d*: s-a-0
- *e*: s-a-0

Table 7.10 lists all test patterns for each of these stuck-at faults. The fault coverage of each test pattern is summarized in the fault matrix shown in Table 7.11. Inspecting the fault matrix reveals that we only need two patterns *abc*-000 and 111 to detect any single stuck-at faults at the inputs and outputs of the full adder.

It is a big reduction of test vectors obtained according to the single stuck-at fault model. But, how well does this test do when it is used in practice? For the sake of this example, we assume that the full adder is implemented by the circuit shown in Fig 7.8, which is a typical standard-cell implementation of the full adder.

Signal *m* is not directly accessible. We will determine a test vector to detect *m*: *s*-*a*-0. We need a test vector that will set *m* to 1, an opposite value of its stuck-at fault. Input vector 000 will do that. Since the signal *m* is not directly observable, it must be propagated to either output *d* or *e*. In order to reflect any change of signal *m* at output *d*, the remaining two inputs of a NAND gate must be set to 1, which is also achieved by the test vector 000. In other words, vector 000 detects this internal fault *m*: *s*-*a*-0.

In order to detect a fault *m*: *s*-*a*-1, *m* must be set to 0. Vector 111 satisfies this requirement. However, it does not provide the necessary values on the other inputs of the NAND gate to propagate the change at *m* to output *d*. So the fault *m*: *s*-*a*-1 is not detectable by the test vectors determined by considering single stuck-at faults at the inputs and outputs of the full adder. The detection of *m*: *s*-*a*-1 requires the inputs of the NAND gate that produces *d* to be 011.

The above example demonstrates the attempt to identify a minimum number of tests to verify the correctness of a chip. First, a number of critical nodes are selected to generate test vectors. This produces a set of test vectors. A process called **fault simulation** is then performed to evaluate the fault coverage of this test set. Faults not considered by the initial fault model are injected in to the circuit simulated by a circuit simulator. We call this a **fault simulation**. The test vectors are applied to the simulated faulty circuit to determine if the fault introduced can be detected by at least one of then. If it does, the fault is covered. Otherwise, new test vectors can be added to enhance the fault coverage, which is defined to indicate the percentage of faults that are detected by the test vectors.

Table 7.10 Test patterns for all stuck-at faults in a full adder

Fault	Test Patterns (*abc*)	Fault-free Output (*de*)	Faulty output (*de*)
a: s-a-1	000	00	01
	001	01	10
	010	01	10
	011	10	11
b: s-a-1	000	00	01
	001	01	10
	100	01	10
	101	10	11
c: s-a-1	000	00	01
	010	01	10
	100	01	10
	110	10	11

Contd.

a: s-a-1	111	11	10
	100	01	00
	101	10	01
	110	10	01
b: s-a-1	111	11	10
	010	01	00
	011	10	01
	110	10	01
c: s-a-1	111	11	10
	001	01	00
	011	10	01
	101	10	01
d: s-a-1	000	00	10
	001	01	11
	010	01	11
	100	01	11
e: s-a-1	110	10	11
	000	00	01
d: s-a-1	011	10	00
	101	10	00
	110	10	00
	111	11	01
e: s-a-1	001	01	00
	010	01	00
	100	01	00
	111	11	10

Table 7.11 Fault matrix

Test vector	*a* *s-a-1*	*b* *s-a-1*	*c* *s-a-1*	*a* *s-a-0*	*b* *s-a-0*	*c* *s-a-0*	*d* *s-a-1*	*e* *s-a-1*	*d* *s-a-0*	*e* *s-a-0*
000	1	1	1				1	1		
001	1	1				1		1	1	
010	1		1		1			1	1	
011	1				1	1				
100		1	1	1				1	1	
101		1		1		1				1
110			1	1	1		1			1
111				1	1	1			1	1

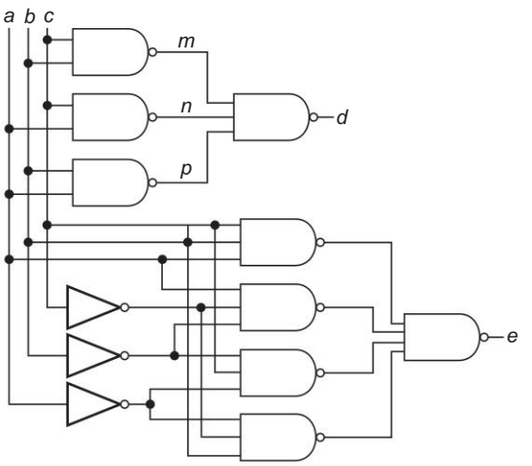


Fig 7.8 A gate-level implementation of a full adder

7.7 □ SEQUENTIAL CIRCUIT TESTING

Testing sequential circuits is difficult because their behaviors depend not only on present input values but also on past inputs. Conceptually, a sequential circuit can be modeled as a sequence of identical combinational circuits. Techniques developed for combinational circuit test generation can then be applied. This approach is illustrated in Fig 7.9. We represent the sequential circuit with n identical combinational circuits. The i^{th} combinational circuit receives input $x(i)$ and state $y(i\text{-}1)$. The output $z(i)$ is observable. Therefore, the i^{th} combinational circuit corresponds to the sequential circuit at the i^{th} clock cycle.

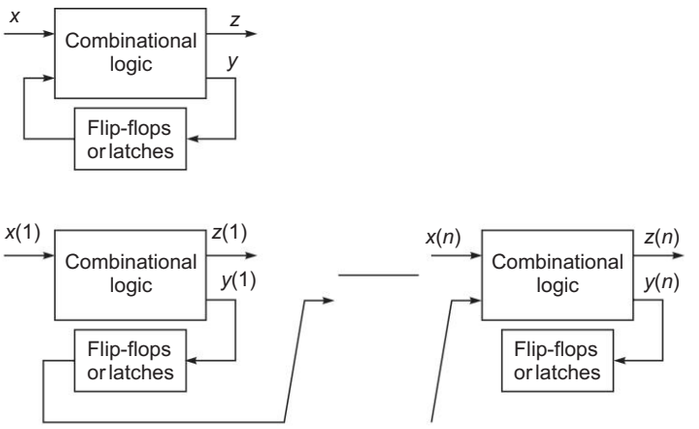


Fig 7.9 Sequential circuit modeled as combinational circuit for test generation

A fault occurring in the original sequential circuit transforms into *n* identical faults in the combinational circuit model; so it has to be treated as a multifault-detection problem. This technique is thus only realizable for sequential circuits with a few states.

Techniques have been developed to simplify the testing of sequential circuits by increasing testability (i.e., controllability and observability). The next section describes a number of design-for-testability approaches

7.8 □ DESIGN-FOR-TESTABILITY

A VLSI chip naturally has limited controllability and observability. One principle in which all IC designers agree is that a design must be made testable by providing adequate controllability and observability. These properties must be well planned for in the design phase of the chip and not as an afterthought. This practice is referred to as Design-For-Testability (DFT).

The testability of a circuit can be improved by increasing its controllability and obsevability. For example, the test of a sequential circuit can be significantly simplified if its state is controllable and observability. If we make the registers storing the state values control points, the controllability of the combinational logic's "hidden" inputs is improved. On the other hand, if we make the flip-flops observation points, the obsevability of the combinational logic's "hidden" outputs is increased.

This is usually done by modifying the registers so that they double as test points. In a test mode, the registers can be reconfigured to form a scan register (i.e., shift register). This allows test patterns to be scanned in as well as responses to be scanned out. A single long scan register may cause a long test time since it takes time to scan value in and out. In this case, multiple scan registers can be formed so that different parts of the circuits can be tested concurrently. Even though a scan-based approach is normally applied to the registers required in the function, additional registers can be added solely for the purpose of DFT.

IEEE has developed a standard (IEEE Std. 1149.1) for specifying how circuitry may be built into an integrated circuit to provide testability. The circuitry provides a standard interface through which

communication of instruction and test data are done. This is called the IEEE Standard Test Access Port and Boundary-Scan Architecture.

Another problem of a sequential circuit testing is that we need to bring the circuit into a known state. If the initialization (i.e., reset) of a circuit fails, it is very difficult to test the circuit. Therefore, an easy and foolproof way to initialize a sequential circuit is a necessary condition for testability. The scan-based test point DFT approach allows registers to be initialized by scanning in a value.

If a circuit incorporates free-running clock generators or pulse generators, it is extremely hard to test. A solution is to provide a means to turn off these circuits and provide the necessary signals externally.

A number of other DFT techniques are also possible. These include the inclusion of switches to disconnect feedback paths and the partitioning of large combination circuits into small circuits. Remember the cost of testing a circuit goes up exponentially with its number of inputs. For example, partitioning a circuit with 100 inputs into 2 circuits, each of which has 50 inputs, can reduce the size of its test pattern space from 2^{100} to 2^{51} (2×2^{50})

Most DFT techniques usually require additional hardware to be included to the design. This modification affects the performances of the chip. For example, the area power, number of pins and delay time are increased by the implementation of a scan based design. A more subtle point is that DFT increases the chip area and logic complexity, which may reduce the yield. A careful balance between the amount of testability and its penalty on performance must be applied.

7.9 □ BUILT-IN SELF-TEST

Built-in Self-Test (BIST) is a concept that a chip can be provided with the capability to test itself. There are several ways to accomplish this objective. One way is that the chip tests itself during normal operation. In other words, there is no need to place the chip under test into a special test mode. We call this the on-line BIST. We can further divide on-line BIST into concurrent on-line BIST and non-concurrent on-line BIST. Concurrent on-line BIST performs the test simultaneously with normal functional operation. This is usually accomplished with coding techniques (e.g., parity check). Nonconcurrent BIST performs the test when the chip is idle.

Off-line BIST tests the chip when it is placed in a test mode. An on-chip pattern generator and a response analyzer can be incorporated into the chip to eliminate the need for external test equipment. We discuss a few components that are used to perform off-line BIST below.

Test patterns developed for a chip can be stored on the chip for BIST purposes. However, the storage of a large set of test patterns increases the chip area significantly and is impractical. A pseudo-random test is carried out instead. In a pseudo-random test, pseudo-random numbers are applied to the circuit under test as test patterns and the responses compared to expected values. A pseudo-random sequence is a sequence of numbers that is characteristically very similar to random numbers. However, pseudo-random numbers are generated mathematically and are deterministic. This way the expected responses of the chip to these patterns can be predetermined and stored on chip. We discuss the structure of a linear feedback shift register shortly, which can be used to generate a sequence of pseudo-random numbers.

The storage of the chip's correct responses to pseudo-random numbers also has to be avoided for the same reason of avoiding the storage of test patterns. An approach called signature analysis was developed for this purpose. A component called a signature register can be used to compress all responses

into a single vector (signature) so that the comparison can be done easily. Signature registers are also based on linear feedback shift registers.

7.9.1 Linear Feedback Shift Register

Linear Feedback Shift Register (LFSR) are used in BIST both as a generator of pseudo-random patterns and as a compressor of responses. Figure 7.10 shows signature analyzer consisting of the feedback shift register, which illustrates the sequence it generates. Each box represents a flip-flop. The flip-flops are synchronized by a common clock and form a rotating shift register. Assume that the initial value in the shift register is 110. It is shown that the shift register goes through a 3-pattern sequence—110-011-101. The sequence repeats afterward.

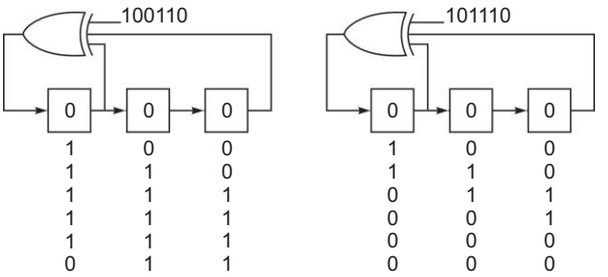


Fig 7.10 Signature analyzer

When a sequence of n bits is encoded by m-bit signature ($m < n$), more than one sequence will map into one signature. There are 2^n unique sequences and 2^m unique signatures in this situation. In average, each signature will represent $2^n/2^m = 2^{n-m}$ sequences. The probability of declaring an incorrect sequence correct since it produces the expected signature is

$$\frac{2^{n-m}-1}{2^n-1} \tag{7.8}$$

The denominator in (7.9) is the number of incorrect sequences. The numerator is the number of incorrect sequences that would map into the signature identical with that of the correct sequence. Normally $n >> m > 1$; so (7.8) can be approximated as

$$\frac{2^{n-m}-1}{2^n} = 2^{-m} \tag{7.9}$$

The probability of drawing an incorrect conclusion from using a signature analyzer can then be made arbitrarily small by choosing a large m. Normally, $m = 16$ would give an acceptable error probability. When the signature is incorrect, the circuit is not functioning properly. If the signature is correct, we can only conclude that the circuit has a high probability to be functioning correctly.

Multiple data sequences can be combined and compressed with a signature analyzer with multiple inputs to produce a multiple-input signature. We conclude this section with Fig 7.11, which shows the use of a pseudo-random pattern generator and a signature analyzer to test a circuit.

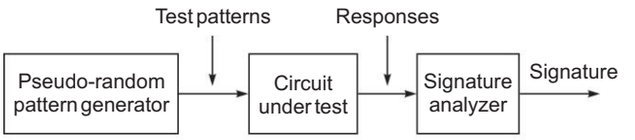


Fig 7.11 Set-up of a pseudo-random pattern generator and a signature analyzer to test a circuit

7.9.2 Finite State Machine Approach for BIST

Flip-flops have two main uses within circuits. Firstly, they are used to store logic values (or, more commonly, a group of flip-flops is used as a data register to store a logic word) for use at some later stage in the process. In this kind of application, testing will often be relatively straightforward, since the inputs and outputs are likely to be reasonably accessible and the relationship between input and output is uncomplicated. The other main use for flip-flops is as the central components in Finite State Machines (FSMs). An FSM is used to control the execution of a sequence of operations; this is achieved by making each operation depend on a state of the FSM, where a state is defined as a particular set of values held in its flip-flops. The FSM changes state under the control of a clock, but the particular sequence of states that it passes through is defined by the signals applied to the inputs of the flip-flops. These signals will be generated by a clock of combinational circuitry—the next-state logic. If the next state depends only on the present state, the FSM has no external inputs (apart from the clock). It produces a fixed sequence of states and is known as an autonomous FSM. In general, however, an FSM can have external inputs which modify its behavior so that the state transition at any time is a function both of the present state and of the external inputs. Such a machine can be represented as in Fig. 7.12, which shows an FSM with two flip-flops, X and Y, and two external inputs, A and B. If the flip-flops are, for the sake of argument, D type, then the next state logic has to produce flip-flop input signals D_x and D_y as functions of A, B, X and y. The requirements for this logic can be expressed in terms of a state transition diagram, an example of which is shown in Fig. 7.13(a). There are several features of this diagram that have a bearing on testing activities and problems.

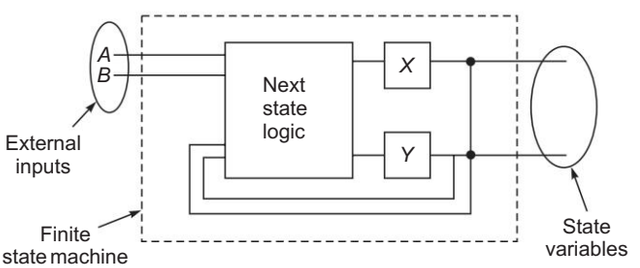


Fig. 7.12 A FSM model

1. An FSM with n flip-flops and m external inputs will contain 2^n states and 2^m transitions per state. Representing all these states and transitions quickly becomes unwieldy to the point of incomprehensibility as the circuit increases in size; this is equally true whether diagrammatic or tabular methods are used.

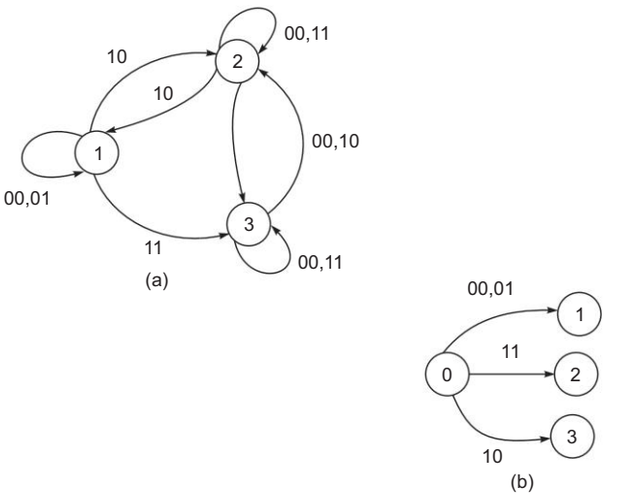


Fig. 7.13 State-transition diagram: (a) System requirement for all combination of inputs and outputs (b) Fourth state with transitions

2. When designing an FSM, the state-transition diagram will be derived from the specification; in particular, the number of states required depends on the application, and can take any integral value. When it comes to implementation, the number of flip-flops in the FSM must be chosen so that there are enough states available; this will often mean that the design contains redundant states. In Fig. 7.13(a), for example, three states are specified; the FSM must, therefore, contain two flip-flops, which means that four states will actually exist. An implementation of the FSM of Fig. 7.13(a) is shown in Fig. 7.14. In deriving this, transitions from state 0 are entered as 'don't cares'. However, once the circuit has been implemented then the logic that is designed to produce the required transitions among the 'working' states will also of necessity define transitions from the redundant state—these are shown in Fig. 7.13(b).

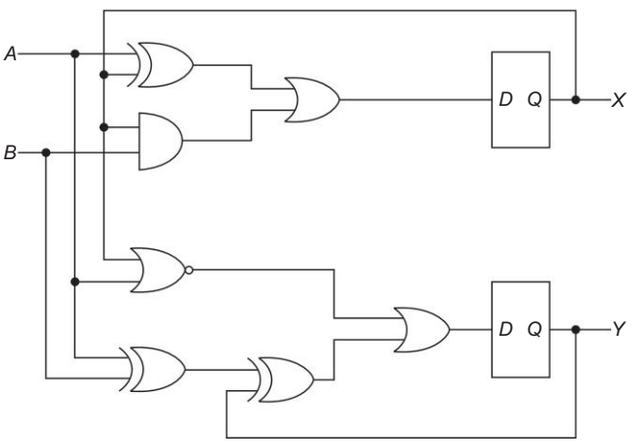


Fig. 7.14 Implementation of FSM as shown in Fig. 7.13

3. The state-transition diagram gives no indication of how the circuit will behave when first switched on. In fact, unless special measures are taken, the circuit can settle entirely unpredictably into any one of the possible states, including any of the redundant states. This indeterminacy has to be allowed for both in the functional design and when testing.

   Working from the circuit diagram, even for the very simple example of Fig. 7.13, the function of the circuit is far from clear. In particular, there is no way in which the working states can be distinguished from redundant ones. Testing has to be developed largely on a structural basis using the circuit diagram. The only real alternative would be a hybrid approach, treating individual flip-flops on a functional basis (checking that they can make each possible transition) while using structural methods for the 'glue' logic.

7.9.3 Embedded State Machines

The discussion so far has centred on FSMs whose state variables have been assumed to be all observable. The problems posed by these circuits are further increased if the FSM is embedded within further blocks of logic so that its behavior can only be inferred by observation of output values. An example of an embedded FSM is shown in Fig. 7.15, which represents an autonomous FSM whose state variables provide the inputs to a block of output logic which forms the single output variable W. This circuit is a sequencer, generating the repetitive waveform shown in Fig. 7.16. The waveform can be seen to have a period of five clock cycles, and can, therefore, be generated by a five-state FSM. The output is required to be high during states 3 and 4, with states 0, 6, and 7 being redundant. Using these redundancies, we can form the function W as

$$W = YZ + \bar{Y}.\bar{Z}$$

The existence of combinational logic between the state variables and the primary outputs can have a number of consequences:

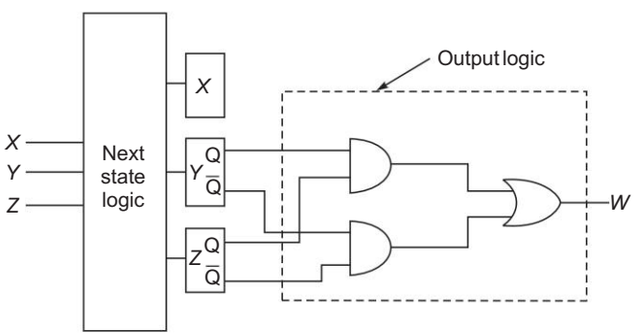


Fig. 7.15 A typical waveform generator having an FSM with logic output

1. The fault cover for any particular test will probably be reduced.
2. To establish a sensitive path through the output logic will require a particular state, which may require a sequence of input patterns.
3. Some faults may well become untestable.

It is worth noticing, in the circuit of Fig. 7.15, that in order to verify the behavior of X, a whole succession of actions would need to be followed:

1. Establish the appropriate conditions for X (input signals and prior state) to exercise the chosen facet of behavior. This will, in general, require a sequence of input patterns.
2. Propagate the fault effect to Y and Z using a further sequence of patterns.
3. Hope that the fault effect will propagate from Y and Z to W (since there is no way exercising further control).

It is not surprising that ATPG systems find great difficulty in generating test sequences for general sequential circuits and that the need for making concessions for testability is being increasingly recognized.

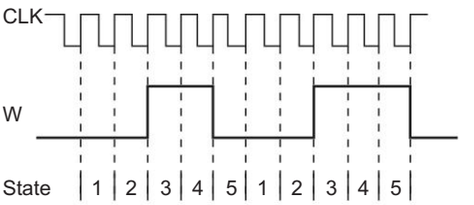


Fig. 7.16 Waveform generated by circuit in Fig. 7.15

7.10 □ ENHANCING TESTABILITY

Any form of testing must consist of two elements—test conditions first to be set up, and then the result of the test to be observed. Figure 7.17 shows representation of testing problem in terms of controllability and observalibity. In order to sensitize a particular fault, it is necessary to establish the appropriate fault-free value at the node of interest by manipulation of some of the PIs. Clearly, this operation of controlling the value at the node can be more or less difficult depending on the circuitry (the 'control logic') between the PIs and the node. The second stage in the testing process is to use further manipulation of PI values so that fault effects are propagated to the POs. Testability enhancement comes down to increasing the controllability or the operability (or both) of internal nodes in the circuit.

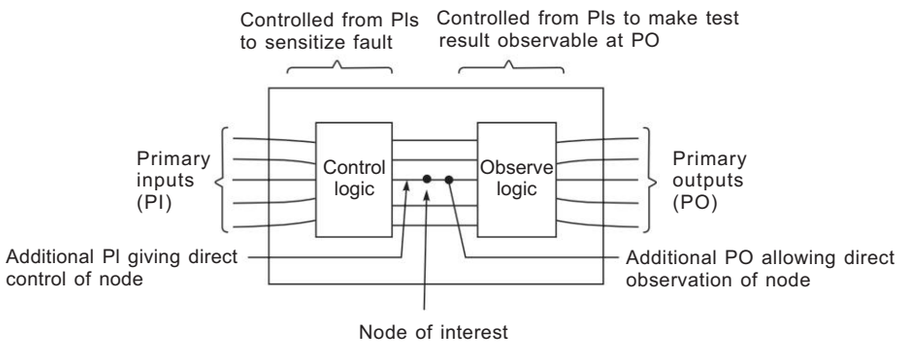


Fig. 7.17 Testing problem in terms of controllability and observalibity

The most direct way of enhancing testability, as indicated in Fig. 7.18, is to connect additional PIs or points to 'difficult' nodes. The extent to which this approach is possible will, in practice, be severely

limited by the availability of I/O pins. These are never plentiful, and at chip level, it is always going to be difficult to secure more than a very few pins dedicated to testing functions. Silicon area is much more likely to be available, so that all DFT schemes engage to a greater or lesser extent in circuit arrangements that trade silicon area for pin requirements.

Test conditions within a circuit can be set up using a single dedicated pin by building in a shift register as shown in Fig. 7.18. Here two PIs are each given a dual function by using a demultiplexer controlled by the dedicated test signal *C*. These two lines are connected to the clock and data inputs of the shift register (a third line could be used to allow a master reset for the shift register, but it is not strictly necessary). The shift register, which can be of any length, can now be loaded serially with test data that can be used to control other parts of the circuit, while, during the subsequent test, the PIs can revert to their normal functions. A single external signal *C* can, by this means, be used to provide any number of control signals (and can indeed control any number of shift registers), the economy in pins being paid for by the need to set up the required test conditions serially.

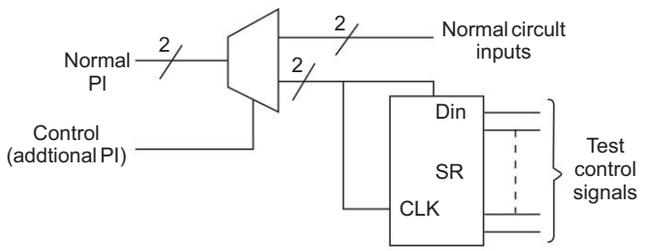


Fig. 7.18 A shift register to provide the control signals to the interior of the circuit

It is often advantageous during testing to be able to break a connection through which one element drives another, and to allow the tester to provide the drive directly. This can be done using a degating circuit as symbolized in Fig. 7.19(a); one method of implementing the circuit is shown in Fig. 7.19(b). By pulling INH low, the data pathway is broken and the data out is controlled directly by DR. With both INH and DR high (or open circuit, with the pull-up resistors in place) the normal data pathway is complete.

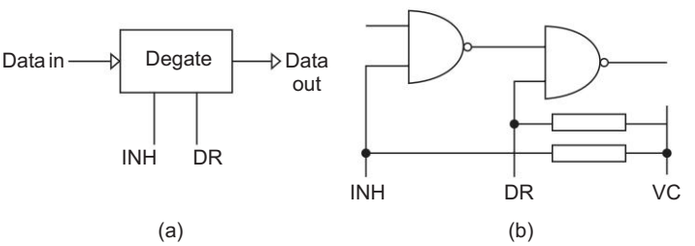


Fig. 7.19 A degating circuit allows a signal path to be broken and to be controlled externally.

(a) A symbol for a degating circuit. With no signal applied to the control inputs, data passes straight through to the output.

(b) One way of implementing a degating circuit. With the control inputs left open-circuit, the signal path is closed.

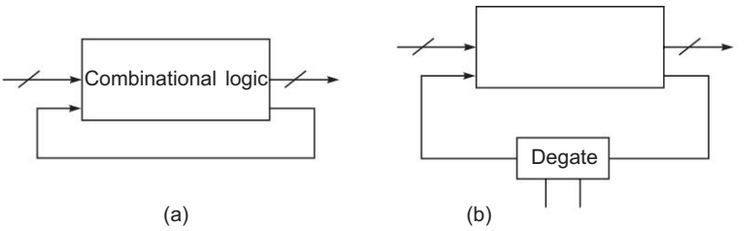


Fig. 7.20 (a) Combinational logic with feedback, forming an asynchronous sequential circuit. (b) Use of a degating circuit to break the feedback path.

One situation in which a signal path can cause difficulty in testing is depicted in Fig. 7.20(a), which shows a block of combinational logic with a feedback path around it. This feedback path will, in general, convert the combinational circuit into an asynchronous sequential one. The use of asynchronous design methods, as discussed earlier, is a very dubious practice for normal functioning, while for the engineer, it makes all aspects of the process more difficult. TPG has to be approached on a structural base using the gate-level equivalent, and control of operations by the ATE is made more difficult because of the absence of a master clock. A fully synchronous design is much to be preferred, but it is considered essential to include a feedback path of this kind; then at least it should be breakable for testing purposes. The use of a degating circuit as shown in Fig. 7.20 (b) is one way of achieving this.

A common cause of testing difficulty is represented in Fig. 7.21(a), which shows an oscillator (typically, a clock generator) embedded within a circuit without any means of either controlling or observing its operation. While the circuit is being tested, the ATE ought to supply the clock signals. At the very least, it needs to be able to monitor internal clock signals so that it can synchronize to them. If it is prevented from doing either then testing become almost impossible. Degating the clock, as shown in Fig. 7.21(b), provides a solution to the problem.

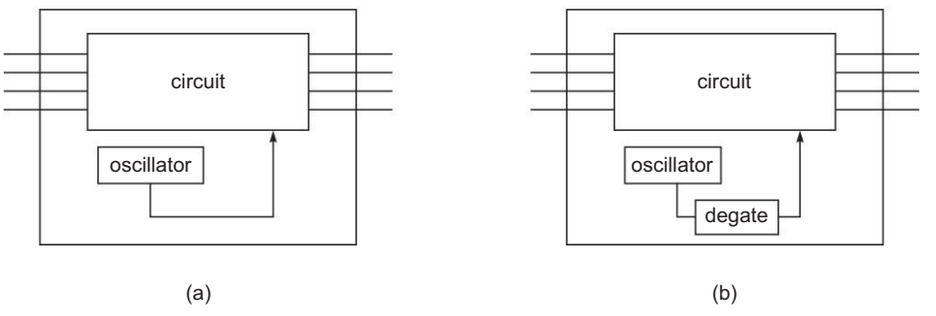


Fig. 7.21 (a) A circuit with an embedded oscillator. (b) Use of a degating circuit to give control to the ATE.

Counters are standard elements that find a variety of uses in circuit implementations. A real-time clock, for example, can be obtained by counting down from the system master clock, and a counter can also be used ready-made. An FSM that generates a fixed sequence of the circuit as shown in Fig. 7.21. A basic clock on the operation of this counter would require clocking it through its range, while checks on the remaining circuitry are likely to require setting the counter to particular values. For both of this purposes, a long counter may take an unacceptably long time to deal with; a twenty-stage counter

requires more than a million pulses to take it through its range. Two improvements can be made, as shown in Fig. 7.22. The first is to make the reset input available to the tester even if it is not needed for the functional system. The second is to break up the counter chain using a degating circuit; by splitting a twenty-stage counter into two ten-stage counters, the range can be scanned in thousand pulses rather than a million.

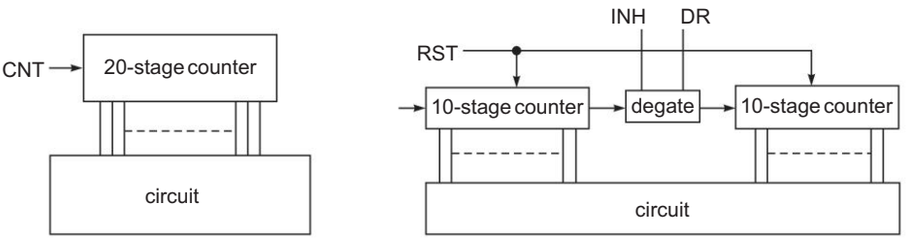


Fig. 7.22 (a) long counters take a long time to set up. (b) Breaking up a counter using a degating circuit

It will be clear that all of the modifications suggested so far in the interests of enhanced testability entail increased costs under some or all of four headings:

(a) Extra Pin Any electronics restricted for test purposes must require at least one dedication pin so as to distinguish test mode from normal mode.

(b) Extra Silicon Additional components (gates, multiplexers and so on) together with the associated wiring make additional demands on silicon area.

(c) Reduced Performance In many cases, additional gates are inserted into the signal pathways. This implies increases in propagation delays.

(d) Reduced Reliability If the circuit has more components, there are more things to go wrong.

While all these costs cannot be denied, the justification for DFT lies in the subsequent reduction in the costs of TPG and test execution. Indeed, without at least some concessions to DFT, it is doubtful whether the most complex chips could be economically manufactured at all.

¤ REFERENCES

7.1. J.P Roth, "Diagnosis of automate failures: a calculus and a method," *IBM Journal of Research and Development*, vol.10, no. 7, July 1966, pp. 278–291.

7.2. F.F. Sellers, M.Y. Hsiao, and C.L. Bearnson, "Analyzing errors with the Boolean difference," *IEEE Trans. On Computer*, July 1968, pp. 676–683.

7.3. P.H. Bardel, W.H. McAnney, and J. Savir, *Built-in Test for VLSI*: *Pseudorandom Techniques*, NY: John Wiley & Sons, Inc., 1987.

7.4. http://standards.ieee.org/reading/ieee/std_public/description/testtech/1149.1-1990_desc.html IEEE standard Tests Access port and Boundary Scan Architecture, IEEE Standard 1149.1 1990, IEEE Standards Board,1990

7.5. K.P. Parker, *The Boundary-Scan Handbook*, 2nd Edition, Analog and Digital, Kluwer Academic Publishers, 1998.
7.6. L. Crouch, *Design for Test for Digital IC's and Embedded Core Systems*, Prentice-Hall, 1999.
7.7. M. Abromovici, M.A. Breuer, and A.D. Friedma, *Digital Systems Testing and Testable Design,* Computer Science Press,1990
7.8. J. Rajski and J. Tyszer, *Arithmetic Built-In Self-Test for Embedded Systems*, Prentice Hall, 1998.
7.9. R.K. Gulati and C.F. Hawkins eds., *IDDQ Testing of VLSI Circuits—A Special Issue of Journal of Electronic Testing: Theory and Applications*, Kluwer Academic Publishers, 1995.

EXERCISES

7.1 Find the pseudo-random sequences in 4-bit LFSRs defined by the following polynomials: (a) $x^4 + x^3 + x^2 + 1$ (b) $x^4 + x^2 + x$ (c) $x^4 + x^3 + 1$ (d) $x^4 + x^3 + x^2$

7.2 Verify the 5-value logic operation for *D*-algorithm given in Fig. 7.5.

7.3 Develop a test set that detects all single stuck-at faults in Fig. P7.1.

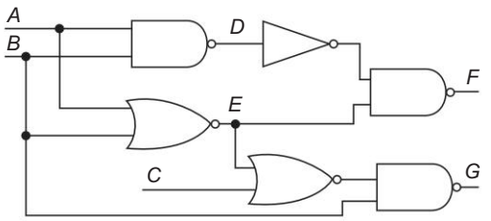


Fig. P7.1

7.4 Find the singular cover for a logic function $Z = \overline{a.b + c}$

7.5 Find the propagation *D*-cube for the logic function $Z = \overline{a.b + c}$

7.6 Find the primitive *D*-cube for $Z = \overline{a.b + c}$ when *Z*: *s*-*a*-1

7.7 Show that $\frac{d(F \oplus G)}{dx} = \frac{df}{dx} \oplus \frac{dG}{dx}$

8

Physical Design of VLSI Circuits

Day by day, the integration scale is increasing multiplicatively and now it has gone up to more than 10^8 transistors in a chip, as per requirement of different applications such as high-speed memory and high-performance special processors. The design of these chips requires automation in the design process, where its algorithmic analysis is used. So the availability of fast and easily implementable algorithms is essential in the discipline. The previous discussion of the circuit is the starting point and also essential for future improvements in the design performances and evaluation. In this direction, physical design is the process of automation in which physical location of active devices and interconnecting them inside the boundary of a VLSI chip are estimated. This chapter focuses on the layout problem that plays an important role in the design process of chip architectures. The cost of fabricating a circuit is a function of circuit area and circuit-layout techniques to produce the layouts with a small area. These layouts show special structure to generate their wireability in which wire-length minimization and power minimization have also to be taken into consideration. In present day's systems, delay minimisation is becoming more crucial than other performance parameters in the chip. The aim is to design fast circuits within a small chip area with low power consumption. As for the medical and electronics industry, high speed, reliability and thermal stability along with cost-effectiveness are the main objectives.

8.1 □ LAYOUT METHODOLOGIES

The thermal stability and reliability of chips are obtained through wire-length minimization whereas the speed and cost-effectiveness are achieved through delay minimization and area minimisation respectively. The layout problems are typically solved in a hierarchical framework. Each stage of this framework should be optimized, while making the problem manageable for subsequent stages. For this, the following subproblems are considered as shown in Fig. 8.1.

- **Partitioning** is a stage of dividing a circuit into different parts so that each component is within the prescribed ranges and the number of the connections between these components is minimised. A good partitioning corresponds to improve the circuit performances and reduce layout cost which is function of area and wire length between connection.
- **Floor planning** is the task of determining the approximate location of each module in the rectangular chip area of a given circuit represented by hypergraph—shape of each module and location of the pins on the boundary of each module may be determined in this phase. A good floor planning should provide minimization of chip area and reduction of the signal delay.

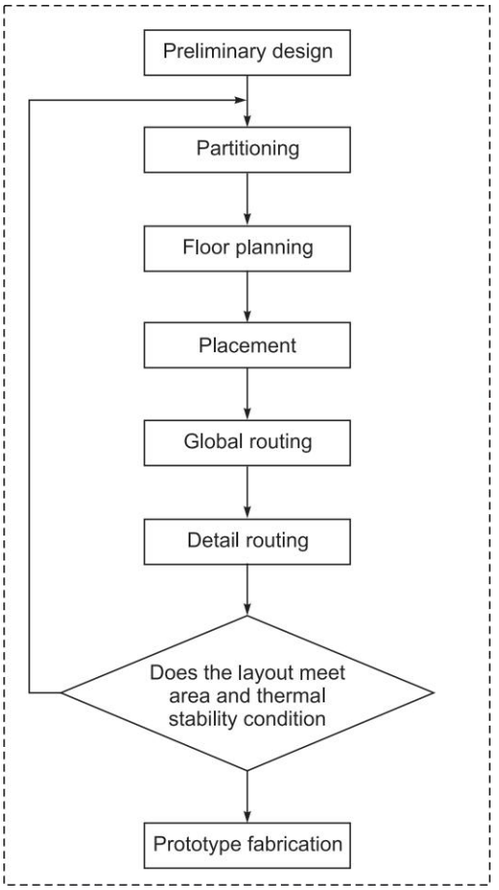


Fig. 8.1 Layout design (physical design)

- **Placement** is the task of determining the best position of the module. Normally, some modules are fixed with floor planning (considering input/output pads). The positions of other modules are determined by employing the alternate cost function which is a function of wire length and chip area. The placement corresponds to the chip area where each module has a fixed shape and area.
- **Global routing** is the task of decomposing a large routing problem into small manageable problems for detail routing, keeping the chip area same. It decomposes the routing region into a collection of disjoint rectilinear subregions. This decomposition is carried out by finding rough paths between these subregions.
- **Detailed routing** follows the global routing. In the traditional method of detailed routing, the horizontal wires on one layer and vertical wires are routed on other layers. The interconnections between vertical and horizontal wires are made by metallic contacts. There are two types of detail routing—single layer and multilayer routing.
- **Layout optimisation** is a postprocessing step where layout is again optimised by compacting area
- **Layout verification** is the testing of a layout to determine whether it satisfies design rules, layout rules and design specifications. In CAD packages, the layout is verified in terms of timing and delay.

The above steps are followed in full custom-design automation. The following sections discuss the development of CAD tools for layout problems. The algorithms used in such tools should be of high quality and efficiency.

8.2 □ PARTITIONING

Circuit partitioning is the task of dividing a circuit into smaller parts where the sizes of the components are within the prescribed ranges and the numbers of connection between two parts are minimized. In physical design, partitioning is a fundamental step in transforming a large problem into smaller sub-problems of manageable sizes. It can be applied at the IC level, board level and system level. The main purpose of the partitioning is to improve the circuit performances and reduce layout costs. Generally, the circuit is transformed into a graph model before partitioning. In physical design, there is no efficient algorithm to apply different steps of partitioning into the circuit directly. So, a circuit is transformed into a graph before subsequent algorithms are applied.

There are two ways of partitioning—bipartitioning and multi-way partitioning. **Bipartition** is a technique to partition the graph into two parts at a time whereas **multi-way partitioning** is a technique to partition the graph into many parts (more than two) into two parts.

8.2.1 Conversion of Circuit into Graph and Hypergraph

A graph $G(V, E)$ consists of a set of vertices V and set of edges E where the circuit elements (shown in Fig. 8.2(a)) are mapped into vertices and connections are mapped into edges. A hypergraph $H(V,L)$ consists of a set of vertices V and a set of hyper-edges L instead of edges in the graph. Figure 8.2 (b) shows the graph model of the circuit and Fig. 8.2(c) shows the hypergraph model. The circuit can be represented either by a graph model or hypergraph model or both graph and hypergraph. The vortex weight is used for indication of size of the corresponding circuit element. The partitioning algorithms used for partitioning are applied to these graph models. Traditionally, it is difficult to design efficient algorithms based on a hypergraph. Thus, it is required to transform a circuit into graph and if hyper graph is present during transformation of circuit into graph, the additional step is required for replacing hyper-edges with asset of edges such that the edge costs closely resemble the original hypergraph when processed in subsequent stages.

Consider a hyper-edge, $e_a = (M_1, M_2 \ldots\ldots M_n)$, (where $n > 2$) with n terminals and the weight of hyper-edge e_a be $w(e_a)$. One way to represent e_a is to put an edge between every pair of distinct modules (M_i, M_j) with weight $w(e_a)/n_0$ where n_0 is the number of added edges. Before applying a partitioning algorithm, it is required to transform the hypergraph into a graph using the following algorithm:

Procedure: Hypergraph transformed into graph

```
begin-1
for each hyperedge e_a = (M_1, M_2.....M_n), do
begin-2
form complete graph with vertices (M_1, M_2.....M_n) with edge (M_i, M_j),
weight is proportional to the number of hyper edges between M_i
and M_j.
find minimum spanning tree T_a of G_o, replace e_a with edges T_a in
hypergraph
end-2
end-1
```

Example 8.1 ◻ *Convert the following circuit into a hypergraph and graph model and convert all the hypergraphs with hyper-edges into a graph with edges.*

Solution The circuit in Fig. 8.2 (a) is converted into hypergraph and graph form as shown in Fig. 8.3(b) and (c) respectively. The figure shows eight vertices for eight transistors.

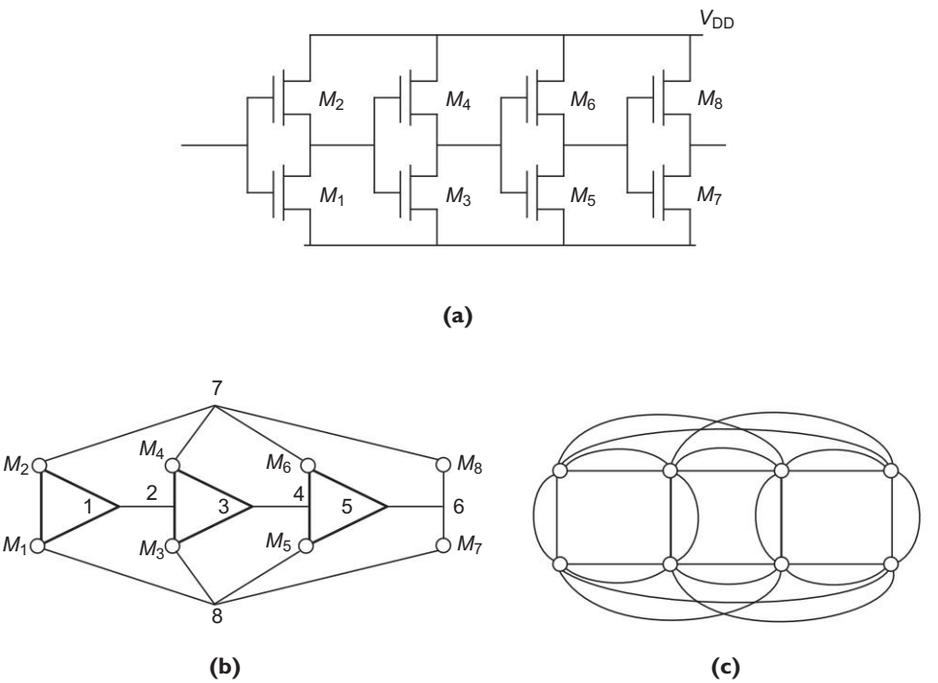


Fig. 8.2 Converted circuit: (a) Circuit model (b) Graph and hypergraph (c) Graph only

8.2.2 Bipartition Algorithm

Most of the algorithm related partitioning are based on bipartitioning in which the graph model of the circuit is partitioned into two parts at a time. They are Kernighan–Lin algorithm, Ratio cut algorithm, and Fiducia Mattheyses heuristic algorithm which are based on bipartitioning.

1. Kernighan–Lin algorithm

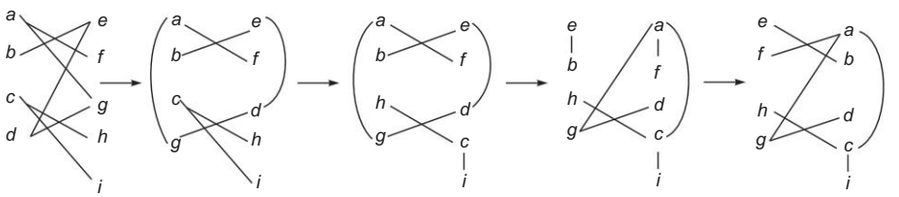
Kernighan–Lin algorithm is based on iterative improvement proposed by Kernighan and Lin. For an unweighted graph G, the technique begins with an arbitrary partition of G into two groups V_1 and V_2 such that $|V_1| = |V_1| \pm 1$ for odd numbers of vertices and $|V_1| = |V_2|$ for even number of vertices (where $|V_1|$, $|V_2|$ are number of vertices in subsets V_1 and V_2 respectively). Then, the vertex pairs (v_a, v_b) are chosen (where $v_a \in V_1$ and $v_b \in V_2$) so that the exchange of these vertices results into decrease of cut cost or slight decrease of cut cost. The cut cost is defined as number of cut of edges by partition line.

If there is a decrease of cut cost, the vertices v_a and v_b are locked. The process is continued, till all the vertices in V_1 and V_2 are locked to get $\text{Gain}_k < 0$, where $\text{Gain}_k = \text{Cut cost}_{k-1} - \text{Cut cost}_k$ and k is suffix representing k^{th} step of iterations. The procedure of the algorithm is given below:

Procedure: Kernighan–Lin algorithm

```
begin-1
Bipartition G into two parts V1 and V2 with |V1| = |V1| ± 1 for odd
number of vertices and |V1| = |V1| for even number of vertices
repeat-2
for k = 1 to n/2 do
begin-3
find a pair of unlocked vertices va and vb where va ∈ V1 and vb ∈ V2
whose exchanges results into the decrease or small increase in
cut cost and mark two vertices as locked.
if Gaink ≤ 0,
end-3
end-2
```

The time complexity is estimated as follows: the 'for loop' in algorithm is executed in $O(n)$ times wheras the body of the loop requires $O(n^2)$ times. Step-1 takes $(n/2)X\ (n/2)$ times and Step(i) takes $(n/2 - i + 1)^2$. The running time of the algorithm is $O(n^3)$ for each pass of the repeat loop. The total running time is $O(cn^3)$, where c = number of times of repeat loop. Figure 8.3 shows an example of Kernighan-Lin algorithm in which number of iterations are made for getting the final solution of the partition. Iteration number (3) gives the final partition solution.



Iteration	Vertex pair	gain	cut cost
0	–	–	7
1	(d, g)	2	5
2	(c, h)	1	4
3	(a, e)	1	3
4	(b, f)	–2	5

Fig. 8.3 Steps of Kernighan-Lin algorithm

2. *Fiduccia–Matheyses Algorithm*

The Kernighan-Lin algorithm has been improved by Fiduccia and Matheyses (FM) where reduction of time complexity per pass is $O(t)$, where t is the number of terminals. The following have to be introduced to the Kernighan–Lin algorithm for FM algorithm. The data structure for two partitioned sets A and B can be written as $(1 - d_{\max} w_{\max}, d_{\max} w_{\max})$, where $d_{\max}$ is the maximum vertex degree and $w_{\max}$

is the maximum cost of an edge. Moving one vertex from one set to another set leads to change in the cost by d_{max}. w_{max} and as a result, a balanced partition is maintained during the process. Maximum vertex weight w should satisfy the balanced partition condition $w \geq w(v) + \max[w(v)]$, where $w(v)$ is the weight of the vertex, v in previous partition set. This balanced partition is obtained by sorting vertex weights in decreasing order.

The algorithm starts with a balanced partition A and B (where $w(A) \leq W$ and $w(B) \leq W$) of graph G. A move of a vertex across the cut is allowable if such a move satisfies the balance condition. To choose the next vertex to be moved, the maximum gain in vertex will be a_{max} in part A and b_{max} in part B. No moves are allowed without decrease of cut cost and for locked vertex, the main advantage of this algorithm over Kernighan-Lin algorithm is no restriction of number of vertices in partition sets A and B.

3. Ratio-cut Algorithm

The ratio-cut algorithm is one of the efficient bipartitioning technique by which one can reduce cut cost more than that of Kernighan-Lin algorithm because there is no restriction in number of vertices in each partition. This approach is based on the following concept. A graph G consists of a number of vertices V and number of edges E. The (V_A, V_B) denotes the partition sets A and B in which $V_A = 1 - V_B$. Let C_{ij} be cost of an edge connecting an edge between two vertices v_i and $v_{j,}$ where $v_i \in V_A$, $v_j \in V_B$. The total cut cost is given by

$$C_{AB} = \sum_{v_j \in V_B} \sum_{v_i \in V_A} C_{ij}$$

The cut-size ratio can be written as

$$R_{AB} = \frac{C_{AB}}{|V_A|.|V_B|}$$

where, $|V_A|$ and $|V_B|$ are the number of vertices of partitions A and B respectively. The ratio-cut algorithm is NP complete and consists of three phases—initialization, iterative shifting and group swapping.

Initialization

(a) Select a node/vertex s arbitrarily and another node/vertex t which is further from the node s so that $x = \{s, t\}$ and $y = V - \{s, t\}$.
(b) Choose a node k whose movement to x will generate the best cut-size ratio and include the node in x and update $x = XU\{k\}$ and $y = Y - \{k\}$.
(c) Repeat Step-2 until the cut provides lower cut-size ratio.

Iterative Shifting An initial partitioning is made and two nodes s and t are kept fixed and initial partitioning is recorded. The next step is iterative shifting which is given below:

(a) Shift the nodes (more than or equal to one) from right to left side of cut line. It is called a right shifting.
(b) Shift the nodes (more than or equal to one) from left to right side of the cut line. It is called a right shifting
(c) Repeat Step 1 and 2 till best cut size ratio is obtained.

Group Swapping For further improvement after iterative shifting, group swapping is made to reduce cut-size ratio by making all the nodes locked. The process is given below:

(a) Calculate the cut size ratio for every node *j* movement by making all the nodes unlocked.
(b) Select unlocked node *j* with movement between two subsets and update the cut-size ratio and if cut-size ratio is improved then lock the node *j*.
(c) Repeat Step-2 with unlocked node, till all the nodes are locked.

Figure 8.4 shows an example of how the initialization and iterative phases of ratio-cut algorithm are implemented.

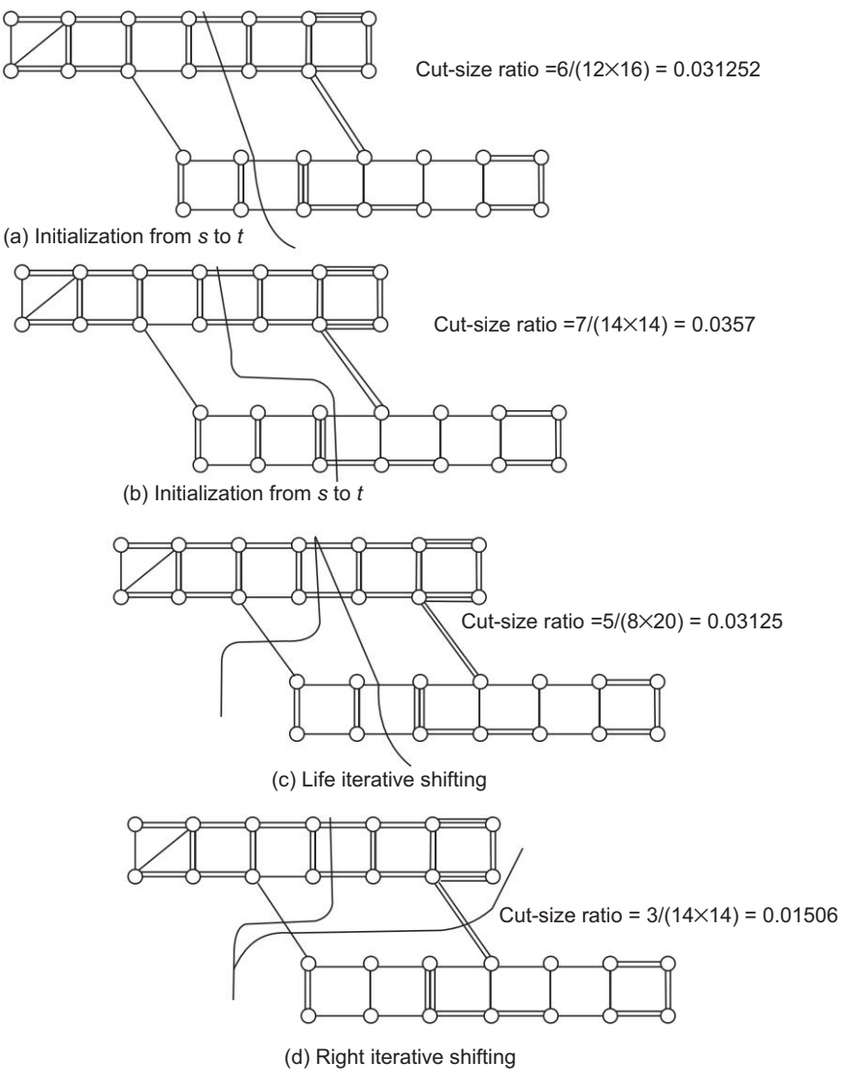


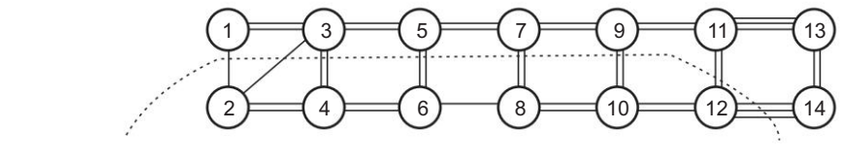
Fig. 8.4 Example of first two phases of ratio-cut algorithm

4. Ratio-cut Genetic Bipartition

In genetic algorithm, the solution of partitioning is expressed as a chromosome. The chromosome is a linear string which is given by

$$< a_1, a_2, \ldots\ldots a_n >$$

where, a_1, a_2, ……a_n are gene values for nodes/vertices 1, 2,….n. For bipartitioning, gene values are based on binary numbers, either 0 or 1, depending on which part it belongs to. If for the node belonging to part-1, gene value is considered to be 0 then for the node belonging to part-2, gene value will become 1. As for an example, we consider Fig. 8.5 representing bipartition into part-1 and part-2.



Gene number	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Gene values of chromosome	0	1	0	1	0	1	0	1	0	1	0	1	0	0

(a)

Crossover-1

Parent-1: 0 1 0 1 | 0 1 0 1 | 0 1 0 1 | 0 0

Offspring-1: 0 1 0 1 | 1 0 1 0 | 0 1 0 1 | 1 0

Parent-2: 1 0 1 0 1 0 1 0 1 0 1 1 1 0

Crossover-2

Parent-1: 0 1 0 1 | 0 1 0 1 | 0 1 0 1 | 0 0

Offspring-2: 1 0 1 0 | 0 1 0 1 | 1 0 1 1 | 0 0

Parent-2: 1 0 1 0 1 0 1 0 1 0 1 1 1 0

(b)

Fig. 8.5 (a) Chromosome encoding for bi-partitioning (b) Crossover

Figure 8.5(a) shows the chromosome encoding for bipartition. The genetic algorithms begin with a set of randomly generated bipartition solutions/chromosomes called populations. Two members of populations are chosen by using best cut-size ratio as Parent-1 and Parent-2. The off/spring chromosomes are generated by using a crossover operator. Figure 8.5 (a) shows crossover operators in which the part of the gene content of parent-1 is copied first, then the part of the gene content of parent-2 is copied and the same from parent-1 and parent-2 alternatively. In the reverse way, the offspring-2 is generated. The part of the contents from parent-1 and parent-2 may be chosen equally or unequally. After crossover, the next step is mutation of the offspring-1 and offspring-2. Each gene of the offspring chromosome is complemented to get low cut-size ratio. The procedure of ratio cut genetic bipartitioning is given below.

Procedure: Ratio cut genetic bipartitioning

```
begin-1
Create an initial population
repeat-2
{ Choose parent-1 and 2 from population;
Offspring-1 = Crossover-1 ( parent-1 and parent-2)
```

```
Offspring-2 = Crossover-2 ( parent-1 and parent-2)
Mutation ( offspring-1 and offspring-2)
If offspring suited then
Replace the earlier offspring
}
until-2
minimum cut-size ratio is obtained
and best solution is obtained
end-1
```

8.2.3 Multi-partitioning

The generalization of multi-way partitioning is also one issue in VLSI design. To reduce computation time, multi-way partitioning is used. There are two approaches in multi-partitioning. In most cases, bipartition is used iteratively to partition the graph into two blocks, then partition each of the blocks into sub-blocks and further each sub-block into other sub-blocks and so on. In another approach, multi-partitioning of the graph is done by partitioning the graph into more than two blocks at the same time to reduce the computation time. We have already discussed bipartitioning. In this section, we discuss genetic multi-partitioning based on the later approach.

In genetic algorithm, the solution of partitioning is expressed as a chromosome. The chromosome is a linear string which is given by

$$< a_1, a_2, \ldots\ldots a_n >$$

where, a_1, a_2, ……a_n are gene values for nodes/vertices 1, 2,….n. For multi-partitioning, gene values are considered to be decimal numbers 0, 1, 2… depending on the number of partitioned parts. For tri-partitioning, gene values are 0, 1, 2. For four partitioning, gene values are 0, 1, 2, 3. For n partitioning, gene values are 0, 1, 2… n – 1. As for an example, we consider Fig. 8.6 representing tri-partition into part-1 (gene value 0), part-2 (gene value 1) and part-3 (gene value 2).

Figure 8.6(a) shows the chromosome encoding for tri-partition. The genetic algorithm begins with a set of randomly generated tri-partition solutions/chromosomes called populations. Two members of

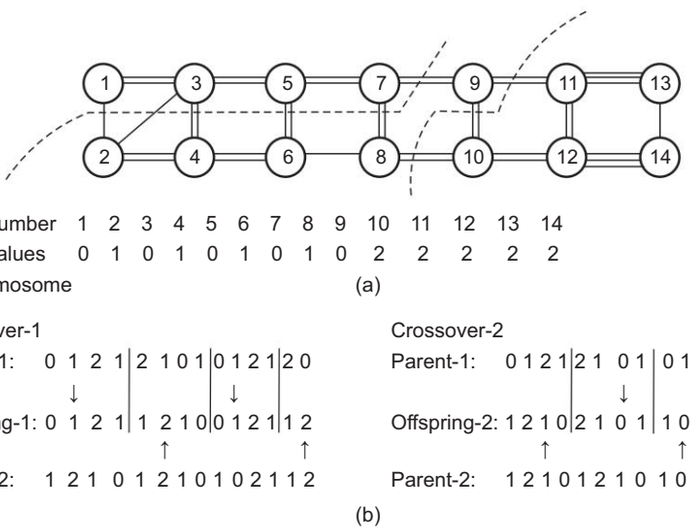


Fig. 8.6 (a) Chromosome encoding for tri-partitioning (b) Crossover

populations are chosen by considering best cut-size ratio as Parent-1 and Parent-2. The offspring chromosomes are generated by using crossover operator. Figure 8.6(b) shows crossover operators in which the part of the gene content of Parent-1 is copied first, then the part of the gene content of Parent-2 is copied and the same from Parent-1 and Parent-2 alternatively. In the reverse way, the offspring-2 is generated. The part of the contents from Parent-1 and Parent-2 may be chosen equally or unequally. After crossover, the next step is mutation of the Offspring-1 and Offspring-2. Each gene of the offspring chromosome is complemented to get low cut-size ratio. The steps of procedure ratio-cut genetic multipartitioning are same as that of bipartitioning.

8.3 ☐ FLOOR PLANS

In the circuit $C(M, N)$ (where M = number of connections, N = number of components) represented by graph $G(V, E)$ (where V = number of models/versions, E = number of edges), the floor planning deals with determination of approximate position for each partitioned module in the rectangular chip area. The following goals are obtained for floor planning:

- Minimise the total chip area
- Make subsequent routing phase easy
- Improve performance, reducing signal delays

It is difficult to achieve these goals together. Mainly minimisation of chip area is considered in floor planning. The set of nets N defines the closeness of modules placing highly connected modules close to each other reducing routing space in a chip. A floor plan is represented by rectangular dissections where the border of a floor plan is a rectangle, since it is a convenient structure for chip processing. The rectangle is separated by several straight lines. There are three types of floor plans-sliceable floor plan, non slice floor plan and hierarchical floor plan. A **sliceable floor plan** is one of the simplest types of floor plans in which a floor plan can be bipartitioned into two sliceable floor plans with horizontal or vertical cut lines, as shown in Fig. 8.7(a). The figure also shows the binary tree. A **nonsliceable floor plan** is a floor plan that cannot be bipartitioned into two sliceable floor plans with vertical or horizontal

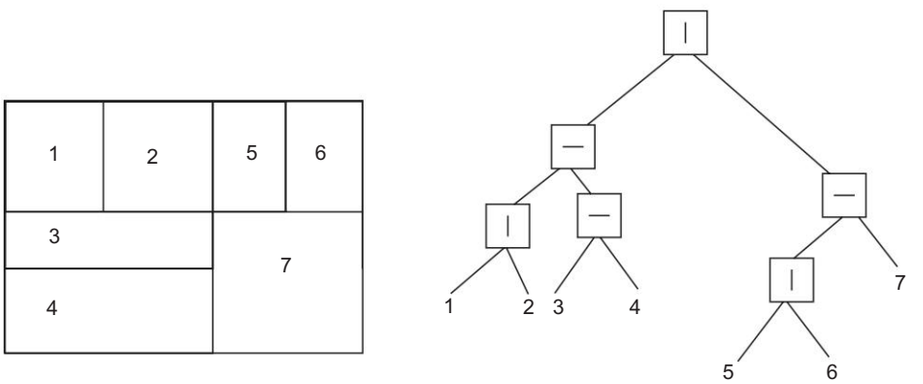


Fig. 8.7(a) Sliceable floor plan with binary tree

cut lines as shown in Fig 8.7(b) where the corresponding binary tree is also mentioned. There are two types of nonsliceable— floor L5 and R5—in which there are five floor-planned rectangles. The **hierarchical floor plans** are a combination of sliceable and nonsliceable floor plans. Figure 8.7(c) shows the hierarchical R5 floor plan with a binary tree.

Most algorithms for floorplan are based on sliceable floor plan. A floorplan sizing problem for a sliceable floor plan is NP complete. Especially, a hierarchical sliceable floor plan has more time complexity than other floor plans.

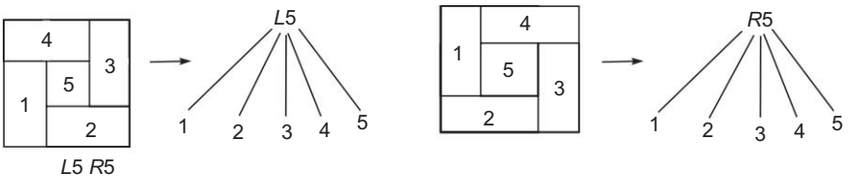


Fig. 8.7(b) Nonsliceable floor plan L5 with binary tree and R5 with binary tree

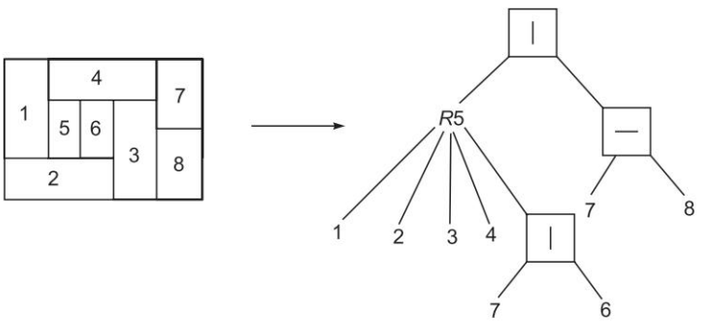


Fig. 8.7(c) Hierarchical R5 floor plan with binary tree

8.3.1 Rectangular Dual-Graph Approach to Floor Plan

Rectangular dual-graph approach is based on the proximity relation of a floor plan. A rectangular dual graph of a rectangular floor plan is a plane graph $G(V, E)$ where V = set of modules and (M_i, M_j) where M_i and M_j are adjacent in the floor plan. Planar Triangular Graph (PTG) representation of rectangle floor plan is represented in Fig. 8.8. A floor plan F is enclosed with infinite region r,u,l,b as shown in

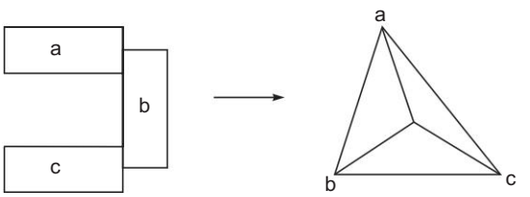


Fig. 8.8 Planar triangular graph having forbidden pattern in a rectangular dual graph

Fig. 8.9(a). The figure also shows corresponding PTG which is also called **extended dual graph**. Let N be the number of verities of extended dual $G_e(n)$. By induction, we can form another dual graph $G_e(K)$ where $K < n$. There are two cases—(a) some vertices have degree 3, and (b) none of the vertices have degree 3.

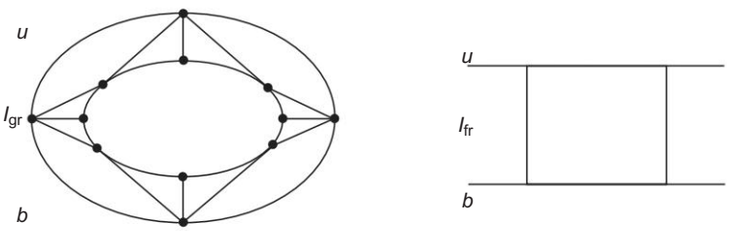


Fig. 8.9(a) Extended dual with floor plan enclosed by four infinite regions

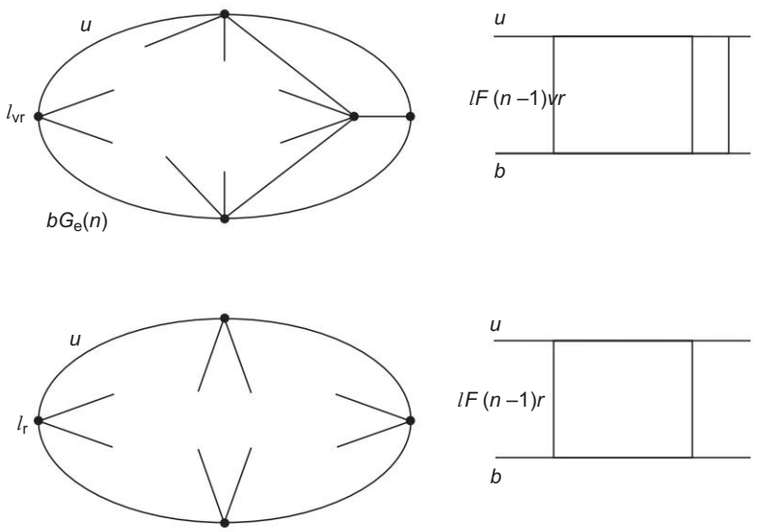


Fig. 8.9(b) Dual decomposition and floor plan merging of rectangular graph

Vertex r has degree 3. Since (r, u), $(r, b)^b \in G_e(n)$ consider vertex 9 and one edge (r, u) only where (r,u,l,b). We can write $G_e(n)$ in terms of $G_e(n–1)$ and its floor plan $F(n\text{-}1)$ over none of verties (r,u,l,b) has degree 3. Find the path $P_v = \{u, = P_1, P_2\text{---}, P_u = b\}$ in $G_e(n)$ from u to b with the following properties:

1. P_2, ---------- $P_{k\text{-}1} \notin (r,u,l,b)$
2. $(P_2, P_1) \notin (G_e\, l_n)$ for some I, and
3. $(P_i, r) \notin (G_e\, l_n)$ and $(P_i, r) \notin (G_e\, l_n)$ for some I

Such a path is called a **vertical splitting path**. The horizontal splitting path from l to r can be defined. $G_e(n)$ composes along P_V to obtain two sub groups G_e and G_r ([Fig. 8.9 (b)]) where G_e consists

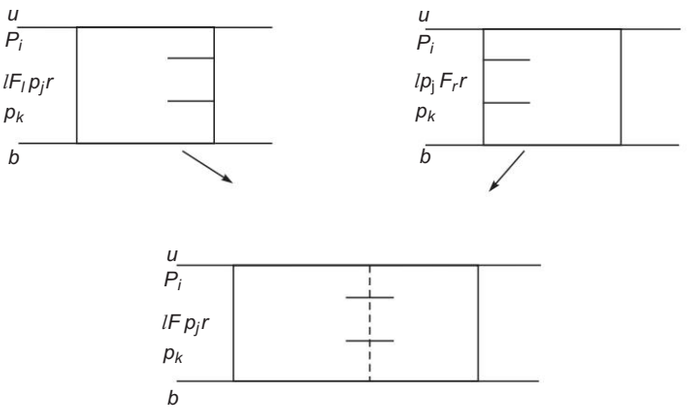


Fig. 8.9 (c) Merging of Floor plan F_l and F_r to obtain F

of vertices to the right of P_V. Their corresponding floor plans are F_e and F_r as shown in Fig. 8.9(c). The floor plans F_e and F_r are merged to obtain the floor plan of G_e.

The rectangular dual-graph approach is not well accepted in floor planning because of many problems of quantitative aspect and complicated dual approach.

8.3.2 Hierarchical Approach

Hierarchical approach to floor planning is widely used and there are two types of approach—bottom-up and top-down approach.

1. Bottom-up Approach

The modules are represented as a graph where the edges represent the connectivity of the modules. The modules with high connectivity are clustered together while limiting the number in each cluster to d or less. A greedy clustering procedure is used to sort the edges by decreasing weights. Figure 8.10 shows the bottom-up hierarchy floor plan where the heaviest edge is chosen and two modules of the edges are clustered in greedy fashion while restricting the number in each cluster to d or less. One of the problems with a simple approach is that some lightweight edges are chosen at higher levels in the hierarchy floorplan. The next high level vertices in a cluster are merged and edge weights are summed up.

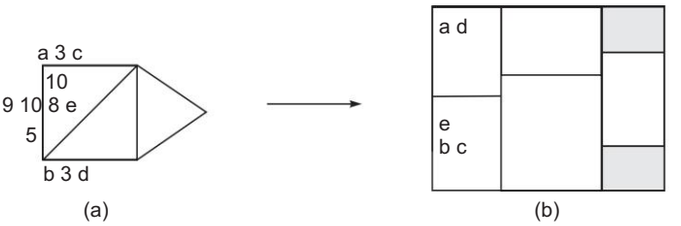


Fig. 8.10 (a) Circuit connectivity (b) Floor plan obtained by greedy bottom-up approach

2. Top-down Approach

A hierarchical floor plan can also be constructed in a top-down manner. The fundamental step in this is assigned in the partitioning of modules. Each partition is assigned to a child floor plan and we consider minimum cut–maximum flow algorithm. We can combine both bottom-up and top-down approach in which a set of clusters are obtained for getting the best floor plan.

8.3.3 Simulated Annealing

Simulated annealing is a technique used to solve floor-planning problems using its optimization approach. The idea of simulated annealing comes from crystal formation concept. When a material is heated, the molecules move around in a random motion and when the temperature slowly decreases, the random movement of the molecules tends to be stopped and eventually it forms a crystal structure: Depending on cooling rate, the materials achieve a stronger crystal lattice.

Considering the above concept, simulated annealing algorithm is formulated on the basis of configuration of the problem sequence. Each configuration provides a feasible solution of the problem. It moves from one solution to other solution till the best cost function is obtained. Initially because of random nature of the problem, high temperature is considered for the problem. As the algorithm proceeds, the temperature decreases and randomness of the problem is also reduced. The movement of one solution to another solution is such that the temperature of the algorithm decreases and the best cost function is obtained at a particular low temperature which is obtained from specification of the vendors. Before discussion of algorithm procedure, some functions are to be defined before implementation in the floor plan. Typically, the number of feasible solutions is an exponential function of the problem size. The following functions are used in this algorithm:

- frozen( ) determines the termination condition of the algorithm.
- equilibrium( ) is used to decide the termination condition of random movement.
- f( ) is a function that returns a value between 0 and 1 to indicate the derivability of accepting the next solution and function f( ) is basically Boltzman probability function $e\ sc/U_{BT}$ where ΔC is cost change and K_B = Boltzman's Constant.
- random( ) is to have a high probability of accepting high cost movement at high temperature and it returns a number between 0 and 1.
- cost( ) is a function that determines global cost of the solution.
- generate( ) is a function that selects the next solution from current solutions following cut edge of the configuration graph.

Algorithm procedure is given below:

```
Input: Modules representing circuit and its sizes.
Output: A solution S with low cost.
begin-I
S: = random initialization;
T: = (initial temperature);
while not frozen (T) do
begin-2
count: = 0;
while not equilibrium(count,S,T) do
```

```
begin-3
count: = count + 1;
next s = generate (s);
if cost (next (s)) < Cost (s) or
(f (Cost (s), Cost (next (s)), T) > random (0,1) then S: next (s);
end-3;
update (T);
end-2;
end-1;
```

Implementation of Floor Planning Based on Simulated Annealing

The important issues of simulated annealing algorithm are

(a) Solution space
(b) Movement from one solution to other
(c) Cost-evaluation function

The algorithm is based on sliceable floo rplans which can be represented by a tree. For easy representation, estimation of floor plans and easy implementation of simulated annealing algorithm, Polish expression notation is used. Polish expression having a string of symbols operators (vertical/horizontal) and operands (modules) is obtained from binary tree of sliceable floor plan. Figure 8.11 shows a floor plan indicating a binary tree and its corresponding Polish expression. The figure shows the Polish expression having operands 1, 2, 3, 4 and operators.

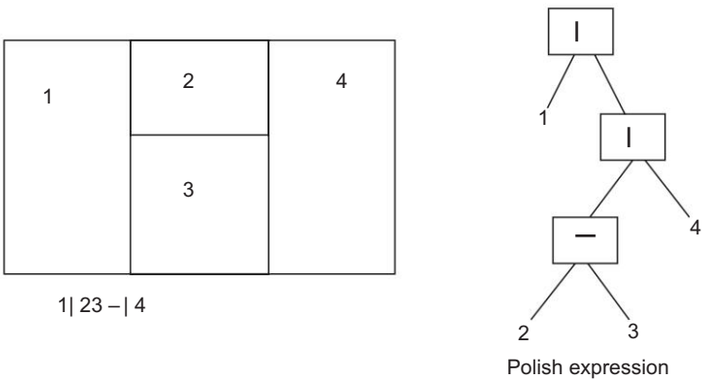


Fig. 8.11 Floor plan and its corresponding binary tree and Polish expression

Movement from one solution to another solution can be translated from one Polish expression notation to another notation. This translations from one to another Polish expression should obey the following rules.

OPT 1 Exchange two operands when there are no other operands in between.

OPT 2 Complement a series of operators between two operands.

OPT 3 Exchange the adjacent operand and operator if the resulting expression is a normalized polish expression where no two consecutive operands are identical.

As for example:

Modules 1: Size (2,2)

Module 2: Size (2,3)

Module 3: Size (1,2)

Module 4: Size (4,2)

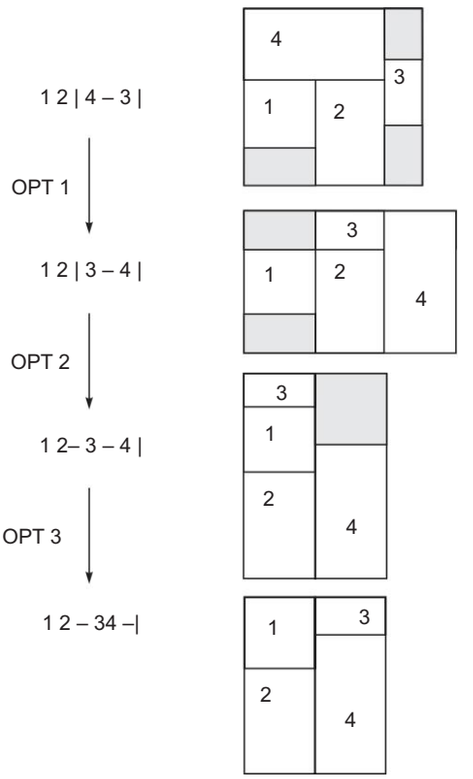


Fig. 8.12 A series of movements in simulated annealing following rules of Polish expression

Figure 8.12 shows series of movements obeying rules of Polish expression. These movements are followed in simulated annealing to get the best solution of floor plan. The figure shows an initial floor plan represented by Polish expression 12|4–3|. After implementation of rule–I, rule–2 and rule–3, the Polish expression becomes 12|3–4|,12-3-4| and 12-34-| respectively. The final Polish expression 12-34-| provides the floor plan of the lower chip area.

8.3.4 Floor-plan Sizing

In VLSI design, the circuit modules are usually of different sizes. A good choice of module implementation may lead to minimised amount of wasted space/unused space. The floor-plan sizing is a

technique to estimate module-implementation area. There are two types of approaches used for floor-plan sizing—hierarchical floor-plan sizing and nonhierarchical floor-plan sizing.

1. Hierarchical Floor-plan Sizing

The hierarchical floor-plan sizing finds an area occupied by cells after implementation of a sliceable floor plan. It is noted that the horizontal and vertical dependency graphs of a sliceable floor plan are series parallel graphs. In this approach, one by one module are considered as per slice floor-plan to find floor plan sizing. There is vertical mode sizing and horizontal mode sizing. The procedure for generating vertical mode sizing is given below:

```
Input: Two sorted list of modules L = { (a,b,) ---------(a_s, b_s)}
R = {(x_1, y_1) …..(x_t, y_t)} where a_i < a_j and b_i > b_j, x_i < x_j ; y_i > y_j
for all i<j.
Output: H = {(c_1, d_1)------- (c_u, d_u)} where u ≤ s + t − 1, c_i < c_j,
d_i d_j for all i<j
begin-1
H: = ϕ
i: = 1, j: = 1, k = 1;
while (i ≤ s) j ≤ t, do
begin-2
(C_k, d_k): = (a_i + x_j, max (b_i Y_i));
H: = HU{ C_k, d_k};
k: = k + 1;
if max (b_i, y_i) = b_i, then
i: = i + 1;
if max (b_i, y_j) = y_j then
J: = J + 1;
end-2;
end-1;
```

Figure 8.13 shows vertical mode sizing of two modules $M_i = (a_i, b_i)$ $M_j = (x_j, y_j)$.

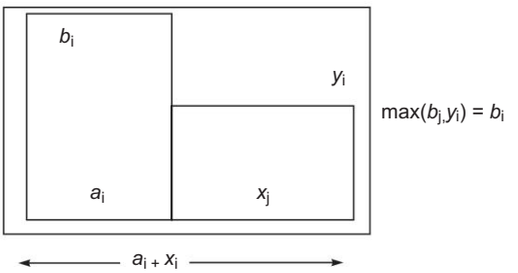


Fig. 8.13 Horizontal mode sizing

Figure 8.14 shows horizontal mode sizing of two modules. $M_i = (a_i, b_i)$ $M_j = (x_j, y_j)$.

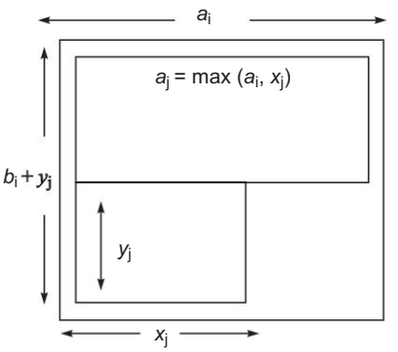


Fig. 8.14 Horizontal mode sizing

The algorithm of horizontal mode sizing is opposite to the vertical mode sizing in which the following is considered:

$$(C_K, d_k) = (\max(a_i, x_j), (b_i + y_j))$$

where $(c_K, d_k) \in H$ and $(a_i, b_i) \in L$ and $(x_j, y_j) \in R$.

2. Nonhierarchical Floor-plan Sizing

Nonhierarchical floor-plan sizing has no restriction on organisation of the modules. The approach is based on mixed Linear Programming (LP). The main part of LP based approaches is the formulation of LP equations where the following notations are used:

w_i, h_i: width and height of module, M_i

(x_i, y_i): co-ordinates of the lower left corner q of module M_i

(x, y): width and height of the final floor-plan

(a_i, b_i): minimum and maximum values of aspect ratio w_i/h_i for module M_o

The non-overlapp constraints are

$$x_i + w_i \le x_j$$
$$x_j + w_j \le x_i$$
$$y_i + h_i \le y_j$$
$$y_j + h_j \le y_i$$

where M_i module is on the left of the module M_j.

The module size constraints are

$$w_i h_i \ge A_i$$
$$a_i \le w_i / h_i \le b_i$$

The values of maximum w and h are

$$W_{min} = \sqrt{A_i a_i}$$
$$W_{max} = \sqrt{A_i b_i}$$
$$h_{min} = \sqrt{A_i / b_i}$$
$$h_{max} = \sqrt{A_i / a_i}$$

For each pair of modules M_i and M_j, two variables are introduced—p_{ij} and q_{ij}—that assume values 0 or 1 and there are two large numbers w and H which are the upper bounds of width and height of the solution. So the inequalities are

$$x_i + w_i \le x_j + w(p_{ij} + q_{ij})$$
$$x_j + w_j \le x_i + W(1 - p_{ij} + q_{ij})$$
$$y_i + h_j \le y_j + H(1 + p_{ij} - q_{ij})$$
$$y_j + h_j \le y_i + H(2 - p_{ij} - q_{ij})$$

The cost function is given by

Min y

$$x_i + w_i \le w$$
$$y \ge y_i + h_i$$

where area is $A = x_y$. The unknown variables are x_i y_i, w_i, p_{ij} and q_{ij}. All other variables are known. These equations are solved by LP solver software.

8.4 □ PLACEMENT

The input in the placement problem is a set of modules and a net list where each module is of fixed size. The net list provides connection information among the modules. The main part of placement is the best position of each module on the chip to achieve appropriate cost function depending on chip area and total wire lengths in the chip. Placement algorithms have two major classes—iterative placement and constructive placement. In case of iterative approach for placement, it starts with initial placement and repeatedly modifies in search of placement with a better cost function. In constructive approach, a good placement is constructed in a global sense. In iterative approach, there is force-directed method and simulated annealing algorithm. The partitioning and resistive network techniques are classified as constructive placement algorithms. The force-directed method can also be applied in constructive approach. Apart from these placements, there is another approach—assignment problem and linear placement.

In case of placement, two parameters, chip area and wire lengths in chip, have to be minimized simultaneously. It is difficult to minimise these two parameters together. To estimate these, it is required to express them in terms of cost function which consists of wire-length cost function L_W and chip-area function ℓ_A. The wire-length cost function ℓ_W is *w*ritten as

$$\ell_W = \sum_{n_i \in N} \frac{P_i}{2}$$

where P_i is the parameter of net and L_W is estimate of the total net length. of course, a small wire length provides a small chip area. These two cost functions can be combined with a scaling factor λ as

$$\text{Cost} = \lambda \ell_A + (1 - \lambda)\ell_w$$

where λ = Scaling factor ($0 \le \lambda \le 1$)

8.4.1 Force-directed Method Iterative approach

Modules that are highly interconnected are to be placed close to each other. We can consider force pulling these modules towards each other as a parameter for placement. The interaction is a parameter for placement. The interaction between two modules M_i and M_j can be expressed as

$$F_{ij} = -C_{ij}\, d_{ij}$$

where C_{ij} is a weighted sum of the nets between two modules M_i and M_j and $|d_{ij}|$ is a vector directed from centre of M_i and M_j and is written as $|d_{ij}| = |x_i - x_i| + |y_i - y_j|$ where (x_i, y_i) and (x_j, y_j) are coordinates of M_i and M_j. The optimal placement is obtained as one that minimizes the sum of force/interaction vector acting on the modules.

In this interactive force directed method we start with our initial positions of the modules.

1. A module with the maximal total force acting unit is identified. Denote this module as M and place it at the coordinate (X, Y) so that there is no overlap and force F_i on it due other modules is almost zero.
2. Repeat Step-1 for all modules with more force directed interaction.
3. Improve the placement with exchange of all placed modules so that total force F $(= \Sigma_{ij} F_{ij})$ is minimized where hf_{ij} = Force directed on i^{th} molule M_i due to i^{th} modules M_i.

In case of interactive force-directed approach, the modules are considered to be some size. If the modules are not of same size, then different strategies have been taken.

Constructive Approach

The following steps are used for constructive force directed algorithm.

Step 1 An initial placement is constructed by placing the modules so that they are in equilibrium with respect to the forces acting on them.

Step 2 Find a placement so that the vector sum of the forces acting on each module is zero.

A solution to this problem can be obtained by solving a nonlinear system of equations as follows. We consider Mo be a module with its final position denoted by (x_0, y_0). The set of modules connected to M_o are denoted by $\{M_1, \ldots, M_s\}$, where M_i has the final position (x_j, y_i), for $1 < i < s$. The x-component of the set of forces acting on M_0 is set to zero which is given by

$$\sum_i C_{0i} d^{x}_{0i} = 0$$

where d^{x}_{0i} is the magnitude of the x-component of the vector d_{0i} from (x_i, y_i) to (x_0, y_0) and C_{0i} is a weighted sum of the nets between M_0 and M_i. Similarly, y component of force is also zero i.e.

$$\sum_i C_{0i} d^{y}_{0i} = 0$$

where d^{y}_{0i} is the magnitude of the y-component of the vector d_{0i} from (x_i, y_i) to (x_0, y_0)

If there are no modules with predetermined positions, then a trivial solution is obtained by placing the center of all modules at an arbitrary point (x, y). There is a restriction in placement i.e. the overlap of modules is not allowed.

8.4.2 Placement Based on Simulated Annealing

The placement algorithm based on simulated annealing starts with an initial placement, accepting all perturbations or moves which result in a reduction in cost function. For simulated annealing, it is required to define the temperature and its relation with length and width of chip. The relation between temperature and length is *w*ritten as

$$L_W(T) = L_W(T_1) \frac{\log T}{\log T_1}$$

$$L_H(T) = L_H(T_1) \frac{\log T}{\log T_1}$$

where T = Current temperature, T_1 = Previous temperature, L_W (T_1) and L_H (T_1) are previous values of length and width of the chip respeclively. The cost function in terms of L_H and L_W can be written as

$$C_1 = \Sigma\,[x(i)\,wh(i) + y(i)\,w_w(i)]$$

where $wh(i)$ and $w_w(i)$ are weight factors of horizontal and vertical span of chip, and $x(i)$ and $y(i)$ are vertical and horizontal span of chip.

The simulated algorithm procedure is given below:

```
Input: Optimization Problem
Output: Solution with low cost
begin-1
S = random initial placement;
T = T1 (initial temperature);
while not froezn (T) do
begin-2
count: = 0;
begin-3
count = count + 1;
next s = generate (s);
if cost (next s) < Cost (s) or
f (cost (s), cost (next (s) T) > random (0,1)
then s: = next (s);
end-3
update (T);
end-2;
end-1;
```

where `f()` is a well-known Boltzman probability function $e^{-\Delta C/K_B T}$, where ΔC = Cost change = Cost (next (s) – cost (s)), K_B = Boltzman constant, and T = Current temperature. The function `f()` returns a random number between 0 and 1. `Generate( )` is a function that selects the next placement or solution from the current solution. `update (T)` reduces the temperature to cool down. The process starts with a high initial temperature.

8.4.3 Module Placement Based on Resistive Network

Resistive-network-based module placement is a constructive approach that uses resistive networks as a working domain. The cost function is the sum of the squares of wire lengths (to make the transformation) to the network domain straightforward. The algorithm includes optimization, relaxation, partitioning and assignment. The algorithm has running time of O (n^{14} log n) where n = number of modules. We consider the modules to be placed at coordinates (x_i, y_i) where i = 0,1…., n. The cost function is given by

$$\phi(X,Y) = \frac{1}{2}\sum_{i,j=1}^{n} C_{ij}\,[(x_i - x_j)^2 + (y_i - y_j)^2]$$

where C_{ij} = Number of wires connected between modules i and j. In matrix form, it is written as

$$\phi(X, Y) = x^T Bx + y^T BY$$

where $B = D - C$, C is the connectivity matrix, D is the diagonal matrix whose i^{th} element d_{ii} is equal to $\sum_{j=1}^{n} C_{ij}$. For optimization we consider a one-dimensional problem because of symmetry of x and y.

This approach is based on resistive network in which the admittance matrix is considered for *n*-terminal linear passive resistive network. The power dissipation in the resistive network is given by

$$P = v^T Y_n v$$

where v is an n-vector matrix representing voltage and y_n is admittance of n^{th} terminal. The cost function of placement for this approach becomes power dissipation. Figure 8.15 shows n-terminal resistive network, considering m modes on the left side are floating and their voltages are denoted by an m-vector v_1. The remaining $(n - m)$ nodes are connected to voltage sources denoted by an $(n - m)$ vector V_2.

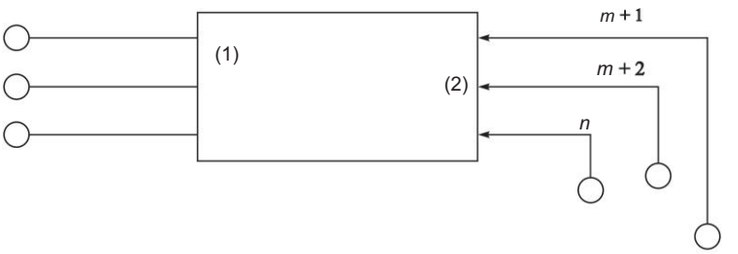


Fig. 8.15 *n*-terminal resistive network

So, the coordinates of n modules are represented by

$$v = \begin{bmatrix} v_1 \\ \hline v_2 \end{bmatrix}$$

The network equations are *w*ritten as

$$\begin{aligned} 0 &= y_{11}\, v_1 + Y_{12}\, v_2 \\ i_2 &= y_{21} v_1 + Y_{22}\, v_2 \\ v_1 &= Y_{11}^{-1} Y_{12}\, v_2 \end{aligned}$$

where y_{11}, $y_{12} = y_{21}{}^T$ and y_{22} are short-circuit admittance sub-matrices. The voltage v_1 represents a set of values which has prescribed slots in terms of the permutation vector $P = \{P_1, P_2 \text{ ------- } P_{\text{m}}\}^T$ where m = Number of modules and $P_2 = I$ to legal value. Let $v_1 = [X_1, X_2 \text{ ------- } X_{\text{m}}]^T$ where x_{i} = Coordinate of the module or voltage at the node i. The constraints equation are written as

$$\sum_{i=1}^{m} x_1 = \sum_{i=1}^{m} P_i$$

$$\sum_{i=1}^{m} X_i^2 = \sum_{i=1}^{m} p_i^2$$

$$\cdots \cdots$$

$$\sum_{i=1}^{m} x_i^m = \sum_{i=1}^{m} p_i^m$$

Module voltages are determined from the above equation.
The first equation can be written as

$$d = l^T v_1 = \ell^T p$$

where l is a unit vector and d is a constant which is equal to the sum of m legal values.

Again assume that in the region there are k modules and m legal values given by the permutation vector $[p, p_2 \ldots p_{\text{k}}]$ and $[x_{01}, x_{02} \ldots x_{\text{ok}}]$ denote the solution obtained from optimization with linear

constraints. [x_{n1}, x_{n2}….. x_{nk}] denote the new coordinates after scaling. Thus, the objective is to minimize $\sum_{i=1}^{k}(x_{ni} - x_{oi})^2$

The constraints are

$$\sum_{i=1}^{k} x_{ni} = \sum_{i=1}^{k} P_{ni}$$

$$\sum_{i=1}^{k} x_{ni}^2 = \sum_{i=1}^{k} P_{ni}^2$$

$$\cdots\ \cdots$$

$$\sum_{i=1}^{k} X_{ni}^m = \sum_{i=1}^{k} P_{ni}^m$$

where $x_{ni} = \frac{x_{oi} - C_o}{a_o} a_n + C_n$. C_o, a_o, C_n and a_n are functions of k, p_i and x_{oi}.

The relaxation step is used for repeated scaling and optimisation. The overall procedure is given below:

1. Initial optimization over entire region using initial equations.
2. Scaling and optimization are made over subregions partitioned in chip area.
3. Repeat Step-2 by performing optimization scaling and relaxation independently to get best minimized power dissipation.

8.4.4 Regular Placement

Regular placement is a placement in which predetermined positions (called targets) are assigned to modules. Here each module should be assigned to a target. There are different approaches used for regular placement assignment approach and genetic algorithm approach.

1. Assignment Approach

Assignment problem can be solved in two steps—relaxed placement and removing overlaps. In **relaxed placement phase**, the positions of the modules for the targets are determined by using cost function and overlapping of modules in a target is also allowed to minimize the cost function. The cost function for the target j is defined as

$$C_{IJ} = \sum_{N_i \in N} W(i)\,[x_{r,i} - x_{l,i}]$$

where, for a net N_i, $x_{l,i}$ is the leftmost position of M_i, $x_{r,i}$ is the rightmost position of M_i, and x_i is possible location the of module M_i. The cost function for each module to place for a target is estimated by the above equation and this results in the reduction of chip area and wire length as given in the equation. At the end of the relaxed placement phase, the solution may have overlapping of modules. All the overlaps are removed in the second step. Firstly, the costs of all modules placed in the targets are estimated. The following steps are used for assignment approach:

1. Assign each module to target and find total cost $\sum C_{ij}$, where $m \le n$, m = number of modules and n = number of targets.
2. Repeat Step-1, till minimise $\sum_{i,j} C_{ij}$, where $M_i = i^{th}$ module placed in the target H_j
3. Find overlappings of modules in the target and remove these overlappings.

Figure 8.16 shows an example of assignment problem in which there are four targets—*a*, *b*, *c*, and *d*, and four modules—1, 2, 3, and 4. The costs of modules assigned to targets are given below:

$C_{1a} = 1$	$C_{2a} = 2$	$C_{3a} = 3$	$C_{4a} = 5$
$C_{1b} = 2$	$C_{2b} = 1$	$C_{3b} = 4$	$C_{4b} = 3$
$C_{1c} = 1$	$C_{2c} = 3$	$C_{3c} = 2$	$C_{4c} = 3$
$C_{1d} = 3$	$C_{2d} = 4$	$C_{3d} = 1$	$C_{4d} = 4$

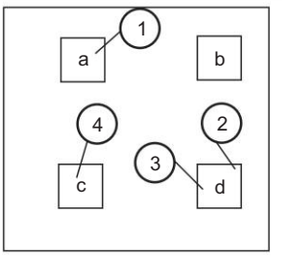


Fig. 8.16 Chip area (having targets *a*, *b*, *c*, *d*) placing modules 1 in a, 4 in *c* and 2 and 3 in *d*

The solution of this assignment problem after removal of overlapping of modules in target *d* is given by

Module-1 ---------- target *a*
Module-2 ---------- target *b*
Module-3 ---------- target *d*
Module-4 ---------- target *c*

2. Genetic Approach

The regular placement using assignment has two steps—relaxed step and removal of overlap and because of this, more computation is needed. In genetic approach, step for removal of overlapping of modules in a target is not required separately and during coding of chromosomes for the solution, it is taken care of. The solutions of the placement problem are evaluated from chromosomes which are coded in the following manner. In this case, number of modules should be equal to number of targets.

Target: *a b c d e f g h*
Modules 1 2 3 4 5 6 7 8
a b c d e f g h

Chromosome-1	4	3	2	1	6	5	7	8
Chromosome-2	2	4	1	3	6	7	8	5

The chromosomes are constructed by planning each module in a target where there is no overlapping of modules. After initial generation of chromosomes, two chromosomes are chosen as parent chromosomes for crossover. The diagonal crossover is used for generation of offspring chromosome as shown in Fig. 8.17.

Crossover operator-1

Parent-1	4	3	2	1	6	5	7	8
	4	3	2	1	6	5	7	8
Parent-2	2	4	1	3	6	7	8	5

offspring Chromosome-1

Crossover operator-2

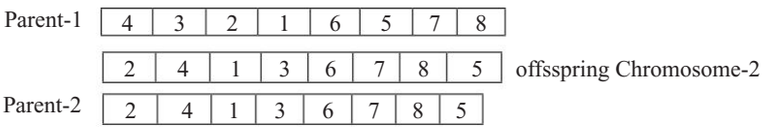


Fig. 8.17 Diagonal crossover operator for genetic algorithm.

The algorithm steps for genetic approach of regular placement is given below:

Step-1: Initial population: generation of chromosomes represently placement

Step-2: Choose two chromosomes randomly as parents—parent-1 and parent-2

Step-3: Offspring chromosome-1 = crossover-1 (Parent-1, Parent-2); offspring chromosome-2 = crossover-2 (Parent-1, Parent-2).

Step-4: Repeat Step-2 and Step-3, till cost function is minimised.

8.5 □ ROUTING

Routing is a step for finding signal paths in chip area. Generally, routing is used for selection of paths based on signal delays. The routing problem has two steps—global routing and detail routing. There are three fundamental concepts used for solving these routing problems—maze running, line searching and Steiner tree.

8.5.1 Maze Running

The maze—running approach is used for finding the shortest path in a geometric domain. It is based on grid form of chip area with obstacles. The chip area is expressed in terms of grids with obstacles. The routing path is to start from one terminal called **source terminal** to reach finally the **target terminal**. There are two ways of search, starting from source to finally target—one-directional search and bidirectional search.

1. One-directional Search

Figure 8.18 (a) shows the grid form of a chip (with obstacles) in which routing is started from source and all the grids adjacent to the source are labelled with 1. Then the grids adjacent to the grids marked with 1 are labelled with 2 and next with 3, and so on to reach the target 1 terminal called **sink terminal**. The labelled grids (adjacent to each other as shown in the figure) from source to target indicate its routing path length. Any unlabelled grid point p that is adjacent to the grid point marked with label i is assigned the label $(i + 1)$. Two grid points are adjacent, if they are either horizontally or vertically adjacent. If they are not adjacent then they are diagonally neighbouring. This type of approach to find a path from one terminal to another terminal is called **Lee's algorithm** or **Lee–Moore's algorithm**. In the figure, total distance from source to target is 8.

The major drawback of maze–running approach is the huge amount of memory used to label the grid points in the process. Attempts are being made to remove this difficulty.

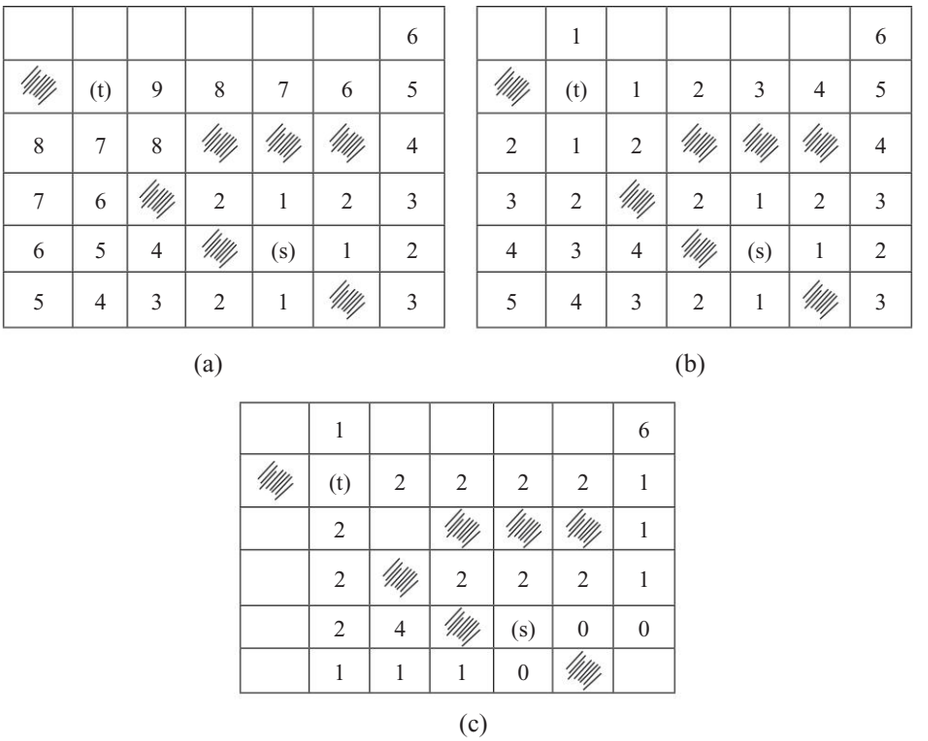


Fig. 8.18 Demonstration of Lee's maze running algorithm: (a) One direction made to remove this difficult search (b) Bidirectional search (c) Minimization of number of bends

2. Bidirectional Search

An effective approach to speed up the maze-running algorithm and to solve the problem of requirement of huge memory is to perform a bidirectional search which it starts both from source and sink terminal, and labels all adjacent grid points of both source and sink terminals with 1. Then, all grid points adjacent to the grid point marked with 1 are labeled by 2. In general, at stage i, all the unlabelled grid points and adjacent grid points with label $i - 1$ are labeled with i. The task is repeated until the search from source s reaches the search from the sink at stage j. If they reach diagonally in Fig. 8.18(b) then the length of the shortest path is $2j + 1$.

3. Minimum Cost Path and Bent Path

The goal is to minimise the length of the path between the source and sink. If two paths give the same shortest path then one should consider a path of minimum number of bends. To find a path of minimum number of bends, all grid points that are reachable with zero bends from source are labeled with 0 and all the grid points that are reachable from grid with label zero with one bend are marked with 1. In general, for stage i, all the grid points that are reachable from the grid point with label $i - 1$ with one bend are labeled by i. For each grid point with label i, it is necessary to store the direction of the path (if there are more than one paths satisfying shortest path) that connects the source to that grid point with i bends. Figure 8.18(c) shows an example for finding minimum number of bends of signal path.

8.5.2 Multilayer Routing

Multilayer routing can be achieved with the maze-running algorithm. The labelling proceeds as before to minimise the number of layers along with minimum number of bends and minimum cost.

4. Line Searching

There are two classes of search algorithms for finding the path between two routing points—source and sink. The first one is a grid search which has already been discussed in the previous section. In grid search, the time and space complexity is too high even though it is easy to construct the search space. To reduce space and time complexity, a second class of search techniques is used called line searching.

The algorithm starts from both points—source and sink to be connected and passes a horizontal and vertical line through both points. These lines are called **probes**. The lines originating from the source are called **source probes** whereas the lines generating from sink are called sink probes. These lines are first-level probes. When the source probes and sink probes meet then a path between source and sink is found. These probes will not meet, if they are intersected by an obstacle which will discontinue the probes from their intersections. A line is passed perpendicular to the previous probe and the constructed line on probe is called **next-level probe**. The task is repeated till at least one source probe meets at least one sink probe and a path between source and sink has been found (Fig. 8.19a).

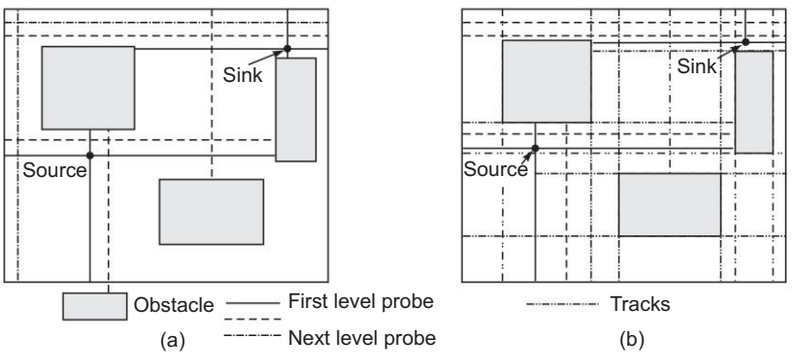


Fig. 8.19 (a) Example of line searching approach (b) Line searching using track graphs

Although the path can be found by using the above, it might take a long time to find the path which is an average path. The line-searching method can be modified to reduce the time by half for finding the path by using the **track-graph method**. A track graph is made by extending the horizontal and vertical sides of each obstacle until another obstacle is reached, in addition to passing a horizontal and vertical line (called first probes) from source and sink. The next probes are made if first-level probes have obstacles till all source probes meet sink probes using track lines. This process is quick and finds a shortcut path early. Figure 8.19(b) shows demonstration of the line-searching approach based on track graph.

8.5.3 Steiner Tree

A tree connecting a set of routing points, $P = \{p_1, p_2 ..... p_n\}$ in the rectilinear plane and some arbitrary points is called a (rectilinear) Steiner tree with minimum total cost. The Steiner tree is based on different problems such as minimization of length and weight factor. The following problems are considered here:

- **Minimum-length Steiner tree**—The goal is to minimize the sum of length.
- **Weighted rectilinear Steiner tree**—Here, the given routing is partitioned into a collection of weighted regions. An edge with length l in i^{th} region and weight w_i has cost $w_i l_i$. The goal is to minimise total cost $\sum_i w_i l_i$
- **Steiner with arbitrary orientation**—Here, geometry + 45° and – 45°, in addition to vertical and horizontal lines, are considered.
- **Minimum length Steiner tree**—In this case, the total routing length $\sum\sum l_{ij}$ has to be minimized for a Steiner tree which connects a set of points to be routed on two-dimensional chip area (where l_{ij} = Routing path between two routing points i and j. The problem is NP-complete.

There are different rectilinear Steiner tree topologies. We can consider routing channels which are parallel lines on which routing points are lying. When the points are on the boundary of a rectangle, this Steiner tree is called switch box. The part of the Steiner tree that is made inside the switch box is called **interior segment**. When the two lines—vertical and horizontal lines—cross each other in a switch box, it is called **cross**. When there are vertical lines, and the first and last vertical lines are connected to horizontal lines of the boundary of a switch box, it is called **earthworms** (Fig. 8.20). A corner is made by vertical and horizontals lines connected to the center with the boundary of a switch box, called **corner topology**.

1. Weighted Rectilinear Steiner Tree

In this approach, the chip area plane has to be divided into different weighted regions R_1, R_2 ……R_m where m = Total number of weighted regions. Region R_1 is assigned weight w_1, region R_2 is assigned weight w_2, and so on. A path is considered to be conducting between two points P_a and P_b. Let l_i denote the length of a path P in the region R_i, where $l_i = |P \cap R_i|$. The total weight of P is $w(P) = \sum_{i=1}^{m} l_i w_i$.

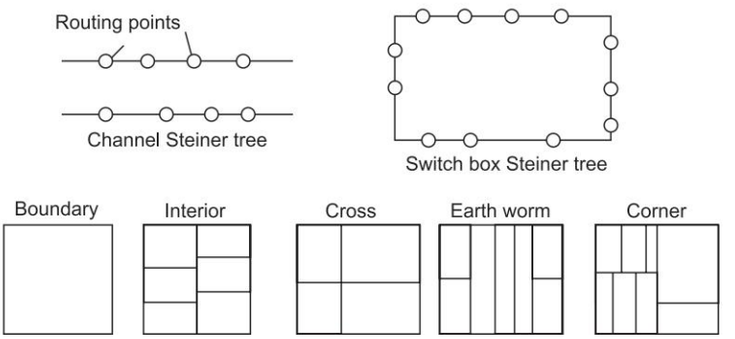


Fig. 8.20 Different topologies of Steiner trees

A minimum Weighted Rectilinear Steiner Tree (WRST) is required to find minimum weighted paths between different routing points. For getting wRST, the first step is to make a track graph on the bound-

ary of obstacles extended to boundary of chip and obstacles as shown in Fig. 8.21. The obstacles are assigned infinite weight. The algorithm procedure is given below:

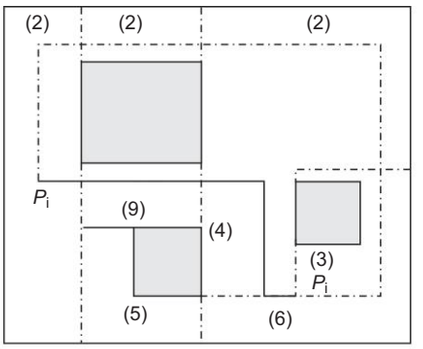


Fig 8.21 Routing of two points using WRST

```
Procedure: Layout – wRST (R, P) ; where routing region R and path P.
  begin-1
 T_P = A minimum Steiner tree of P
 for j = 1 to n-1 do
 begin-2
 for i = 1 to K do
 begin-3
 L_i = MERGE (Sub-1, Sub-2, Path_j (ej) )
 cleanup (L_j)
 end-3
 save minimum weight tree
 end-2
 end-1
```

where function `MERGE ( )` gives a path between routing points and the function `cleanup( )` is the function which removes repeated edges to obtain a tree.

2. Steiner Trees with Arbitrary Orientation

The Steiner tree, discussed in the previous section, is based on rectilinear geometry. Using this geometry, the shortest path may not be obtained. The most commonly employed geometric environments are Euclidean space and rectilinear space. In Euclidean geometry, arbitrary orientation is allowed whereas in rectilinear geometry, horizontal and vertical orientations are permitted. In Steiner tree with arbitrary orientation, it is required to consider the uniform λ-geometry (e.g., 45° environment) which removes the problems of implementation of Euclidean geometry and provides better results than rectilinear geometry. This allows orientation making angles $i\pi/\lambda$. Figure 8.22 shows λ-geometry representation for finding the routing path between two points P_1 and P_2. The following properties of λ-geometry have

to be established. The $\frac{\lambda}{2}$ geometry is rectilinear geometry which is a special case of Steiner tree with arbitrary orientation. There are different λ-geometries as shown in Fig. 8.22(b), (c) and (d).

The SMT T_1 can be replaced by SMT T_2 with direction edges in λ-geometry. The line segments are connected to Steiner point's angle as evenly as possible. A generalization of the LAYOUT_WRST algorithm can be employed to effectively construct a Steiner tree in λ-geometry.

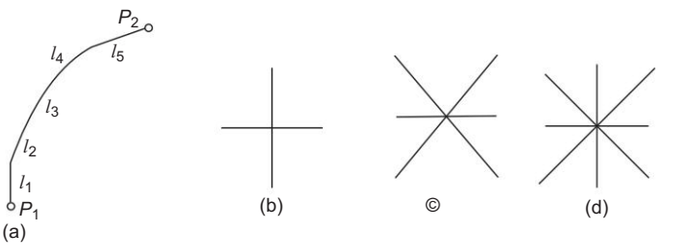


Fig. 8.22 (a) Measuring distance (b) λ = 2 geometry (c) λ = 3 geometry (d) λ = 4 geometry

8.5.4 Global Routing

The routing of chip is complicated as a large number of routing paints have to be found for paths. There are two types of routers used for routing of the chip—**global router** which is used to decompose a large routing problem into small and manageable sub-problems, whereas the **detail router** is used to route each small and manageable sub-problem. The decomposition in a global router is carried out by finding a rough path for each net, i.e., sequence of sub-regions passed through in order to decrease chip size and wire length, and distribute the congestion over the routing area. The sub-regions depend on floor planning or placement steps before global routing.

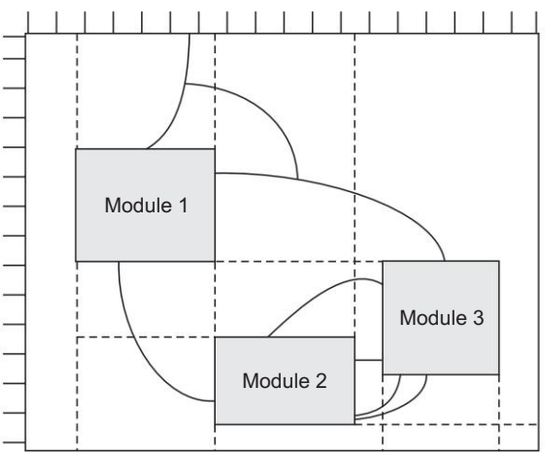


Fig. 8.23 An example of global routing

After floor planning and placement, the routing region is partitioned into simpler regions which are rectangular in shape. The partitioning of sub-regions for routing is to define the routing graph. Two sub-regions are connected when the channels are adjacent without affecting chip area. These sub-regions are placed in such a way that two sub-regions are closer to each other when there are more channels as shown in Fig. 8.23. In this routing, the exact position of each module is determined to make the routing net. There are different approaches used for global routings in which routings are made between different sub-regions. The approaches are sequential approach, hierarchical approach, randomized approach, integer linear programming approach, and one-step approach.

1. Sequential Approach

In sequential approach for global routing, nets are routed one at a time, i.e., sequentially. The ordering of the nets has to be obtained for going for the second step in which each net is routed by Steiner tree. The sequential approach for global router has made an attempt to find the Steiner tree, minimizing wire length and traffic in this region. It is difficult to get minimization of both the parameters. Using minimum-length steiner tree, the wire length can be minimized but it is difficult to minimize the traffic by Steiner tree because it is minimized heuristically. To remove the difficulties, modifications are made on the Steiner tree. Instead of taking minimum-length Steiner, a weighted Steiner tree is used when dealing with wire length and traffic density. In this method, the first step is ordering of the nets and on the basis of ordering, the second step is formulated as a Steiner-tree problem. The routers are made one by one. Ordering in a Steiner tree is called **LAYOUT-WRST**. The constraint λ^j is introduced to balance weight and length in WRST. At j^{th} pass, we will find WRST of net N with minimum $\sum_i \lambda_i^j W_i^j l_i^j$ for all nets where W_i^j is the weight of region R_i that N passes through and l_i^j is the length of N in R_i. The value λ_i^j is found so that $\lambda_i^j W_i^j$ approaches to 1 as j increases. The algorithm procedure is given below:

```
Routing (R, P):
 begin-1
 w: = initial weight of routing function R;
 for i = 1 to n do
 w(Ni ) = α; (initial weight of all nets)
 Pass j;
 for i = 1 to n do
 begin-2
 Ni = current net;
 Temp: = LAYOUT-WRST (Ni); (minimize ∑ λi^j Wi^j Li^j of net Ni)
 if Temp ≤ w (Ni) then
 begin -3
 Ni: = Temp (accepting routing of Net Ni)
 update w;
 end-3;
 end-2;
 end-1;
```

Practically, λ_i^j is selected as

$$\lambda_i^j = \left(\frac{i}{j} + \frac{i}{W_i^j} \right)$$

2. Hierarchical Approaches

The hierarchical-routing approach is based on hierarchy on the routing graph to decompose a large routing problem into sub-problems of manageable size. There are two types of hierarchy on the routing graph—top-down and bottom-up approach. Figure 8.24 shows a routing graph based on cut tree. Each interior node in the cut tree represents a primitive global routing problem. Here, each sub-problem is solved optimally by translating it into an integer programming problem. The partial solutions are found by using integer programming.

We consider the root of the hierarchical structure to be *T* at level-1 and the leaves of *T* are at level-*h*, where *h* is the height of *T*. In case of top-down approach, the routing is made step by step from level-1 to level–*h*. At level-*i*, the floor patterns corresponding to nodes larger than *i* are deleted. A solution is obtained for each updated routing graph which is associated with nodes at level-*i*. Each solution is combined with solution at level $(i - 1)$. The step refines the routing to cover one or more levels and it reaches the highest level *h* to get a trial solution. The description of top-down approach is given below:

Procedure: TOP _ DOWN _ ROUTING

```
begin-1
  Complete routing R1 solution to the level -1
  for i = 2 to h do
  begin-2
  for all nodes n at level (i-1) do
  Complete the solution Rn of the routing problem Ln
  Combine the solutions Rn for all the nodes n
  and the solution Ri-1 into Ri
end-2
end-1
```

Combining the solution of one level into that of the next level is a crucial step in this approach. Bottom-up approach for hierarchical technique uses the partial routing combined by processing tree nodes in the bottom-up manner. In this case, each net that runs through the cut level must be interconnected (while maintaining the capacity of the constraints) when the results of two nodes originating from the same node are combined.

Procedure: Bottom-Up approach

```
begin-1
  Complete the solution to the level-K (K is the highest level)
  for i = k to 1 do
  begin-2
  for all the nodes n at level (i-1), then
  begin-3
  Complete the solution Rn of the routing problem Ln
  by combining the solution to the children of node n
end-3
end-2
end-1
```

3. Randomized Routing

Randomized routing is based on the integer linear program formulation, where integral constraint is omitted and is converted to a new linear relaxation problem. The next step is to obtain integer solutions which are close to optimal solution. The steps of this type of routing are given below:

Step-1: Obtain a solution to the routing problem R with removal of integral constraint and let the solution be $x = \alpha$.

Step-2: Use the probability α_i of the variable x_i for solution 1. For solution 0, the probability is $1 - \alpha_i$.

Step-3: Repeat Step-2 for creating another solution.

Step-4: Choose the best solution with highest probability.

$E(X_i)$ = Expected value of variable $x_i = \alpha_i$

The objective function is min $C(X)$

$$\text{where } C(X) = \sum_{i=1}^{n} (C_i X_i) .$$

If ψ is the cost solution after a single evaluation

$$\psi = \sum_{i=1}^{n} (C_i X_i)$$

The expected value of ψ

$$E(\psi) = E[\sum_{i=1}^{n} C_i X_i] = \sum_{i=1}^{n} C_i E(X_i) = \sum_{i=1}^{n} C_i \alpha_i$$

Thus, the expected value of ψ is the optional cost of linear relaxation of the routing problem.

4. Integer Linear Programming

The total routing area consists of the nets defined in terms of parameter multiplicity (the number of terminals). If a net n, has multiplicity of K_n then the net n is defined as a set $n = \{ n_1, n_2 \ldots . n_k \}$. T_n^j is a route available to the net n. Each net n is labeled with a cost factor, $w(n) \geq 0$.

$x_{n,j}$ is a variable for each net and route T_n^j

$x_n^j = 1$ for the net n uses the route T_n^j

$= 0$

The load on the edge e, is defined as

$$U(x, e) = \sum \sum W(n) x_n^j$$

E_n is the set of edges of the following graph:

$$w(x) = \sum_{e \in E} l(e) U(X, e)$$

where $l(e)$ = Length of edge, e. There are two basic conditions to formulate the problem in this approach. In the first type, the capacities of edges are considered. In the second type, the capacities of edges are ignored and a relaxed version of the problem is solved.

(a) Constrained Routing Condition

$$x_n^j \in (0, 1) \text{ for all } n \text{ and } j$$

$$\sum_{j=1}^{l_n} (x_{n,j}) \leq 1 \text{ for all nets}$$

$$U(X, e) \leq C(e) \text{ for all edges } e \in E$$

The first two constraints show that one admissible route is chosen for each net. The constraints show the capacity constraints for all edges. The main objective is to minimize the wire length and minimize the number of nets routed at the same time. Thus, the following cost function is to be minimised with linear combination of these parameters.

$$C = \lambda \sum_{n \in N} W(n)(1 - \sum_{j=1}^{l_n} x_n^j) + W(n)$$

(b) Unconstrained Global Routing Condition The capacity constraints are eliminated

$$\sum_j x_n^j = 1$$

$$U(e)/C(e) \leq X_L \text{ for all } e \in E$$

where x_L is maximum load on any edge.
The cost function is *w*ritten as

$$C = \lambda x_L + W(n)$$

where λ = Scaling factor.

By considering these conditions, routing nets with all edges are evaluated. The main disadvantage of this approach is that it is extremely slow in comparison to other approaches.

5. One-step Approach

The one-step approach involves the decomposition of the chip area in the form of $n \times n$ matrix by horizontal lines and then to use one or more terminals depending on the restrictions. For a routing, R, $w(R)$ denotes the maximum number of wires passing from one cell to adjacent ones in the routing R. Here, minimum $w(R)$ is considered in optimal global routing for a given problem P. The $w(P)$ denotes the diversity/width of optional routing R that provides a solution of the problem P. $w(n)$ is the maximum diversity or width of the $n \times n$ matrix decomposition. Cut (P) denotes the maximum number of nets crossing the boundary of the cell and P is the number of nets connected to a terminal. So λ is defined as $\lambda = \dfrac{\text{Cut}(P)}{P}$

In this case, the routing is made as follows:

1. Divide the chip into squares whose sides are λ.
2. Route these squares independently in arbitrary one-turn manner with width at most 0 (Cut (P)) and next route nets have a square arbitrary at a point on the perimeter of square.
3. Proceed with Step-2 through bottom-up recursion.

Figure 8.24 shows global routing using one-step approach. It is an extremely fast algorithm but not very effective in terms of performance of global routing. It requires combination of effective heuristics for practical implementation.

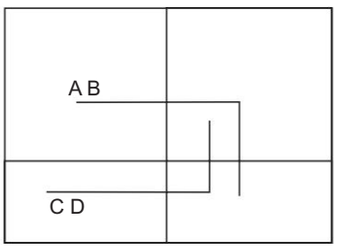


Fig. 8.24 One-step approach 2 × 2 array

8.5.5 Detailed Routing

The more useful approach for slowing the detail routing problem is Lee–Moore's maze-running algorithm which was discussed earlier. It is also possible to bypass the global routing stage, and detail routing can be started for routing of the entire chip area by using Lee–Moore's maze-running algorithm. But the two-stage routing approach of global routing followed by detail routing is the most commonly used and a powerful technique for realizing interconnections in VLSI circuits. In its global stage, the method first partitions the routing region into a collection of disjoint rectilinear sub-regions. Typically, the routing region is decomposed into a collection of rectangles. Then, each sub-region is interconnected to each other by using floating terminals in which all the nets cross a given boundary of the routing sub-region. Once all the floating terminals are fixed after routing of all sub-regions, each sub-region that is inside is routed using two kinds of methods—channels and switch boxes.

The **channels** refer to the routing regions having two parallel rows of fixed terminals, whereas **switch boxes** are generalization of channels that allow fixed terminals over all four sides of the region. In detailed routing, channels and switch-box routers perform completion of connections for routing.

1. Channel Router

Channel router is based on two share channels in which a routing region is bounded by two parallel boundaries. For a horizontal channel, fixed terminals are located on the upper and lower boundaries and floating terminals are allowed on left and right ends. So, the channel routing is to route a specified net list between two rows of terminals, as shown in Fig. 8.25.

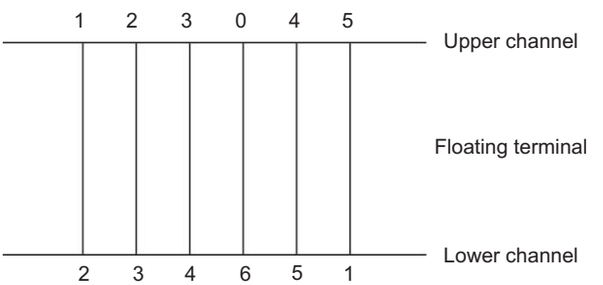


Fig. 8.25 Channel router with two share channels

When the channel length is fixed, the area goal is to minimize the channel width. The channel routing problem is the channel width, which is formulated as follows: Given a collection of nets = $\{N_1, N_2 \ldots N_n\}$, connect them while keeping the channel width minimum. The problem is given below:

1. The input consists of two rows of terminals—upper boundary channel and lower boundary channel.
   Top = $t(1)$, $t(2)$ ---------- $t(n)$ = Set of top terminals.
   BOT = Set of bottom terminals = $b(1)$, $b(2)$ ---------- $b(n)$.
2. The output consists of Steiner nets with vertical/horizontal overlaps and minimum number of bending.
3. The goal is to minimize number of tracks.

The channel routers use the following algorithms for routing—left-edge algorithm, yet another algorithm, greedy channel routing and hierarchical routing.

(a) Left-edge Channel Routing The left-edge channel routers use top-down row-by-row approach. If a top terminal and bottom terminal have the same abscissa, they are connected to a distinct net. The horizontal segments of the net connected to the top terminal should be above the horizontal segment of the bottom terminal. This algorithm gives the routing solution with minimum number of possible tracks which provides no vertical constraint-related obstacles.

(b) Yet Another Channel Router Yet Another Channel Router (YACR) operates under the assumption that vertical tracks are added whenever needed within a channel. It allows the addition of horizontal tracks and introduction of horizontal jogs on a vertical layer which may remit in wire overlap. This approach to handle vertical constraints was introduced in YACR-II. Figure 8.26 shows YACR-II having vertical constraints and resolution with Maze I pattern.

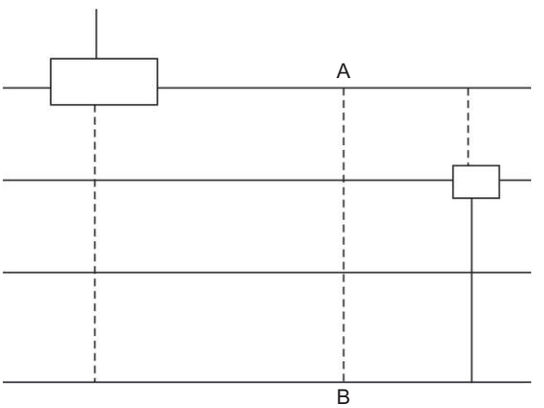


Fig. 8.26 YACR–II with vertical constraints

The tracks are defined as horizontal wire segments placed in tracks and branches are vertical wires connecting trunks to the top and bottom of the channel. The router has two phase approaches:

1. In the first phase, a vertical constraint graph is generated for finding tracks. If there is a conflict in the graph, it goes to Step-2.
2. The branch-layer routing assignments are placed for all columns that do not violate vertical constraints.

(c) Greedy Channel Router The greedy channel router is one of the popular channel routers which is commonly used. In the greedy channel router, the routing is made from left to right, in a column-by-column manner completing wiring within a given column before proceeding to the next. In each column, the router tries to optimize the utilization of wiring tracks in a greedy fashion in the following steps:

1. Make feasible connections to any terminal at the top and bottom of the column and bring the nets safely to the first track.
2. Free up as many tracks as possible and make vertical jogs to collapse the nets that occupy more than one track.
3. Shrink the range of tracks occupied by nets still occupying more than one track. Add dog logs to reduce the range of split nets by bringing the nets to an empty track.
4. Introduce the jog to move the net to an empty track close to the boundary of its target terminal. This tends to maximize utilization of vertical wiring so that it reduces column congestion.
5. Add a new track. If the terminal cannot be connected up in Step-1 because the channel is full then extend the channel by inserting a new track between existing tracks.
6. Extend processing to next column when the processing of the current column is complete. The router extends the wiring to the next till all the terminals are connected for routing.

The router starts with the number of tracks equal to channel density and if there is congestion in the column then add tracks and extend processing of the column from left to right.

(d) Hierarchical Router Hierarchical decision-making approach is used to handle large-scale routing problems. It is applied at each level of the hierarchy to consider all the nets at once. Two schemes have been used in this direction—top-down and bottom-up approach. In bottom up, the chip area is cut into square cells which are small enough to handle and then paste cells are placed successively after routing of each cell.

Figure 8.27 shows the top-down approach used for hierarchical routing. It starts from the top with 2×2 super cells (representing the whole chip) which are first routing cells. The next level of hierarchy is considered to be horizontal first and then, vertical hierarchy is considered next. Necessary connections across the boundary are made.

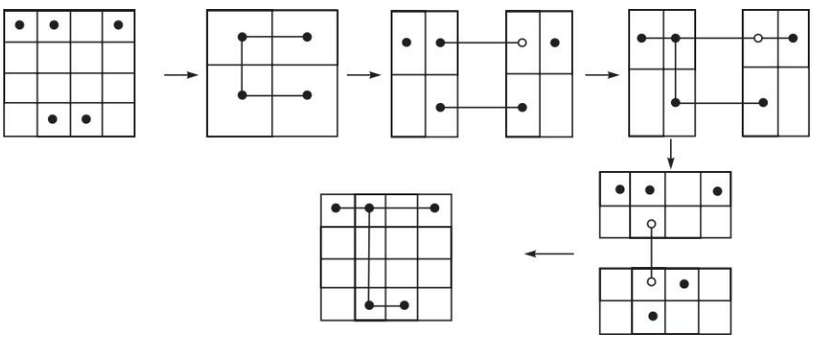


Fig. 8.27 An example of hierarchical top-down approach

2. Switch Box Routing

The routing region with fixed terminals on four sides makes a switch box. Switch boxes formulation in the routing area is called building style routing. The objective of a switch-box router is to interconnect all the terminals belonging to the same net with minimum length and via mode. Although hierarchical channel routing approach makes the routing net quick due to its high speed, it cannot provide minimum total length as provided by the switch-box router. There are two switch-box routing schemes which are effective—beaver and greedy switch-box routing.

(a) Beaver Switch-box Router The beaver switch-box routing algorithm consists of three successive parts—Corner routing, line-sweep routing, and thread routing. All three sub-routers have priority queue of nets to route. **Priority queue** is used to determine the order that the nets are routed to prevent routing conflicts.

The corner router connects terminals that make a corner connection in which a connection is formed by two terminals if

- they belong to the same net,
- they lie on adjacent sides of the switch box, or
- there are no terminals belonging to the net that lies between them on the adjacent sides.

The net has terminals on either two or three sides of the switch box. For corner connection, the ordering is performed for four corner nets. If the overlap cycle occurs for corner connection, four terminal cycles are used as shown in Fig. 8.29.

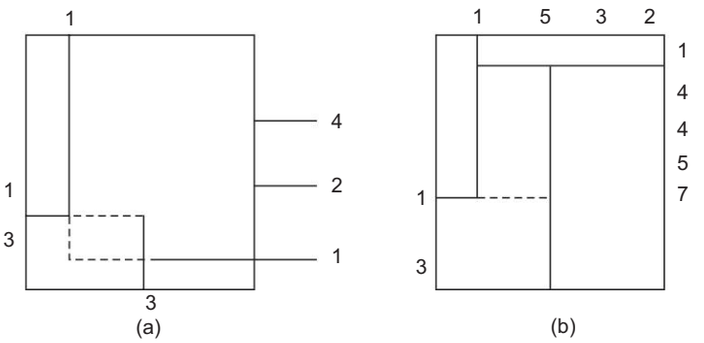


Fig. 8.28 (a) Overlap cycle (b) Four-terminal cycle

A four-terminal cycle occurs when a four-terminal net has its terminals positioned at four sides as shown in Fig. 8.28. The line-sweeper router is an adaptation of the computational geometry technique of plan sweeping. The line-sweep priority queue is initialized with unrealized nets. The line-sweep router use five types of wire connections—single bend, single straight-line wire, dogleg wire, horseshoe consisting of three wires, and staircase consisting of three wires.

The thread router is a maze-type router that does not restrict its search for a connection to any preferential form. This router performs minimum-length connections to realize the remaining unconnected nets. Since the thread router has no restriction in its connection preference, it makes a connection for a net if it exists. It is based on maze-running algorithm. To remove the routing conflicts, the track control is used in this approach. The algorithm procedure for beaver's approach is given below.

Beaver's approach

```
begin-I
initialized control information;
initialized corner—pq;
connect route;
if-2, there are unrealized nets then
initialize line sweep—pq;
line sweep route;
if-3, there are unrealized nets then
relax control constraints;
reinitialize line sweep—pq;
line sweep route;
if-4, there are unrealized nets then
initialize thread pq;
thread route;
end if-4;
end if-3;
end if-2;
perform layer assignment;
end-1;
```

Figure 8.29 shows beaver's with box-routing solution for a chip.

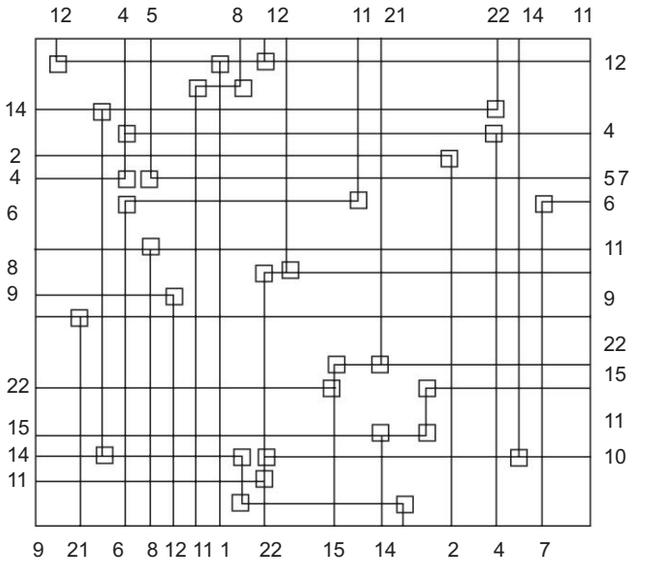


Fig. 8.29 Beaver switch-box solution of routing

(b) Greedy Switch-box Router The greedy switch-box router is a two step method—first, it scans the switch box from left to right, column by column or bottom to top and row by row, and then it takes action according to prioritized method at each column before proceeding to the next. The algorithm procedure is given below.

```
begin-I
initialize the left side of the switch box;
determine goal tracks;
column count = 1;
while-2 (goal tracks not reached) and (column count ≤ maxcol) do
greedy route column;
column count = column count + 1;
end-2;
perform layer assignment;
end-1;
```

8.6 □ PERFORMANCE IN CIRCUIT LAYOUT

Recent advances in VLSI technology allow the fabrication of more than 10^6 transistors in a single chip. Before making layout of a VLSI chip, it is required to test layout performances. In this direction, we consider interconnection delay problem via minimization problem, power minimization, and training-driven routing problem which are required to be solved.

8.6.1 Delay Models

Delay models of ICs are based on timing-analysis methods which are analog simulation and timing simulation for digital circuits. Analog simulation is based on IV characteristics of the devices. Worse-case delay analog need finding of worse-case input patterns. The analysis is done by using numerical integration for analog circuits. For most cases, timing analysis is made for digital circuits in which gate delay and interconnection models are used.

1. Gate-delay Model

The simplest gate-delay model is based on propagation delay or gate delay T_D from a gate input to a gate output. Figure 8.30 shows gate-delay model consisting of input gate loading or gate capacitance, C_g, output gate loading capacitance, C_g, and the gate output on resistance, R_{on}.

The gate output begins changing at twice $t + T_D$. If the driving load capacitance is C_L then the rise time of the output is $\tau = R_{\text{on}} (C_{\text{d}} + C_{\text{L}})$. The on resistance R_{on} is a function of transistor length L and width w and is given by

$$R_{\text{on}} \approx \frac{1}{\mu C_{\text{gox}} (V_{\text{DD}} - V_{\text{T}})} \cdot \frac{L}{W}$$

where μ = Conductive constant, and C_{gox} = Thickness of gate's dielectric oxide. The worst case delay analysis of the gate needs two steps:

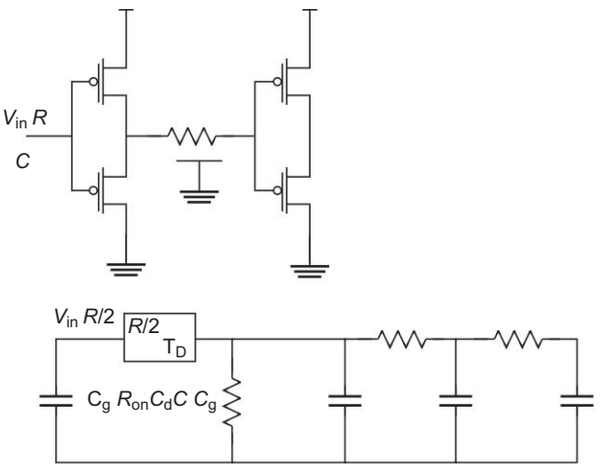


Fig. 8.30 Gate-delay model of simple CMOS inverter circuit

1. Find the longest delay path in the circuit. The delay path is the path from output of the gate to the power supply.
2. Given the latest arriving signal, convert the longest resistive path into an equivalent signal on resistance model.

2. Interconnect Delay

Interconnect delay is a result of parasitic effects which depends on IC wiring structure. Figure 8.31 shows three-dimensional views of a wire and an electrical model of parasitic effect of the wire. Interconnect capacitance C_W arisen from three components—area referred as parallel plate capacitance and the fringing field component. For a wire of width w and length l, dielectric thickness T_{ox}, F_r = Fringing factor, the wire capacitance C_W is given by $C_W = \frac{\delta_{ox}.W.l}{t_{ox}}.f_r$, δ_{ox} = Dielectric constant interconnect.

Resistance R_W which arises from receptivity ρ is given by $R_W = \frac{\rho l}{wh}$ and other effects are interconnect

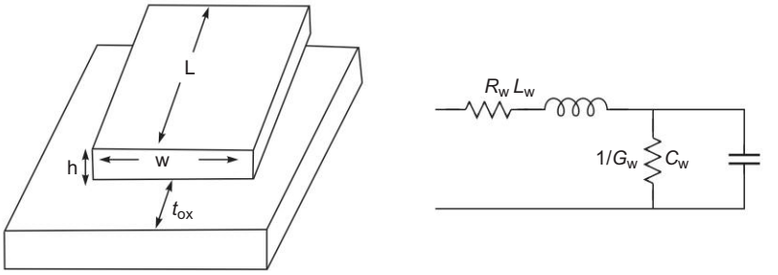


Fig. 8.31 Three-dimensional view of wire and electrical model of parasitic effect of wire

inductive L_W and shunt conductance G_W which are negligible in IC technology. From the distributed transmission-line model, the voltage response in frequency domain is written as

$$V(S) = \frac{1}{S\,\mathrm{Cosh}\sqrt{S\,R_w C_w}}$$

where $S = j\,f$ and f = Frequency

A simple approximation of time constant for a wire gives:

$$T_r = R_w\,C_w = \frac{\delta_{ox}.W.l}{t_{ox}}.\frac{\rho l}{wh}\,f_r = \frac{\delta_{ox}\,Pl^2}{t_{ox}h}\,f_r$$

Total interconnection time delay is written as

$$T_{id} = \frac{1}{2}R_w\,C_w + R_w\,C_L = \frac{1}{2}\frac{\delta_{ox}\,\rho\,l^2}{t_{ox}\,h}\,f_r + \frac{\rho l}{Wh}C_L$$

where C_L = Load capacitance.

8.6.2 Via Minimization

Via (or contact holes) between different layers of interconnection on dense integrated circuits can reduce production yield, degrade circuit performance (enhancement of propagation delay) and occupy a large amount of chip area. There are two types of via minimization problem—constrained via minimization and unconstrained via minimization.

1. Constrained Via Minimization

The objective is to find the position for via, such that no two wires of different nets intersect in the same layer and the numbers of via is minimum. The problem can be solved by using crossing graph/crossed edge. A crossed graph is one where the vertices represent the intersection of nets and an edge is a representation between two adjacent intersections. Figure 8.32 (a) shows constraint via minimization, whereas Fig. 8.32(b) is the crossed graph of Fig. 8.32 (a). The faces of the graph are indicated by *a*, *b*, *c*, *d*, *e*, *f*, and *g* in which *g* is the infinite face. A face with an odd number of edges in its boundary in *G* is an **odd-cycle face** and a face with even number of edges is called an **even-cycle face**. For each odd-cycle face, at least one via is needed whereas for each even-cycle face, one via is added with an odd-cycle face. To find a cross graph with even set of path interconnecting, a pair of odd cycle vertices is to be set up and the total number of edges is to be minimized. Such a pair of odd cycle vertices in

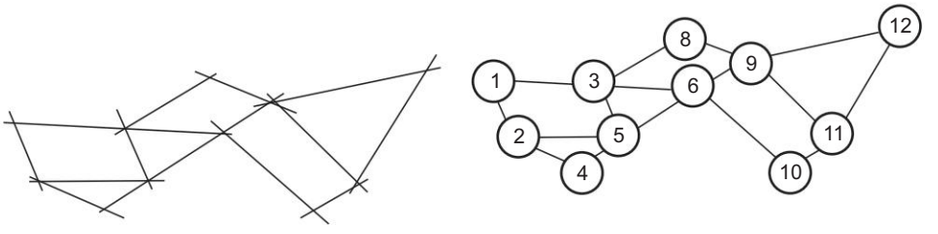


Fig. 8.32 (a) Constrained via minimization (b) Crossed graph

a graph represents an edge between two vertices with a weight where weight is the number of edges of shortest path between corresponding vertices. A minimum-weight matching algorithm is applied to find a matching pair of odd-cycle faces and via is placed along the paths that connect each pair of odd-cycle faces. After getting a crossed graph, it is needed to have clustering where clusters are formed and assigned to different layout layers with minimum weight. The graph formed by clustering is called a clustered graph.

The algorithm procedure is given below:

```
Algorithm Max Cut (L)
begin-I
construct cluster graph G* of L;
construct the triangular graph Ge of G* by adding new edges to G*;
construct the geometric dual graph Ge of Ge;
construct the graph from Gd replaced by a star;
find a minimum complete matching M of G/
end-I
```

2. Unconstrained Via Minimization

In case of unconstrained via minimization, the routing of wires and layer assignment are made simultaneously. Here, geometric mapping is required. Figure 8.33 shows an example of geometric mapping. The geometrical mapping can be made by using a graph in which a two-share-channel UVM problem is formulated to get an optimal solution. In a UVM problem, it is necessary to find the maximum set of nets U, such that each net NEU can be placed entirely on layer-1 or layer-2. Then a topology is obtained by using one via for each net NESU or a total $K = |S| - |U|$ via.

The unconstrained via minimization can be solved by using the K-layer UVM method in which K routing layers are used to find routing with minimum numbers of via.

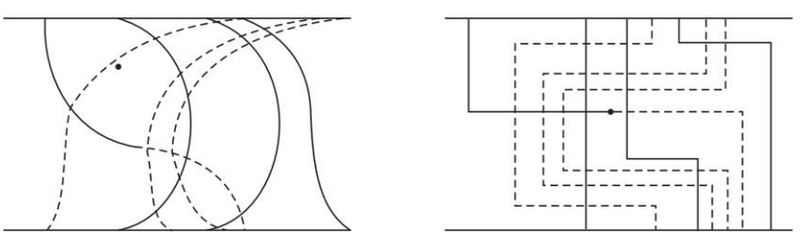


Fig. 8.33 The net without via and geometrical mapping

8.6.3 Power Minimization

Power consumption in CMOS circuits occur due to charging and discharging capacitive loads during output transitions at gates, short-circuit current that flows during output transitions at gates, and the leakage current. This power consumption is dynamic in nature. The short-circuit current and leakage current can be made sufficiently small with proper device and circuit-design techniques. The average dynamic power consumption by CMOS gate is given below.

$$P_{aw} = 0.5 \frac{V_{dd}^2}{T_{cycle}} C_l N = 0.5 \frac{V_{dd}^2}{T_{cycle}} (C_g + C_W) N$$

where C_L is Load capacitance at the output of the node, V_{dd} is supply voltage, T_{cycle} is clock period, N is Number of output transitions at the gate, C_g is load capacitance at input of fan-out gates and C_W is Load capacitance due to the interconnection tree formed between the driver and its fan-out gates. In logic synthesis for low power, minimization of $\sum C_{w_i} N_i$ is obtained and in physical design, minimization of $\sum C_{w_i} N_i$ is made where the suffix i represents i^{th} CMOS. The objective functions for power minimization are based on cell-placement algorithm where total wire length and wire density are considered as performance parameters. In power-consumption formula, constant factor is $0.5 \frac{V_{dd}^2}{T_{cycle}}$ and objective function $L = \sum C_{w_i} N_i$ and the quadratic function is used as follows:

$$L_2 = \sum_i C_{wi}^2 N_i^2$$

The quadratic function is combined with iterative partitioning of the circuit. After each global optimization step, the circuit is further partitioned and the partitioned information is introduced in global optimization technique as a constraint.

8.6.4 Timing-driven Routing

Timing-driven placement has become a fundamental problem in which timing issues are given importance at the placement level. Later, timing-driven routing can improve circuit performance. For timing-driven routing, delay minimization can be used. The procedure for delay minimization is more effective in detailed routing. The minimum-spanning-tree technique is used for delay minimization. Here, we consider radius R of a signal net which is the length of the shortest path from source S to for that sink x., where R = Max distance (S, X). To find radius R and length of routing paths, **Bounded Radius Minimum Routing Tree (BRMST)** is used. A formal description of algorithm for LAYOUT-BRMST is given below:

Procedure LAYOUT-BRMST

```
begin-I
Compact MST_P;
E = edges of MST;
Q = (V,E);
L = depth-first tour of MST_P,
S = 0;
for i = 1 to |L|- 1 do
begin-2
S = S + distance (X_i, x_i + i);
if S ≥ ε × dist(S, X_i + 1), then
begin-3
E = E ∪ min path (S, x_i + 1);
```

```
S = 0;
end-3
end-2
T = shortest path tree of Q;
end-1
```

An example of LAYOUT-BRMST is shown in Fig. 8.34. Figure 8.34(a) shows input paints and MST. Figure 8.34 (b) represents LAYOUT–BRMST where shortest distance between source and input paint v_2 is found as a radius. For high-speed digital system, clock period determines the rate of processing. A clock network is required for distribution of clock signal from a clock generator to synchronizing components. For clock signal distribution, the following parameters have to be minimized:

1. Clock skew which is defined as maximum difference of delays from clock source to clock pins
2. Clock phase delay which is defined as true maximum delay from the clock source to clock pin
3. Clock rise time (skew rate) of the signals at clock pins defined as the time taken by the waveform from a V_{L0} to V_{H1} value.
4. Sensitivity to clock skew, clock rise time, and clock phase delay.

In case of a processor consisting of digital circuit clock, networks are designed properly to get minimization system resources such as power and area. The buffered clocked-tree technique is used to taid minimization of the above parameters. This approach constructs clock network. It partitions the clock tree into sections using buffers which are used in its source paths. Figure 8.35 shows the buffer clock tree.

The clock network construction problem presents a trade-off between wire length and skew. Here, trade-offs present a challenge to the designer and to those seeking to automate the clock-design process.

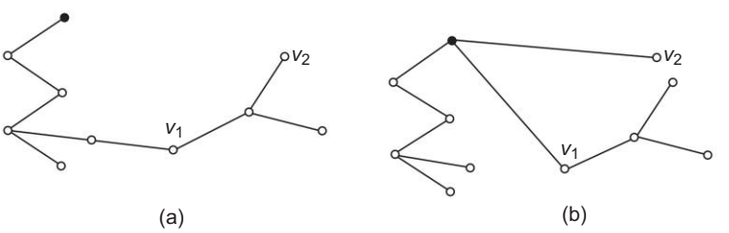


Fig. 8.34 (a) Input points and MST (b) Layout-BRMST

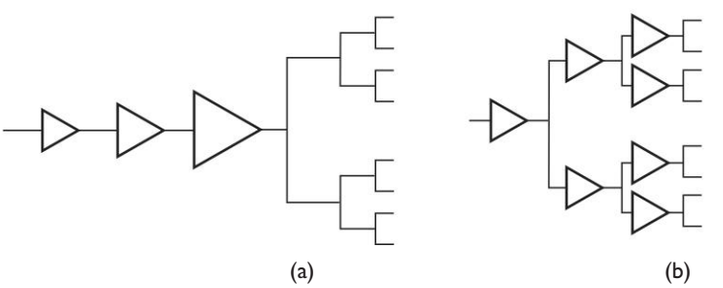


Fig. 8.35 (a) Buffer chain driving clock tree (b) Buffer clock power-up tree

□ REFERENCES

8.1. Agarwal, P.K. and M.T. Shing, Algorithm for Special Cases of Rectilinear Steiner Trees: I. Points on the Boundary of a Rectilinear Rectangle, Networks 20(4):453–485, 1990.

8.2. Aho, A.V., M.R. Garey, and F.K. Hwang, 'Rectilinear Steiner Trees: Efficient Special-Case Algorithm, Networks 7:35–58, 1977.

8.3. Aho, A.V., J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, MA, 1974.

8.4. Akama, T., H. Suzuki, and T. Nishizeki, *Finding Steiner Forests in Planar Graphs,* in The First Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 444–453, ACM, 1990.

8.5. Akers, S.B., M.E. Geyer, and D.L. Roberts, *IC Mask Layout with a Single Conductor Layer*, in Design Automation Conference, pp. 7–16, IEEE/ACM, 1970.

8.6. Antreich, K.J., F.M. Johannes, and F.H. Kirsch, *A New Approach for Solving the Placement Problem Using Force Models*, in International Symposium on Circuits and Systems, pp. 481–486, IEEE, 1982.

8.7. Anway, H., G. Farnham, and R. Reid, *Plint Layout System for VLSI Chip Design*, in Design Automation Conference, pp. 449–452, IEEE/ACM, 1985.

8.8. Asano, T., and H. Imai, *Partioning a Polygon Region into Trapezoids*, Association for Computing Machinery 33(2):290–312, 1986.

8.9. Baker, B.S., S.N. Bhatt, and F.T. Leighton, *An Approximation Algorithm for Manhattan Routing*, in Proc. 15th Annual Symp. Theory of Computing, pp. 477–486, ACM; 1983.

8.10. Bakoglu, H.B., *Circuits, Interconnections, and Packaging for VLSI,* Addison-Wesley, Reading, MA, 1990.

8.11. Barnes, E.R., *An Algorithm for Partitioning the Nodes of a Graph*, Technical report, IBM T.J. Watson Research Center, Dept. Comput. Sci., 1981.

8.12. Bentley, J.L., and T. Ottmann, *Algorithm for Reporting and Counting Geometric Intersections,* IEEE Transactions on Computers, C-28:643–647, 1979.

8.13. Berger, B., M.L. Brady, D.J. Brown, and T. Leighton *Nearly Optimal Algorithms and Bounds for Multilayer Channel Routing*, unpublished paper, 1986.

8.14. Bhasker, J., and S. Sahni, *A Linear Algorithm to Find a Rectangular Dual of a Planar Triangulated Graph,* Algorithmica 3(2):274–278, 1988.

8.15. Bhat, N., and D. Hill, *Routable Technology Mapping for LUT-Based FPGA's*, in International Conference on Computer Design, pp. 95–98, IEEE, 1992.

8.16. Blanks, J.P. *Near Optimal Placement Using a Quadratic Objective Function,* in Design Automation Conference, pp. 609–615, IEEE/ACM, 1985.

8.17. Blodgett, A.J., *Microelectronic Packaging,* Scientific American, (July):86–96, 1983.

8.18. Brady, M.L., and D.J. Brown, *Optimal Multilayer Channel Routing with Overlap,* in Fourth MIT Conference on Advanced Research in VLSI, pp. 281–296, MIT Press, Cambridge, MA, 1986.

8.19. Brayton, R.K., C. McMullen, G.D. Hachtel, and A. Sangiovanni-Vincentelli, Logic Minimization Algorithms for VLSI Synthesis, Kluwer Academic Publishers, Boston, MA, 1984.

8.20. Breuer, M.A., *A Class of Min-cut Placement Algorithms*, in Design Automation Conference, pp. 284–290, IEEE/ACM, 1977.

8.21. Brown, S., J. Rose, and Z. Vranesic, *A Detail Router for Field-Programmable Gate Arrays*, in International Conference on Computer-Aided Design, pp. 382–385, IEEE/ACM, 1990.

8.22. Brown, S., J. Rose, and Z. Vranesic, *A Detailed Router for Field-Programmable Gate Arrays*, IEEE Transactions on Computer Aided Design 11:620–628, 1992.

8.23. Brown, S., J. Rose, and Z. Vranesic, *A Stochastic Model to Predict the Routability of Field-Programmable Gate Arrays,* IEEE Transactions on Computer Aided Design 12:1827–1838, 1993.

8.24. Burstein, M., and R. Pelavin, *Hierarchical Channel Router,* Integration: The VLSI Journal, 1, 1983 (also published in Proc. 20th Design Automation Conference, 1, 1983).

8.25. Burstein, M., and R. Pelavin, *Hierarchical wire Routing,* IEEE Transactions on Computer-Aided Design, CAD-2(4):223–234, 1983.

8.26. Carden, R.C., IV, and C.K. Cheng, *A Global Router Using an Efficient Approximate Multicommodity Multiterminal Flow Algorithm,* in Design Automation Conference, pp. 316–321, IEEE/ACM, 1991.

8.27. Charney, H.R., and D.L. Plato, *Efficient Partitioning of Components,* in Design Automation workshop, pp. 16.0–16.21, IEEE, 1968.

8.28. Chen, H.H., and C.K. Wong, *Wiring and Crosstalk Avoidance in Multi-Chip Module Design,* in IEEE Custom Integrated Circuits Conference, IEEE, 1992.

8.29. Cheng, C.K., and E.S. Kuh, *Module Placement Based on Resistive Network Optimization,* IEEE Transactions on Computer Aided Design 3(3):218–225, 1984.

8.30. Cheng, C.K., and Y.C. Wei, *An Improved Two-Way Partitioning Algorithm with Stable Performance,* IEEE Transactions on Computer Aided Design 10(12):1502–1511, 1991.

8.31. Chiang, C., M. Sarrafzadeh, and C.K. Wong: *A weighted-Steiner-Tree-Based Global Router with Simultaneous Length and Density Minimization,* IEEE Trans. on CAD/ICS 13(12):1461–1469, 1994.

8.32. Chiang, C., M. Sarrafzadeh, and C.K. Wong: *An Optimal Algorithm for Constructing a Steiner Tree in a Switchbox (Part 1: Fundamental Theory and Application),* IEEE Transactions on Circuits and Systems 39(6):551–563, 1992.

8.33. Cho, J.D., and M. Sarrafzadeh, *A Buffer Distribution Algorithm for High-Speed Clock Routing*, in Design Automation Conference, pp. 537–543, IEEE/ACM, 1993.

8.34. Chyan, D., and M.A. Breuer, *A Placement Algorithm for Array Processors,* in Design Automation Conference, pp. 182 –188, IEEE/ACM, 1983.

8.35. Cohoon, J.P., *Distributed Genetic Algorithms for the Floorplan Design Problem,* IEEE Transactions on Computer Aided Design 10(4):483–492, 1991.

8.36. Cohoon, J.P., et al, *Floorplan Design Using Distributed Genetic Algorithms*, in International Conference on Computer-Aided Design, pp. 452–455, IEEE, 1988.

8.37. Cong, J., and Y. Ding, *An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Design,* Technical Report CSD-920022, University of California at Los Angeles, May 1992. (Also published in Proceedings of the ICCAD, 1992.)

8.38. Cong, J., L. Hagen, and A. Kahng, *Net Partitions yield Better Module Partitions*, in Design Automation Conference, pp. 47–52. IEEE, 1992.

8.39. Cong, J., A. Kahng, G. Robins, M. Sarrafzadeh, and C.K. Wong, *Provably Good Performance—Driven Global Routing*, IEEE Transactions on Computer Aided Design 11(6):739–752, 1992.

8.40. Cong, J. and C.L. Liu, *Over-the-Cell Channel Routing,* IEEE Transactions on Computer Aided Design 9(4):408–418, 1990.
8.41. Dai, W.M., and E.S. Kuh, *Simultaneous Floor Planning and Global Routing for Hierarchical Building-Block Layout*, IEEE Transactions on Computer Aided Design 6(5):828–837, 1987.
8.42. Deutsch, D.N., *A Dogleg Channel Router,* in Design Automation Conference, pp. 425–433, IEEE/ACM, 1976.
8.43. Dunlop, A.E., and B.W. Kernighan, A Procedure for Placement of Standard Cell VLSI Circuits, IEEE Transactions on Computer Aided Design 4(1):92–98, 1985.
8.44. El Gamal, A., J. Greene, and V. Roychowdhury, *Segmented Channel Routing is Nearly as Efficient as Channel Routing,* Proc. Advanced Research in VLSI, pp. 193–211, 1991.
8.45. Friedman, E.G. *Clock Distribution Design in VLSI Circuits—an Overview*, in International Symposium on Circuits and Systems, pp. 1475–1478, IEEE, 1993.
8.46. Gao, S., and M. Kaufmann, *Channel Routing of Multiterminal Nets,* in Proceedings of 28th Annual Symposium on the Foundations of Computer Science, pp. 316–325, IEEE, 1987.
8.47. Greene, J., V. Roychowdhury, S. Kaptanoglu, and A. El Gamal, *Segmented Channel Routing*, in Design Automation Conference, pp. 567–572, IEEE/ACM, 1990.
8.48. Hagen, L., and A.B. Kahng, *A New Approach to Effective Circuit Clustering*, IEEE Transactions on Computer Aided Design 11(9):422–427, 1992.
8.49. Hamachi, G.T., and J.K. Ousterhout, *A Switchbox Router with Obstacle Avoidance*, in Design Automation Conference, pp. 173–179, IEEE/ACM, 1984.
8.50. Hambrusch, S.E., *Channel Routing Algorithm for Overlap Models*, IEEE Transactions on Computer Aided Design CAD-4(1):23–30, 1985.
8.51. Krishnamurthy, B., *An Improved Min-Cut Algorithm for Partitioning VLSI Networks,* IEEE Transactions on Computers C-33:438–446, 1984.
8.52. Krohn, H.E., *An Over-the-Cell Gate Array Channel Router*, in Design Automation Conference, pp. 665–670, IEEE/ACM, 1983.
8.53. Kuhn, H.W., and A.W. Tucker, *Nonlinear Programming,* in Proceedings of the 2nd Berkley Symposium on Mathematical Statistics and Probability, pp. 481–492, University of California Press, Berkeley, 1951.
8.54. Lai, Y.T., and S.M. Leinwand, *Algorithms for Floor-plan Design via Rectangular Dualization*, IEEE Transactions on Computer Aided Design 7(12):1278–1289, 1988.
8.55. Lee, J.F., and D.T. Tang, *VLSI Layout Compactor with Grid and Mixed Constraints,* IEEE Transactions on Computer Aided Design CAD-6(5):903–910, 1987.
8.56. Lee, K.W., and C. Sechen, *A New Global Router for Row-Based Layout*," in International Conference on Computer-Aided Design, pp. 180–183, IEEE, 1988.
8.57. Leiserson, C.E., and F.M. Maley, *Algorithms for Routing and Testing Routability of Planar VLSI Layouts*, in Symposium on the Theory of Computation, pp. 69–78, ACM, 1985.
8.58. Lie, M., and C.S. Homg, *A Bus Router for IC Layout*, in Design Automation Conference, pp. 129–132, IEEE/ACM, 1982.
8.59. Lin, I., and D.H.C. Du, *Performance-Driven Constructive Placement*, in Design Automation Conference, pp. 103–106, IEEE/ACM, 1990.
8.60. Lin, T.M., and C.A. Mead, *Signal Delay in General RC Networks*, IEEE Transactions on Computer Aided Design 3(4):331–349, 1984.

8.61. Luk, W.K., *A Greedy Switchbox Router,* Technical Report CMU-CS-84–148, Carnegie-Mellon University, 1984.

8.62. Luk, W.K., P. Sipala, M. Tamminen, D. Tang, L.S. Woo, and C.K. Wong, *A Hierarchical Global wiring Algorithm for Custom Chip Design*, IEEE Transactions on Computer Aided Design CAD-6(4):518–533, 1987.

8.63. Makedon, F., and S. Tragoudas, *Approximate Solutions for Graph and Hypergraph Partitioning*, in Algorithmic Aspects of VLSI Layouts (M. Sarrafzadeh and D.T. Lee, eds.), pp. 133–166, World Scientific, Singapore, 1993.

8.64. McGeoch, L.A., D.S. Johnson, C.R. Aragon and C. Schevon, *Optimization by Simulated Annealing: An Experimental Evaluation (Part 1),* AT&T Bell Lab., Murray Hill, NJ, 1985.

8.65. Mead, C., and L. Conway, *Introduction to VLSI System*, Addison-Wesley, Reading, MA, 1980.

8.66. Muroga, S., *VLSI System Design,* John Wiley & Sons, New york, 1982.

8.67. Nair, R., *A Simple yet Effective Technique for Global wiring*, IEEE Transactions on Computer Aided Design CAD-6(2):165–172, 1987.

8.68. Natarajan, S., N. Holmes, N.A. Sherwani, and M. Sarrafzadeh, *Over-the-Cell Channel Routing for High Performance Circuits,* in Design Automation Conference pp. 600–603, IEEE/ACM, 1992.

8.69. Otten, R. H. J. M., *Efficient Floorplan Optimization*, in International Conference on Computer Design, pp. 499–503, IEEE/ACM, 1983.

8.70. Palczewski, M., *Plane Parallel A* Maze Router and Its Application,* in Design Automation Conference, pp. 691–697, IEEE/ACM, 1992.

8.71. Pedram, M., B. Nobandegani, and B. Preas, *Design and Analysis of Segmented Routing Channels for Row-Based FPGA's*, IEEE Transactions on Computer Aided Design 13:1470–1479, 1994.

8.72. Preas, B., and M. Lorenzetti, *Physical Design Automation of VLSI Systems,* Benjamin/Cummings, Menlo Park, CA, 1988.

8.73. Preparata, F.P., and W. Lipski, Jr., *Optimal Three-Layer Channel Routing,* IEEE Transactions on Computer Aided Design C-33(5):427–437, 1984.

8.74. Ramanathan, P., and K.G. Shin, *A Clock Distribution Scheme for Non-Symmetric VLSI Circuits,* in International Conference on Computer-Aided Design, pp. 398–401, IEEE/ACM, 1989.

8.75. Rivest, R.L., and C.M. Fiduccia, *A Greedy Channel Router,* in Design Automation Conference, pp. 418–424, IEEE/ACM, 1982.

8.76. Roychowdhury, V.,J. Greene, and A. El Gamal, *Segmented Channel Routing*, IEEE Transactions on Computer Aided Design 12:79–95, 1993.

8.77. Rubinstein, J., P. Penfield, and M.A. Horowitz, *Signal Delay in RC Tree Networks,* IEEE Transactions on Computer Aided Design CAD-2(3):202–211, 1983.

8.78. Sakurai, T., *Approximation of wiring Delay in MOSFET LSI*, IEEE Journal of Solid-State Circuits 18(4):418–426, 1983.

8.79. Sangiovanni-Vincentelli, A., and M. Santomauro, *YACR: Yet Another Channel Router,* in Proc. Custom Integr. Circuits Conf., Rochester, NY, pp. 460–466, IEEE, 1982.

8.80. Sarrafzadeh, M., *Channel-Routing Problem in the Knock-Knee Mode Is NP-Complete*, IEEE Transactions on Computer Aided Design 6(4):503–506, 1987.

8.81. Schuler, D.M., and E.G. Ulrich, *Clustering and Linear Placement,* in Proc. 9th Design Automation workshop, pp. 50–56, ACM, 1972.

8.82. Sechen, C., *VLSI Placement and Global Routing Using Simulated Annealing,* Kluwer, Deventer, The Netherlands, 1988.
8.83. Shahookar, K., and P. Mazumder, *A Genetic Approach to Standard Cell Placement Using Meta-Genetic Parameter Optimization*, IEEE Transactions on Computer Aided Design 9(5):500–511, 1990.
8.84. Shahookar, K., and P. Mazumder, *VLSI Cell Placement Techniques,* ACM Computing Surveys 23(2):143–220, 1991.
8.85. Shargowitz, E., and J. Keel, *A Global Router Based on Multicommodity Flow Model*, Integration: The VLSI Journal 5:3–16, 1987.
8.86. Sherwani, N. A., *Algorithms For VLSI Physical Design Automation,* Kluwer Academic Publishers, Boston, MA, 1993.
8.87. Shih, M., and E.S. Kuh, *Circuit Partitioning under Capacity and I/O Constraints*, in IEEE Custom Integrated Circuits Conference, IEEE, 1994.
8.88. Stockmeyer, L., *Optimal Orientation of Cells in Slicing Floorplan Designs*, Information and Control 57(2):91–101, 1983.
8.89. Szymanski, T.G., *Dogleg Channel Routing is NP-Complete*, IEEE Trans. on CAD 4(l):31–41, 1985.
8.90. Uehara, T., and W.M. van Cleemput, *Optimal Layout of CMOS Functional Arrays,* IEEE Transactions on Computers C-30(5):305–312, 1981.
8.91. Varga, R.S., *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1962.
8.92. wei, Y.C., and C.K. Cheng, *Ratio-Cut Partitioning for Hierachical Designs,* IEEE Transactions on Computer Aided Design, 40(7):911–921, 1991.
8.93. Wong, D.F., H.W. Leong, and C. L. Liu: Multiple PLA Folding by the Method of Simulated Annealing, in Custom Integrated Circuits Conf., pp. 351-355, 1986.
8.94. Wong, D.F., H.W. Leong, and C.L. Liu, *Simulated Annealing for VLSI Design*, Kluwer Academic, Boston, MA, 1988.
8.95. Wong, D.F., and C.L. Liu, *Floorplan Design of VLSI Circuits,* Algorithmica 4:263–291, 1989.
8.96. Yeap, K.H., and M. Sarrafzadeh, *Floorplanning by Graph Dualization: 2-Concave Rectilinear Modules*, 1993.
8.97. Zhu, K., and D.F. Wong, *On Channel Segmentation Design for Row-Based FPGA's*, in International Conference on Computer-Aided Design, pp. 26–29, IEEE, 1992.

¤ EXERCISES

8.1 Consider a hypergraph H, where each hyper-edge interconnects at most three vertices. We model each hyper-edge of degree 3 with three edges of weight 1, on the same set of vertices, to obtain a weighted graph G. Prove that an optimal balanced partitioning of G corresponds to an optimal balanced partitioning of H. Prove that this cannot be done if each edge of H interconnects at the most four vertices.

8.2 Consider a path graph v_1,..., v_n,. Here is, v_i is connected to v_{i+1}, for $1 < i < n - 1$. Apply the Kernighan–Lin algorithm to this graph. As the initial partition, let v_a, for all odd values of a be in one set, and v_b, for all even values of b, be in the other set.

8.3 Convert the following circuit in Fig. P8.1 in to graph $G(V, E)$, where V = Number of vertices and E = number of edges. Find the bipartition of graph by Kernighan-Lin algorithm. Apply ratio-cut algorithm to find the bipartition. Then apply genetic algorithm to find bipartition. Compare the results for the same.

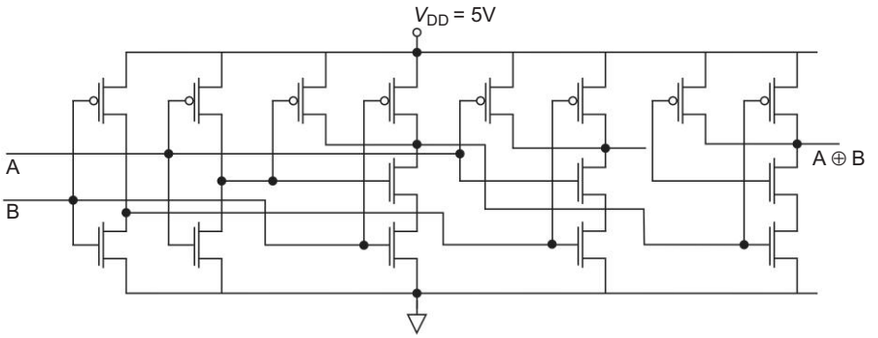


Fig. P8.1

8.4 Consider a complete binary tree with n nodes. Apply the Kernighan–Lin algorithm to this graph. As the initial partition, let v_a for all internal vertices, be in one set and v_b, for all leaves, be in the other set.

8.5 Formulate multi-partitioning genetic algorithm based on cut-size ratio and explain the same with the an example shown in Exercise 8.3.

8.6 Consider a graph with n vertices and maximum degree k. Design an algorithm for partitioning the graph into g groups such that the number of vertices in each group is at most s, and the number of edges connected to each group is at most b. Analyze the quality of your algorithm for different values of k, g, and b. For what values is your algorithm optimal?

8.7 Consider a circuit whose adjacency graph is a complete binary tree with seven nodes. Find an initial placement of the modules, using a constructive force-directed algorithm, in a 3×3 gate array environment. Write a set of nonlinear equations and solve them to find an initial placement of the modules. In your formulation, place the branches of the tree on the four corner modules.

8.8 Design a cost function for the general building-block placement problem which considers the wire length, estimated area, module overlap, and aspect ratio of the entire layout.

8.9 Prove that there is a one-to-one correspondence between a sliceable floor-plan and a normalized Polish expression.

8.10 Given a Polish expression corresponding to a given slicing floor-plan, show that the expression 12-3-... -n- can be reached, and vice versa, using OP1, OP2, and OP3.

8.11 Find an optimal implementation of floor plan of the following modules—M,... $M8$ by using Polish expression. Also, find the optimal sizing for each of the following sliceable floor plans:

$M1$: 4×3

$M2$: 4×5

$M3$: 4×4

$M4$: 3×5

$M5$: 5×6
$M6$: 2×6
$M7$: 5×5
$M8$: 1×5

8.12 Solve the following generalization of the slicing floor-plan sizing problem. Given a slicing tree corresponding to a set of modules, each module has a set of implementations and each implementation is specified by three integers (w, h, p). As before, w and h, respectively, represent the width and the height of the implementation, and p represents the power consumption of the implementation. Design an algorithm that finds an implementation of the modules that minimizes $A + \lambda.P$, where A is the area of the slicing floor plan, P is the power consumption of the floor plan (being the sum of the power consumption of each module), and A is a user-specified constant. Analyze the time complexity of your algorithm.

8.13 Implement the Kernighan–Lin algorithm for a hypergraph. Our goal is to find a balanced partition with minimum cost.

Input format: each input starts with the weight of a hyper-edge followed by the vertices interconnected by it. Specification of the hyper-edges are separated by commas.

3 1 4, (* there is a hyper-edge of weight 3 connecting vertices I and 4 *)
2 1 4 2,
6 2 3 5, 1 4 5, 7 2 3 4

8.14 Use simulated annealing to find a minimum-area slicing floor plan of M. The size and orientation of each module is fixed.

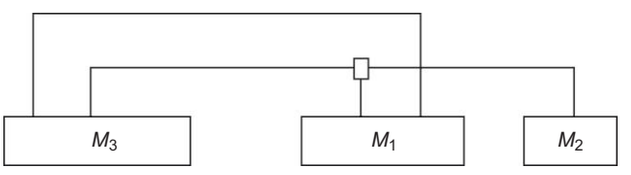
Input format of modules:
2 2, 2 2,
2 1, 2 3, 3 5, 2 4

8.15 Consider a set of modules. Each module has a set of terminals on the upper side of its horizontal edge. Find a linear placement with small density. Draw the modules and nets, and report the density of your solution.

Input format.

3 (* number of modules *)
$M1$ 6, 3 1, 5 4; (* module 1 occupies 6 grid points. At the third grid point, there is a terminal of net 1 and at the fifth grid point, there is a terminal of net 4 *)
$M2$ 4, 2 1 ;
$M3$ 7, 3 1, 2 4 ;

Output format: The output format is shown in Fig. P8.2. Show all nets and their routing.



No. of tracks = 2

Fig. P8.2

8.16 Consider a set of modules in a gate-array environment. Find a placement with minimum cost. The cost of a net is the smallest rectangle enclosing all terminals of a net (the distance between two adjacent modules in the same row is 1). The cost of a solution is the sum of the costs of the nets. Start with a random placement of these modules. Implement an iterative force-directed algorithm that improves the initial placement. Next, start with a better initial placement (not a random one) and apply the same iterative force-directed algorithm to it. Which one performs better?

Input format.

3 (* number of modules *)

*M*1 2 3 (* module 1 is connected to modules 2 and 3 *)

*M*2 1

*M*3 1

Output format: The output format is a gate-array placement as shown in Fig. P8.3. Show all nets and write the total length of your placement.

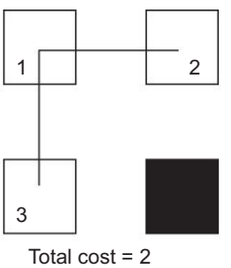


Fig. P8.3

18.17 There are four modules—*a*, *b*, *c*, and *d* and targets—1, 2, 3, and 4. The cost matrix is given by

Module Targets	*a*	*b*	*c*	*d*
1	2	1	2	3
2	1	0	4	2
3	3	2	2	4
4	0	5	1	5

Find the regular placement by using genetic algorithm. Implement regular placement by assignment algorithm.

8.18 what is the running time of Lee's maze router when there is only one two-terminal net in an $n \times n$ grid and the rectilinear distance between the two terminals is d? For what configuration of obstacles is the running time independent of n and dependent only on d?

8.19 What is the running time of the line-searching example? Give an example that takes a long time for the line-searching algorithm to complete.

8.20 Apply the line-searching algorithm to the example shown in Fig. 8.19. Apply the concept of track graphs to the same example.

8.21 Discuss the advantages and disadvantages of maze-running, line-searching, and search-based techniques on track graphs. Emphasize the quality (i.e., the length) and running-time measures.

8.22 Prove that the total weight (i.e., cost) of a minimum-spanning tree in an edge-weighted graph is at most two times the length of an optimal Steiner tree in the same graph.

8.23 A rectilinear Steiner tree consisting of at most k vertical lines is called a k-comb Steiner tree. Design an efficient algorithm for finding a k-comb Steiner tree of a given set of n terminals in the plane. How bad could such a tree be, i.e., what is the maximum ratio of the length of an optimal k-comb Steiner tree to the length of an optimal Steiner tree? Express your result in terms of k and n.

8.24 Consider a set of points where a point is distinguished as source. Design an algorithm for finding a Steiner tree interconnecting all points (including the source) such that the distance between the source and every other point in the tree is small. Elaborate on the quality of your solution.

8.25 Route the following channel consisting of 10 columns using the left-edge algorithm, where 0 indicates an empty position:

TOP = 3 4 0 1 2 4 3 5 2 1

BOT = 1 0 3 0 5 0 4 2 1 5

8.26 Design a greedy algorithm to order the channel in a given placement so as to minimize the number of switchboxes.

8.27 Implement Lee's maze running algorithm. Input format. Input specifies the grid size, position of the two terminals, and the position of the obstacles (the northwest corner is grid).

8 7; (* size of the grid *)

6,6, 2 3; (* positions of source and target *)

3 4, 3 6, 3 1, 1 3,5 4, 5 3, 6 4 ; (* positions of the obstacles *)

8.28 Consider a set of points in a plane. Find a minimum spanning tree interconnecting the points. Then, design an efficient algorithm for finding a Steiner tree connecting the same set of points. Give a table comparing the length of a minimum spanning tree with the length of the resulting Steiner tree, for various values of n, where n is the number of points.

Input format: The input consists of the location of the given points in the plane. 30, 11,23 (* there are 3 points *)

Output format: The output format is shown in Fig. P8.4. The edges of the spanning tree are shown as straight lines. However, their length is a rectilinear length. The edges of the Steiner tree are shown as rectilinear lines (and the distances are also rectilinear).

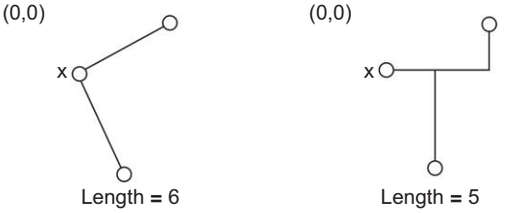


Fig. P8.4

8.29 Design a simulated annealing algorithm for solving the previous problem. Define your moves. Use the same input and output formats as the previous problem. Do you think simulated annealing is suitable for this problem? Explain.

7 6; (* size of the grid *)

5 5, 2 2; (* positions of source and target *)

3 1, 4 3, 1 3, 0 0; (* positions of the obstacles *)

8.30 Implement the left-edge algorithm. Input is the set TOP and BOT (terminals on the top row and the bottom row, respectively).

Input format:

1 2 0 3 Top

2 3 1 0 (* Bottom *)

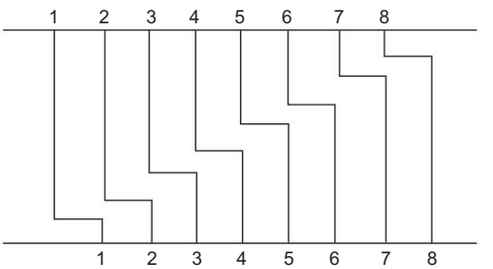


Fig. P8.5

8.31 Solve an instance of the channel-routing problem employing the Greedy algorithm. Use the same input formats as in the previous exercise.

8.32 Consider a set of modules in an FPGA environment. Find placement and routing. The main objective is to find a routing and to minimize the total length of the nets.

Input format:

16 (* number of modules *)

4 (* number of rows and columns of cells *)

N1 2 3, (* net 1 interconnects modules 2 and 3 *)

N2 1 3 5,...

As shown in Fig. P8.6, the number of tracks in each channel is always 5 and the width of each cell is always 8.

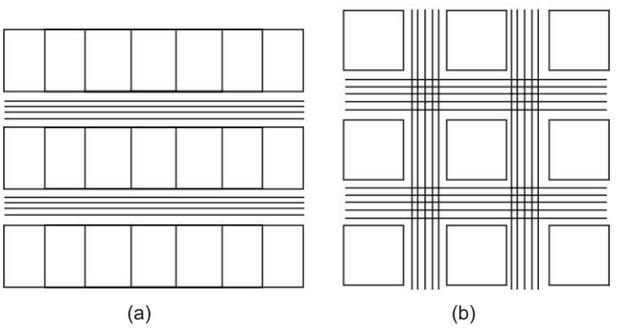


Fig. P8.6

8.33 Consider a set of modules. Each module has a set of terminals on the upper side of its horizontal edge. One terminal per net is specified as the source. Each net is

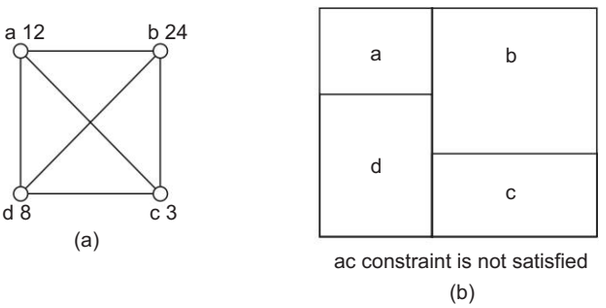


Fig. P8.7

Fig. P8.7 Input and output formats, timing-driven floor planning, assigned a timing constraint; i.e., the length between the source and the sink of the net should be less than the given constraint. Find a linear placement satisfying all timing constraints. Among all such placements, find one with small density.

Input format:

3

*M*1 6, 3 1, 5 4;

*M*2 4, 2 1 ; M3 7, 3 1, 2 4 ;

*N*1 1 7, N2 3, 1 2

The first line indicates that there are 3 modules. The second line indicates that module 1 occupies 6 grid points. At the third grid point, there is a terminal of net 1 and at the fifth grid point, there is a terminal of net 4. The other two modules are similarly specified. Then, it is specified that net 1 (*N*1) has its source on module 2 and its budget length is 7 units, and so on.

Output format: The output format is shown in Fig. P8.8 (grid units corresponding to *M*1 are shown. You do not have to show it in your output). Show all nets and their routing. Highlight portions of nets whose timings are not satisfied.

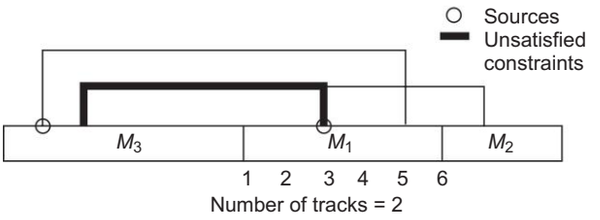


Fig. P8.8

9

Designing of Digital Circuits Using VHDL Programs

Hardware description language is a powerful language which constructs code description of complex control logics. It is a tool by which many complicated digital circuits can be described and designed and implemented without device fabrication. This type of implementation is done in Field Programmable Gate Array (FPGA) and CPLD. It is required to write or represent the logic circuits in any programmable software language or code. The Very high speed Hardware Description Language (VHDL) is one of the languages used for the same. It is an industry standard for description, modeling, and synthesis of digital circuits and system via simulation.

9.1 ☐ DIGITAL DESIGN FLOW BY USING VHDL CODES

Figure 9.1 shows how VHDL code is used for designing and synthesis of digital circuits.

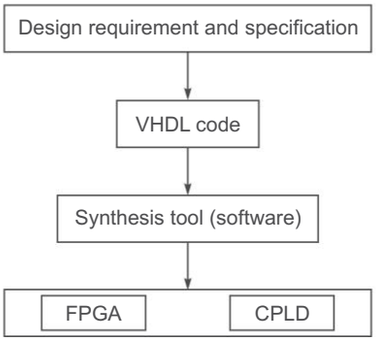


Fig. 9.1 Flow chart of VHDL based design and synthesis

According to design requirements and specifications, the digital circuits are represented with VHDL codes and then synthesized via simulation with a synthesis tool. Then, these are implemented in FPGA or CPLD. The design-tool flow is shown in Fig. 9.2. The inputs of the synthesis software tool are VHDL design source code, synthesis directives, and device selection. Before simulation of VHDL the design codes for digital circuits, the device platform has to be selected for devicespecific synthesis and optimization under synthesis directives. The output of the synthesis software tool provides an

architecture specific netlist or set of equations used as inputs for filter or placing or routing tools which performs tasks for placing and routing. The output of their tools provide information about the resources utilization, point-to-point timing analysis, device programming files (JEDEC format), and post layout simulation model.

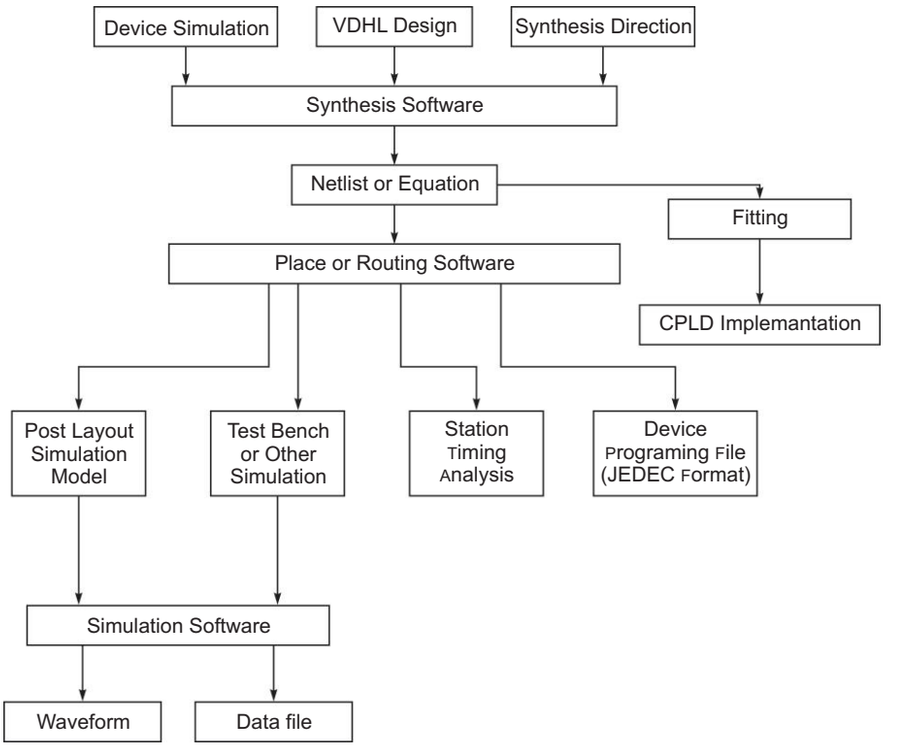


Fig. 9.2 VHDL design tool

Regarding device platform, we will discuss FGPA and CPLD in the next section. After discussion of FPGA and CPLD, VHDL codes will be discussed. One of the primary objectives of VHDL codes is to represent logic design of digital circuits.

9.1.1 Field Programmable Gate Arrays

A field programmable (FPGA) architecture is an array of logic cells that communicate with another end with I/O via wires within routing channels. FPGAs are used for rapid design prototyping and implementation. It consists of prefabricated logic cells, wires and connectors, and switches. Because of their attractive manufacturing cost for low-volume production, FPGA usage has grown rapidly for ASIC implementation.

The logic cell can implement any Boolean logic function of its input. There are two types of logic cell architecture—Look-Up Table (LUT) based cell and Multiplexer (MUX) based cell. In LUT based cell architecture, each logic cell mainly consists of a *K*-input single-output programmable memory capable of implementing any Boolean function of K inputs which follows the truth table.

In MUX based cell architecture, multiplexers are used to implement arbitrary Boolean functions of K inputs. Cell terminals are connected to the routing wires via programmable switches that interconnect the wires to achieve the desired routing patterns.

The FPGA implementation of digital circuits is given below:

1. Rapid design, implementation, and prototyping.
2. Reuse and erase ability
3. Easy implementation of ASIC
4. Reconfiguration of circuits
5. Reprogram ability of circuits

There are two major classes of commercial FPGA architecture—row-based and array-based architectures. In case of row-based architecture, logic cells and routing wires are arranged in row fashion, like a standard cell-layout style. The routing channels consisting of horizontal wires are segmented by programmable switches. Cells arranged in rows are connected by column wires to connect terminals on different rows.

In case of array-based architecture, two-dimensional grid arrangements are used and cells, routing terminals and switches are uniformly distributed. Horizontal and vertical wires are connected at programmable switchboxes where electrical connections can be made.

The main objective of FPGA is to achieve 100% routing completion. The routing is based on the FPGA architecture. The routing of the above architecture is described below.

1. Array-based FPGA

Figure 9.3 shows the routing architecture for array-based FPGA consisting of logic cells known as connection blocks (C-block) and switch block (S-block). There are vertical and horizontal channels which pass through C-blocks and at the crossing point of vertical and horizontal channels, the S-blocks are situated. The flexibility of a C-block, F_C is defined as the number of tracks a logic pin can connect to, and the flexibility of an S-block, F_S is defined as the number of outgoing tracks that the incoming tracks can connect to. Since S-block contribute resistances and capacitances, it is required to allow the routing paths to pass through minimum number of S-blocks. In Fig. 9.3, long wire segments (channels) that pass through more than a single C-block allow connection with few switches, along the routing pass by lowering parasites. The C-block and S-block consists of programmable switches which connects the vertical and horizontal channels. All channels have the same number of prefabricated tracks (W).

The route of the net is called if all its terminals are connected to exactly one track. The routing for array-based FPGA architecture is performed in the following manner:

The tracks in each horizontal channel are numbered from top to bottom and tracks in vertical channel are numbered from right to left. The number assigned to a track is referred as the track's id. The diagonal positions of switches in a S-block connect a track in horizontal channel with tracks with same id in vertical channel. This is called **diagonal *S*-block**.

The routing of FPGA is based on a graph consisting of a sequence of wives segments called **course graph**. The course graph $G(V, E)$, (where V = Number of vertices, E = Number of edges) decides specific wire segments implementing a particular connection.

1. In the first phase, an expanded graph is generated for each net by experimenting with the route switches and wire segments along the path described by the course graph.

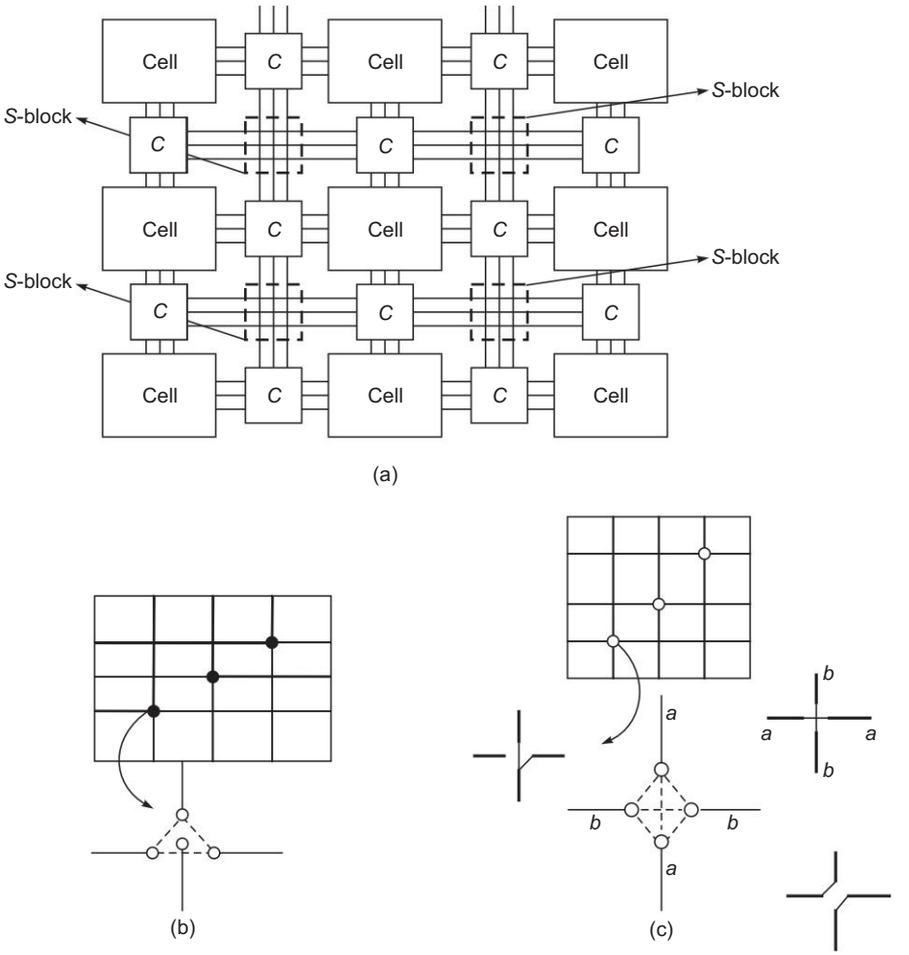


Fig. 9.3 Row-based EPGA: (a) Architecture (b) Connection in *S*-block (c) Switching in *S*-block

2. In the second phase, course-graph expansion places all the paths from all of the expanded graph into a single path list. The router selects paths from the list based on the cost function. Each selected path defines the detailed route of the corresponding connection.
   There are many algorithms for finding these routing:
   (a) Greedy-bin packing router
   (b) Multi-terminal net router

(a) Greedy-Bin Router Due to the property of array-based FPGA architecture, the routing resources are uniformly portioned into domains of equal capacity. The track domain is called a **bin**. The router uses these bins where bin geometry is fixed and same for all. Object size can be expanded from minimum requirements depending on the geometry of resources. The Greedy-Bin Packing (GBP) router is based on global to detail minimum track mapping where bin packing net is used. We define pin

density of a *C*-block as the number of unrouted two-pin net connection points within *C*-block which is updated and the next net is selected for routing.

(b) Algorithm Procedure Decompose all multi-pin nets into two pin nets.

```
  For every track domain
  {Scan the sorted queue formed by all two pin nets ordered from long
to short. Make at least five scan passes, each pass having the gradu-
ally relaxed screening rule
• Pass 1: Pick a net only among those whose pin is located in a C-block
  with current highest pin density. Route it by finding minimum
  manhattans distance path without passing through the C-block with
  highest pin density.
• Pass 2: A net can be packed without passing through the C-block
  with highest pin density.
• Pass 3: Pick a net only among those whose pin is located in the
  C-block with the current highest pin density. The net is routed,
  if a path exists for it which is at most one segment longer than
  minimum manhattans distance.
• Pass 4: A net can be packed if it is possible to find a path for it
  which is at most one segment longer than the minimum manhattans
  distance.
• Pass 5: If a net can fit into the currently packed domain (bin), it
  is accommodated.}
```

The heuristic packs as many nets as possible in a very greedy way into a track domain (bin) which is not yet full. In each bin, higher priority is given to longer nets which do not increase the routing density in the *C*-block and which are routed within minimum manhattans distance.

(c) Multi-terminal Net Router Performance and logic utilization is one of the main problems for FGPA. The multi-terminal net route is an array-based FGPA router that enhances logic utilization. In each multi-terminal net, the aim is to achieve 100% routing with the channel width and the routing delay. Each net max $\{ l(a_k^i, b_k^i)\}$ where $l(a_k^i, b_k^i)$ is Manhattans distance between $l(\theta_k^i, n_k^i)$. A net n_k is a set of one output pin and one or more output bias $n_k = \{\theta_k, i_k^1, i_k^2, i_k^2, ___ i_k^{pk}\}$ where p_k = Total number of input pins. The terminologies of a multi-terminal net router are—channel section, global graph, and detailed graph. A channel section is defined as the set of routing segments between two successive switch blocks in a horizontal row/vertical column. Two channel sections "*i*" and "*j*" are said to be adjacent if they share a common switch block. A global graph is a directed cyclic graph $G(V_G, E_G)$ rooted at the vertex V_0. There exits an edge V_i, V_j if "*i*" and "*j*" are adjacent channel sections as shown in Fig. 9.4(a). The bottom vertices are called **leaf vertices**.

A detailed graph is an expanded form of global graph in which we search for minimum cost wiring. The detailed graph is shown in Fig. 9.4(b). In the detailed graph, we expand global graph rooted from more than one vertex. The algorithm procedure of a multi-terminal net router is given below.

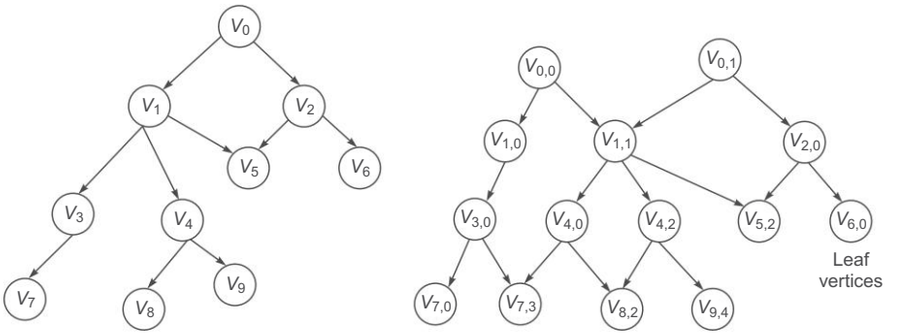


Fig. 9.4 Global graph for FPGA

Procedure Route: Input: A logic cell array, a set of net lists obtained after placement and pin assignment, a width of routing channel W flexibilities F_S and F_C, and topologies of S-block and C-block.

Output: A multi-terminal routing, if it is feasible with channel with "W".

Step 1: For each net n_k, $1 < k < N$, form the ordered sequence (i_k^{j1}, i_k^{j2},……., i_k^{jp}) of input pins such that $l(o_k, i_k^a) \leq (o_k, i_k^b)$, $1 \leq r \leq p_k - 1$, $a = j_{r-1}$, $b = j_r$.

Step 2: For each net n_k such that its ordered sequence of input pins is not empty, do steps 3-7.

Step 3: Take input pin $i(n_k)$ from the top of the ordered sequence of input pins.

Step 4: Construct a global graph $G(V_G, E_G)$ with the root net representing the channel section in which i_n belongs.

Step 5: Expand global graph into detailed graph $D(V_D, V_D)$.

Step 6: Select the minimum-cost leaf node (one without degree equal to zero) and track back in minimum-cost path to the routed portion $CR(n_k)$.

Step 7: Update cost of all segments and remove the routed input pins from net n_k.

The cost function depends on the number of segments in the adjacent channel section. The cost is defined as,

$$\text{Cost Function} = \sum_{j=1}^{n_d} \sum_{i} \frac{1}{S(i,j)} + \sum_{k} \frac{\alpha}{P(k)}$$

where, $S(i, j)$ = Segment connectivity for i^{th} segment, $1 \leq i \leq n_d$.
n_d = Number of adjacent channel section,
$P(k)$ = Pin connectivity where k = Number of unused wiring segments.

2. Row-Based FPGA

The routing architecture of row-based FPGA is shown in Fig. 9.5. The logic cells are laid out in rows separated by routing channels laid down horizontally. Each input and output of the cells connect to a dedicated vertical segment. Each crossing of vertical and horizontal segments has programmable switches and by programming a switch, a path is formed between two crossing segments. The routing of row-based FPGA is based on segmented channel routing in which wiring of the routing is configured by programming.

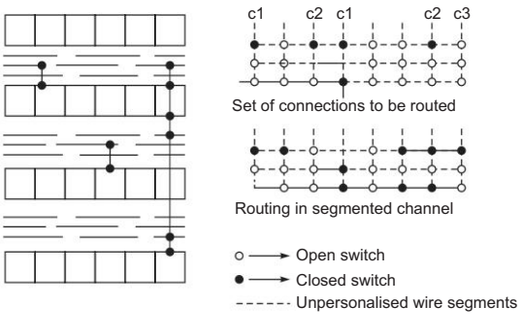


Fig. 9.5 Architecture of row-based FPGA

If there are no switches where segments in the same track meet then each connection must be in a single segment. This type of routing problem is called **one-segment routing problem**.

A set of connections are routed in a segmented channel and each connection occupies at most k segments. This problem is called a k-segmented routing. Let C, T be the number of connections and number of tracks left respectively. The connection is assigned from left end and each connection being processed, find the set of tracks in which the connection would occupy one segment (unoccupied). To find the selection of k-segment routing problem, frontier and assignment trees are introduced. The assignment tree is based on the constructed data structure to represent the effect of optionally assigning each connection in each track. A frontier tree is based on a function that shows how a valid routing of a set of connections c_1, c_2, ….., c_i, can be extended to include the next connection c_{i+1}, where c_i is the end point left (c_i) and right (c_i). For a routing of connection c_1, c_2, …. c_i, the frontier vector X is defined as

$$X = \{x[1], x[2], \ldots\ldots, x[t]\}$$

where, $x[t]$ is the leftmost column in the track t in which the segment present in column is not occupied by any of the connections c_1, c_2, …., c_i. Once the assignment graph is made, a valid routing can be obtained by tracking back a left root path in the assignment tree. The algorithm procedure for k-segment routing is given below.

```
For each node Xi in level i;
For each track t, 1 ≤ t ≤ T
    begin-1
        if Xi[t] ≤ left(ci+1)
         {if ci+1 could occupy ≤ k-segment in track t;
    begin-2
    (ci+1 can be assigned to track t)
    Let Xi+1 be the new frontier
    when ci+1 is assigned to t;
```

```
if Xi+1 is not yet in the level(i+1);
{
    add node Xi+1 ;
    add an edge from Xi to Xi+1}
  end-2
Xi > left (ci+1) so ci+1 cannot be assigned to track t;
end-1
```

This approach can be slightly modified to minimize the summation of the lengths used to route the connections. An additional attribute weight w is used and it corresponds to the total length of all segments assigned to connections $c_1, c_2, \ldots, c_{i+1}$.

3. Configurable Logic Block (CLB)

Configurable logic block is an important component in FPGA architecture**.** The digital circuits in terms of VHDL code can be implemented by configuring logic blocks with routing structures consisting of vertical wires and horizontal wires via many programmable connections. There are two types of CLB—SRAM based cells and anti-fuse based cells. The logic-cell architectures influence routing performances.

(a) SRAM-Based Cells SRAM based cell is the largest cell with fewer inputs and outputs. These cells can implement larger programmable functions without incurring routing delays which can play an important role in circuit performance because of larger resistance and capacitance of the programmable element. Since it has fewer outputs as ratio of the number of gates in logic cell is less, it is less efficient for implementing small programmable functions. In this architecture, static RAMs (SRAM) are used to control the state of pass transistor which can establish connections between horizontal and vertical wires. Figure 9.6 shows a SRAM cell used to control states of pass transistors and SRAM cell architecture. An SRAM memory cell consists of two inverters (each inverter has one *p*MOS and one *n*MOS) latched to each other, and an *n*MOS transistor is used to which select the program bit. The SRAM based CLB cells are reprogrammable and physically altered when programmed. SRAM cells are volatile which means that the states of cells are lost when power is not applied. As with anti-fuse elements, the programming circuits for SRAM elements must include the addressing and data registers. The SRAM cells are larger than anti-fuse FPGA CLB, then SRAM CLB elements can be placed as densely as anti-fuse FPGA elements. SRAM FPGAs cannot have routing architecture for which there is a programmable element at every interconnection.

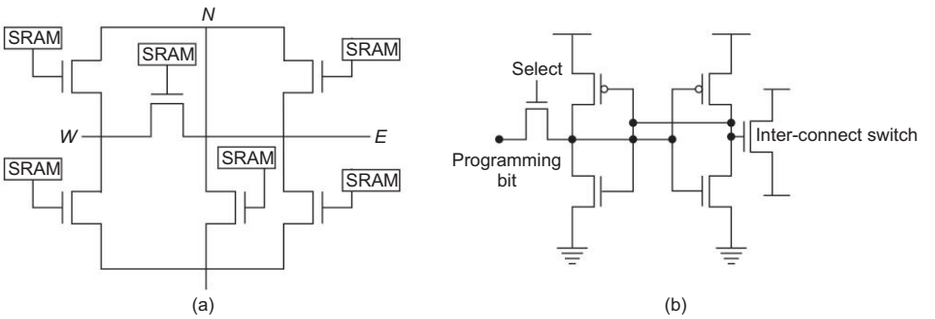


Fig 9.6 (a) SRAM cell used to control (b) SRAM memory cell

Figure 9.7 shows XC-4000 configurable logic block which has 12 input signals, 1 clock input, and 4 outputs. This is an example of SRAM-cell-based CLB. It has a complex cell in which two 4-input LUTs are connected to another 3-input LUT. Each CLB can implement any function of 4 or 5 variables and some up to 9 variables. Alternatively, a CLB can be configured to implement two functions of four variables or one of two variables and another of five. Each CLB has two flip-flops. The CLB can also be configured for special arithmetic circuits such as a two-bit adder with carry-in and carry-out or as a read /write RAM of 16-bit far string data.

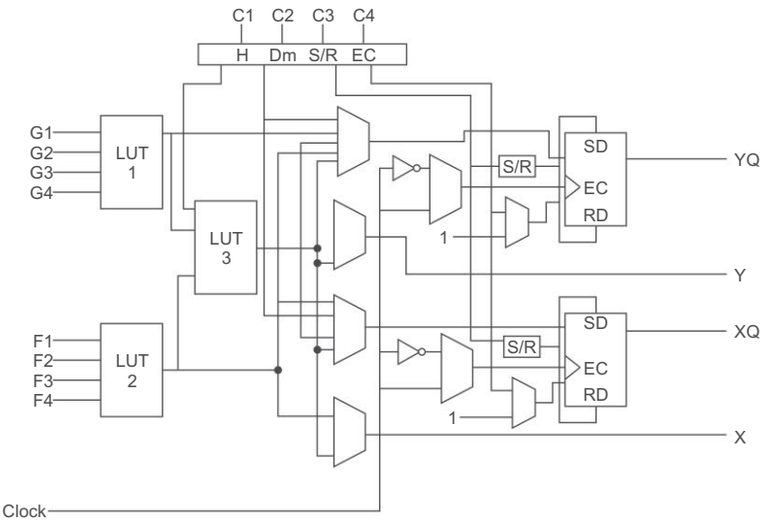


Fig. 9.7 XC-4000 configurable logic block

Figure 9.8 shows the Xilinx *X*C-4000 interconnect in which the wires are extended for connection to the wires of other CLBs. Each side of a CLB has two wires—one as data input, one wire as an output

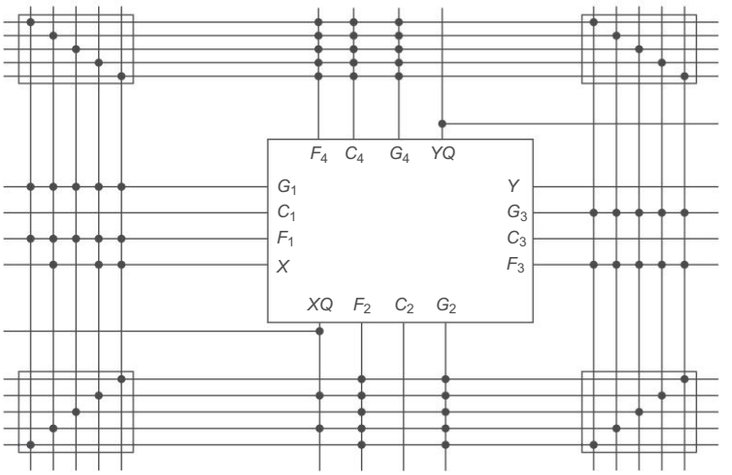


Fig. 9.8 XC-4000 Interconnect

and the other as a control input, and only one side has one more wire for CLK signal input. These wires are connected to other wires via a switch block and connection block as per routing programmable architecture.

(b) Anti-fuse FPGA Anti-fuse FPGA user logic cells have more inputs and more outputs because of the availability of the wires to transport signals and fuse which allow nearly any wire to connect to any other wire. Routing doesn't impose any limitation. ACT3 and ORCR cells are anti-fuse-type CLB. Because of many input data and outputs, it can implement a function of up to 11 inputs. Figure 9.9 shows antifuse based CLB which is a logic cell of Actel's ACT3 consisting of eight inputs and one output. In this type of logic cell, there are combination modules and sequential modules which include a *D* flip-flop. The combination module consists of a 2^n to 1 multiplexer with OR gate and AND gate.

There is another logic cell which is SRAM and is developed by Lucent Technologies.

It consists of fourteen inputs and five outputs. There are four input look-up Tables (LUT).

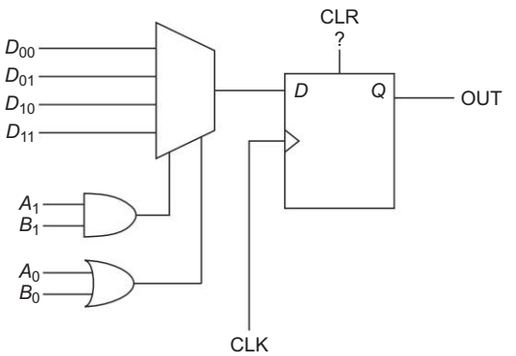


Fig. 9.9 ACTS logic modules below

Figure 9.10 shows the logic cell named ORCA developed by Lucent Technologies.

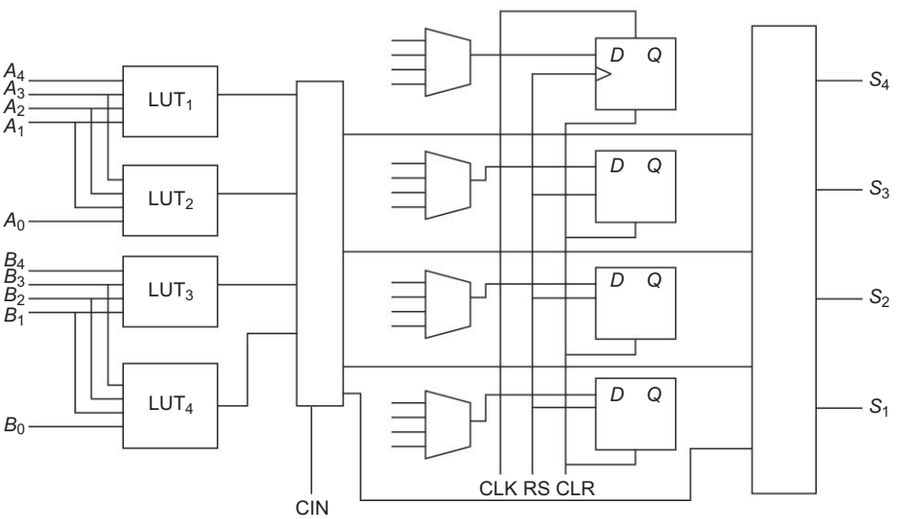


Fig. 9.10 ORCA logic cell

9.1.2 Complex Programmable Logic Device

The complex Programmable Logic Device (CPLD) is an extension of PLD to a higher level of implementation to improve the system performance. Instead of making PLD larger with more inputs, product terms and macro cells, a CPLD contains multiple logic blocks (which are same as a small PLD). The logic blocks communicate with one another using signals routed via a programmable interconnection. This architecture arrangement makes more efficient use of chip area leading to better performance.

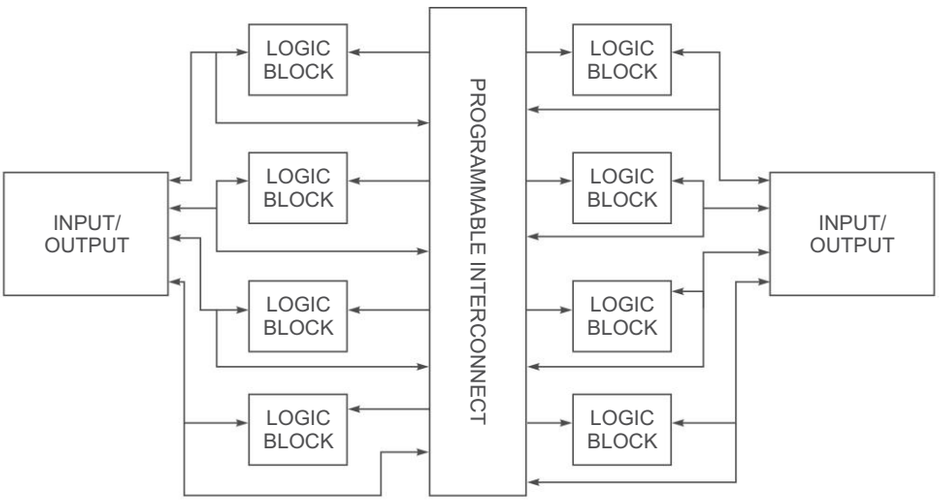


Fig. 9.11 Generic CPLD architecture

Figure 9.11 shows a generic CPLD architecture consisting of logic blocks which are connected to input/output terminals and programmable interconnect.

1. Programmable Interconnect

The Programmable Interconnect (PI) route signals from I/O to logic block inputs or from logic block outputs (macro-cell outputs) to inputs of the sane or other logic block. There are two types of programmable interconnect—array-based interconnect or multiplexer based-interconnect.

Array-based interconnect allows any signal in the PI to route any logic block. Each term in the PI is represented by a vertical wire and is assigned as an input to a given logic block, so there is one term for each logic block.

Figure 9.12(a) shows array-based PI and Fig. 9.12(b) shows a multiplexer based PI in which there is one multiplexer for each input to a logic block. The signals from PI are connected to the inputs of a number of multiplexers for each logic block. The selection of lines in a multiplexer are programmed to allow one input for each multiplexer to propagate into a logic block. The routing ability is increased by a wider multiplexer allowing each signal in the PI to connect to the input of several multiplexers for each logic block.

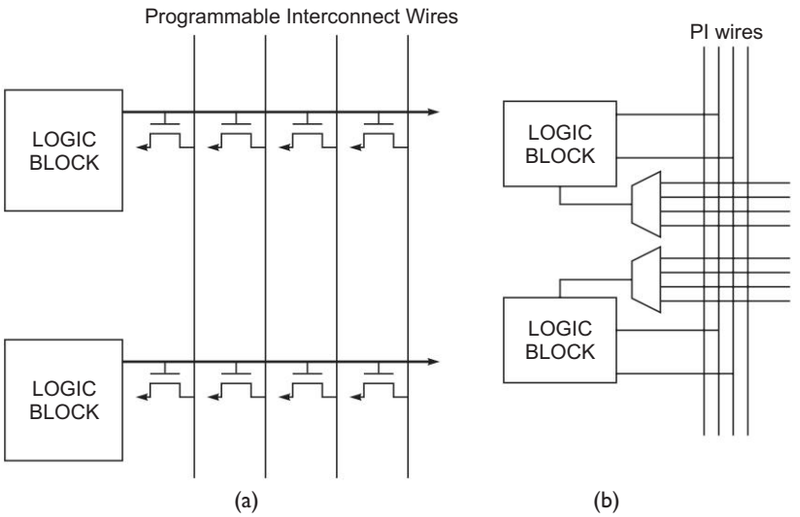


Fig. 9.12 Programmable interconnection: (a) Array based (b) Multiplexer based

2. *Logic Block*

Figure 9.13 shows a basic block which is similar to a Programmable Logic Device (PLD) consisting of product-term array, product-term distributor and macro cells. In the figure, the signals from a PI are transferred to the product-term array and then product terms are connected to I/O cells via macro cells.

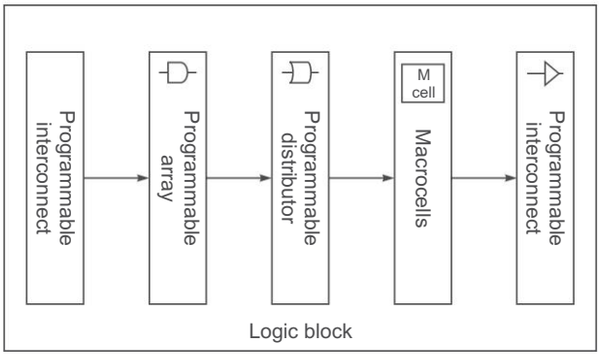


Fig. 9.13 Generic logic block

A product-term array identifies the average number of product terms per logic block, macro cells, and the maximum number of product terms per logic block. There is a slight difference in these arrays for different CPLDs. Product-term distribution follows different schemes for different CPLDs. There are two schemes mainly used for product-term distribution—product-term steering and product-term sharing. In **product-term steering**, a product term can be used by the particular macro cell. In **product term-sharing**, the same product term is used by multiple macro cells. Figure 9.14 shows product terms and product-term distribution based on sharing type.

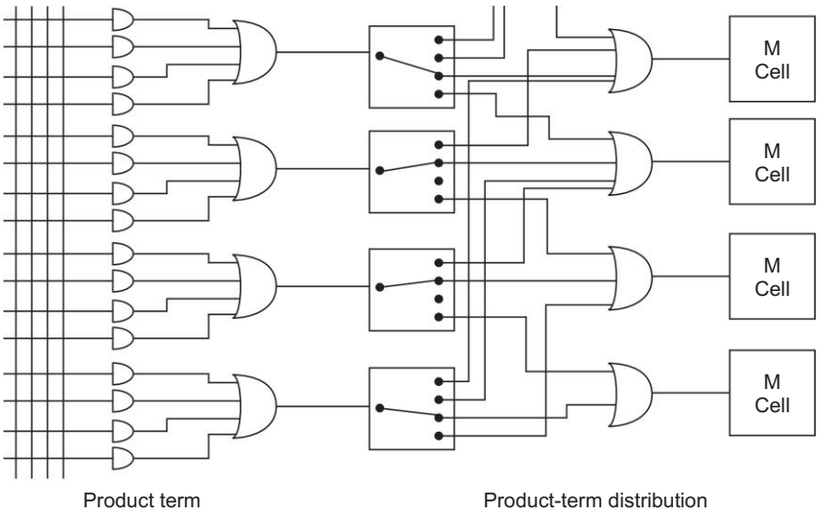


Fig. 9.14 Product-term distribution bases on sharing

Macro cells provide flip-flops and polarity control. **Polarity control** enables the implementation of either true or complement of an expression. So CPLD macro cells are configurable. There are several inputs of macro cell and polarity-based multiplexer in macro cells used for control of I/O of macro cells. Figure 9.15 shows an I/O cell connected to a macro cell.

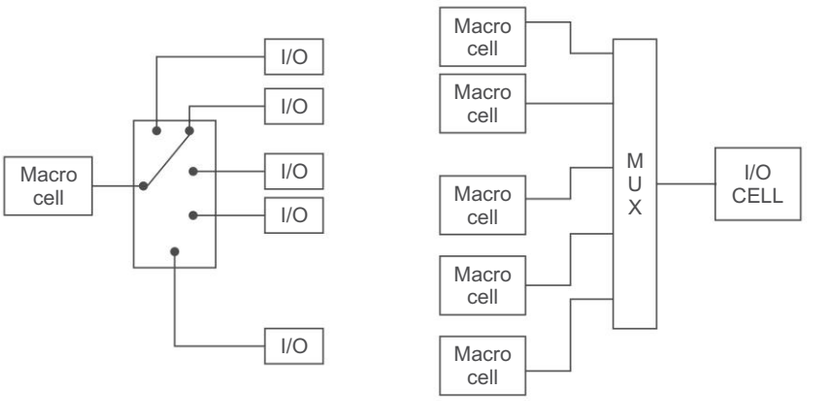


Fig. 9.15 Macro cell with I/O cell

9.2 □ VHDL LANGUAGES OR CODES

There are two parts of VHDL description of circuits—entity declaration and architecture body. Apart from these, line numbers are not part of VHDL codes but are used for indication of specific lines of code.

9.2.1 Entity Declaration

An entity declaration describes the inputs and outputs of a design entity and parametric values. The input and output (I/O) could be the I/O of an entity which is also the device level description—I/O of a chip. It also can represent a schematic symbol which describes components connection to the rest of the design. As for an example, entity declaration of a adder which consists of 4-bit inputs (*a* and *b*), a carry input, a 4-bit output (sum) and carry output (co) is given below:

```
Entity add4 is port (
a, b: in std_logic_vector(3 down to 0)
c1: in std_logic
sum: out std_logic_vector(3 down to 0)
c0: out std_logic);
end add4;
```

The symbolic equivalent is shown in Fig. 9.16.

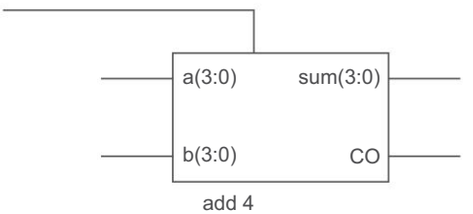


Fig. 9.16 4-bit adder block

Ports

Each I/O signal in an entity declaration is referred as a port which is analogous to a pin in a schematic symbol. A port is a data object, which can be assigned values and used in an expression, like other objects. The set of ports defined for an entity is referred as a port declaration. Each port has a name called a mode which describes the direction in which data is transferred through a port. The following modes are considered:

- **in:** Data flows only into entity. The driver for a port of mode, **in** external to the entity. Mode is used for clock inputs, control onputs, and unidirection data inputs.
- **out:** Data flows only from source to output of entity. Mode **out** is inside the entity and is used for count output.
- **buffer:** Internal feedback which is needed to declare a port as mode **buffer** or declare a separate signal for use within architecture body (an internal signal). A port that is declared as mode **buffer** is similar to a port that is declared as mode out.
- **in/out:** Bidirectional declaration in which mode in out allows data flow into or out of the entity.

The entity declaration for add 4 can be processed in VHDL software/compiler if it is written as follows:

```
library ieee;
use ieee: std_logic_1164 all;
entity add 4 is port (
    a,b: in std_logic_vector (3 down to 0);
    c1: in std_logic;
    sum: out std_logic_vector (3 down to 0);
    c0: out std_logic);
end add 4;
```

9.2.2 Architecture Bodies

An architecture body is an important part of VHDL code and is associated with entity declaration. The architecture body describes the contents of an entity. There are three types of style used for description of architecture body—behavioral description, dataflow description and structural description. These types describe the design at mixed mode or the design matches the styles as we see fit for the same. They allow us to describe a design at different levels of abstraction from using algorithms to gate-level primitives.

1. *Behavioral Description* Behavioral descriptions are referred as high-level descriptions because of their resemblance to high-level programming. Here, rather than specifying the structure or netlist of a circuit, the designs are specified with a set of statements that model function or behavior of entity or port of entity.

In this style, first lines are required to make `std_logic` vector types visible to this entity. Then, the lines contain entity declarations. Then, we have an architecture body which uses an algorithm to describe the design function. A process statement constructs embodying algorithms in VHDL codes of design. A sensitivity list identifies which signals cause the process to execute. Sequential statements are used to model the function of devices used in the circuits. The '`if` statement' is used to indicate that 'equals' should be asserted when '`a`' is equal to '`b`'. The main part of behavior style of architecture bodies are process statements and sequential statements. Example of behavioral style is given below:

```
 1. library ieee;
 2. use ieee std_logic_1164.all;
 3. entity eqcomp4 is port(
 4. a,b: in std_logic_vector(3 down to 0);
 5. equals: out std_logic);
 6. end eqcomp4;
 7. architecture behavioral of eqcomp4 is
 8. begin
 9. comp process(a,b)
10. Begin
11. if a = b, then
12. equals <= 1
13. else
```

```
14. equals <= 0
15. end if
16. end process  comp
17. end behavioral
```

2. Dataflow Descriptions

Dataflow style specified how data will be transformed from signal to signal and input to output without the use of sequential statements. The difference between behaviour style and dataflow style is that one uses process and the other uses how data will flow. In the dataflow descriptions, conditional statements (when-else) or selected signal assignments (with-select-when) statements rather than algorithmic statements are used for representing simple equations. The example of `eqcomp4` in a dataflow description is given below:

```
Library ieee
Use ieee.std_logic_1164.all;
Entity eqcomp4 is port (
a,b: in stdd_logic_vector(3 downto 0);
equals: out std_logic
end eqcomp4
architecture dataflow of eqcomp4 is
begin
equals <= 1 when a=b else 0;
end dataflow
```

We can describe the same comparator using Boolean equation in dataflow:

```
Architecture bool of eqcomp4 is
Begin
Equals <= not a(0) xor b(0)
          and not a(1) xor b(1)
          and not a(2) xor b(2)
          and not a(3) xor b(3)
end bool;
```

The bool equation description is also a dataflow description because it describes the way in which data flows from signal to signal. Dataflow architectures are concurrent signal-assignment statements rather than processes and their sequential statements.

3. Structural Descriptions

The structural design requires that `and4` and `xnor2` components be defined in a package and that this package be compiled into a library. It consists of VHDL netlists. These netlists are very much like schematic netlists: components are instantiated and connected together with signals. To instantiate a

component is to place it in a hierarchical design. An instantiation is either an act of placing a component or an instance of a component in structural design in hierarchy in which separate entity declarations and multiple architectural bodies are used.

The example of `eqcomp4` in structural design is given below:

```
Library ieee
Use ieee.std_logic_1164.all;
Entity eqcomp4 is port(
a,b: in std_logic_vector(3 downto 0)
aeqb: out std_logic
end eqcomp4;
use work.gatepkg.all;
architecture structure of eqcomp4 is
signal x: std_logic_vectot(0 to 3)
begin
n0: xnor port map (a(0),b(0),x(0));
n1: xnor port map (a(1),b(1),x(1));
n2: xnor port map (a(2),b(2),x(2));
n3: xnor port map (a(3),b(3),x(3));
n4: xnor port map (a(4),b(4),x(4));
end struct;
```

In the above example, separate entity declaration and architecture body pairs are created for the `and4`, `xnor2` and `eqcomp4` design entities.

4. Comparison Analysis of Behavioral, Dataflow, and Structural Architecture

The different architectures produce different design descriptions but are functionally equivalent. Design and equations result in different circuit implementations. Before writing codes for describing a design, it is required to choose a device architecture resource properly, otherwise synthesis software may perform poorly. These synthesis tools produce a sum of products and logic equations and pass these equations to the placer and router without having optimized the logic for target architecture. If the synthesis tools have no placer and router then the design may require more space for synthesis.

In case of behavioral architecture, a high-level programming language is used and a set of statements specify the design of the circuit without using a netlist whereas dataflow architecture specifies how data will be transferred from signal to signal and from input to output without use of sequential statements. In the structural architecture, the components are defined in a package and the package is compiled in a library. The structural design methodology is often used to decompose a design into manageable units and is also used to attain higher degree of control over synthesis than other architectures.

For designing of complicated circuits such as microprocessor and microcontroller, all three architectures are used simultaneously for synthesis and simulation. This type of architecture is called **mixed design methodology**.

9.2.3 VHDL Language Statements

VHDL contains a few language constructs that are specific to simulation and verification than to synthesis. Synthesis software may ignore such structures or rules. The functional results of simulating codes must be matched with functionality of hardware specified by synthesis. So both simulation and synthesis tools are used with the same code. The different statements execution such as "process", concurrent statements, simple interpretation statements, multiple driver signal statements, and synthesis statements should be known.

1. Process Statements Execution

A process statement is either being executed or suspended. It is executed when one of the signals in its sensitivity has an event or a change in its value. In this case, sequential statements are executed. These sequential statements are `if` statement, loop statements and `case` statements. A process continues to execute until the last statement is reached and at this point, the process suspends and it is executed again where there is an event on signal. A "wait" statement is used to suspend the process. Synthesis software supports a "wait "statement at the beginning or end of the process.

This process can be written as follows:

```
Interesting: process (a, b, c)
begin
     x<=0;
     if (a=b or c=1),
               then x<=1;
     end if;
end process interesting;
```

If *b* and *c* are '1' and a change from 0 to 1 at T_c time then process 1 will execute when the simulation time is T_c. At this time, the current value of *x* is 0 and a transaction is added to the signal driver for *x*: ('1', $T_c + \delta$) where δ is a small delay.

2. Simple Interpretation Execution

A simple way to interpret signal assignment in a process is to assume that all expressions are based on the current value of the signals on the right-hand side of the <= symbol, that a signal will be updated with the value in the last assignment, all the signals will be updated with the value in the last assignment, and that all the signals will be updated at the end of process when it suspends. An example representation of AND gate is given below:

```
entity my_and is port (
    a_bus: in bit_vector(T down to 0);
    x: buffer bit)
    end my_and
 architecture of wont work of my_end is
     begin
     ending: process(a_bus)
```

```
        begin
              x<='1'
          for I in 7 down to 0 loop
           x<=a_bus(i) and x;
        end loop;
        end process;
end wont work;
```

The following two processes show the equivalence of a sensitivity list and an explicit wait statement:

```
    Proc1: process (a,b,c)
        begin
          x<=a and b and c;
                end process
          Proc 2: process
      begin
          x<=a and b and c;
          wait on a,b,c;
      end process;
```

Both the processes will execute when there is a change in value of signals a, b or c and the sequential signal assignments x <= a and b and c; are the values of logical AND of a, b, c at current simulation time T_c. The signal x is assigned after a time $T_c + \delta$, where δ is the time delay after which the process is executed. This clause of the process can be written as

```
Proc 3: process (a,b,c)
        begin
        x<=a and b and c after 5ns;
end process
```

This process statement represents a 3-input AND gate.

3. Signal Driver Representation of Process Statement

An event is said to be scheduled for a signal driver for x which is represented by a projected output waveform. Each time signal assignment is made, that signal's projected waveform is updated. A projected output waveform is a set of transactions that specify new values for a signal and the time at which the signal will assume new values. In most signal drivers, there are two transactions–current transaction specifies the current value and time, and the next transaction if it exists, specifying the new value to be assumed.

X ←	(current value T_c)	(New value $T_{c+\delta}$)

4. Concurrent Statement Execution

A concurrent signal assignment statement has an implicit sensitivity list that includes all signals in the right-hand side of the <= symbol. Concurrent statements execute any time a signal in the implicit sensitivity list transitions. When there are multiple concurrent statements, they do not execute sequentially.

5. Simulation Cycle Execution

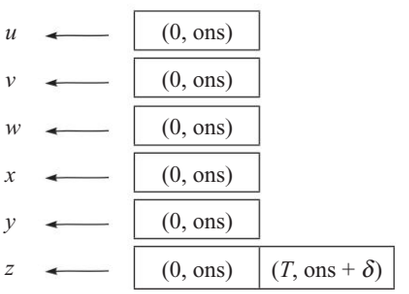
When a model is simulated, it goes through an initial phase and then repeated simulation cycles. After an initialization process, simulation cycles are run and each consisis of the following steps:

- Signals are updated. Updating will cause signals to transition, and events to occur on these signals.
- Each process that is sensitive to a signal that has an event on it in the current simulation cycle is executed.
- The simulation time of the next cycle T_n is determined. If the simulation time for the next cycle is a delta delay or multiple delta delays from current simulation time then the current simulation time T_c remains same and a delta cycle consisting of the same steps as above is executed; otherwise the current simulation time is set to the next simulation cycle time ($T_c = T_n$).

The example of a simulation cycle is given below:

```
        entity delta in port(
        a,b,c,d: in bit;
        u,v,w,x,y,z: buffer bit);
end delta;
Architecture delta of delta is:
begin
        z<= not y
        y<= w or x
        x<= u or v
        w<=u and v
        v<= c or d
        u<=a and b
        end delta
```

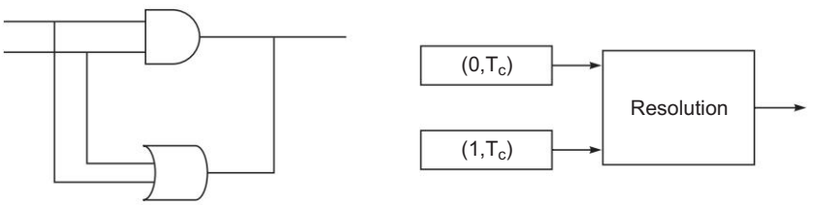
In the initialization phase, all signals are said to be '0'. Next each concurrent statement is executed. The signals for drivers are updated as per the following:



These delta cycles are required to update signals before a current simulation time can advance.

6. Multiple Drivers and Resolution Functions Executions

VHDL does not allow a signal to have more than one driver unless it has an associated resolution function. A resolution function is used to comute a value for a signal based on multiple drivers. We consider the following example for multiple drivers having two drivers/gates for a signal output as shown in Fig. 9.17.



Truth table

a	b	a And b	a OR b	y
0	0	0	0	0
0	1	0	1	x
1	0	0	1	x
1	1	1	1	1

Fig. 9.17 Example of multiple driver circuit

The logic of the above example can be described by the following VHDL code:

```
Architecture multiply_driven_signal of my design is
               begin
                     Y<=a and b
                     Y<=a or b
               end multiply_driven_signal
```

In this description, two concurrent signal assignments cause two drivers to be created for *y*. Although these two signal assignments to the same output signal in the same process do not create separate drivers, sequential signal assignments to the same processes would have the same effect. A resolution function can be used to compute a value for *y* based on the values of the driver. From the truth table, it is clear that if two drivers are the same, the output is known; otherwise it is not.

```
    function compute_value(a,b: x01) return x01 is
          begin
                If    a/=b then return('x');
                      else return(a);
          end;
```

9.2.4 Identifiers, Data Objects, Data Types, and Attributes

1. Identifier

Basic identifiers consist of alphabetic, numeric and/or underscore characters. The following values are followed:

- First character must be a letter.
- Last character cannot be an underscore.
- Only letters, digits, and underscores can be used for identifiers.
- Two underscores in succession are not allowed.
- Reserved words (such as *select*) cannot be an identifier.

The following equivalents transmission clock can be written as `tx_clk`. There are other identifiers which are given below:

```
Three_state_enable
      HIT_1124
      Se 17D
```

The following are not legal identifiers:

```
-tx_clk, 8B10B, large#number, link_ _bar, select rx_clk_
```

The above identifiers follow the rules which are mentioned above.

2. Data Objects

Data objects hold values of specified types. There are four classes of data objects—constants, signals, variables, and files.

(a) Constants A constant holds a value that cannot be changed within the design description. This value is usually assigned on declaration. The following constant may represent the width of a register:

```
constant width: integer:=8;
```

The constants must be declared in a package, entity, architecture, or process declarative region. A constant defined in a package can be referenced by any entity or architecture for which the package is used.

(b) Signals Signals can represent wires and can, therefore, interconnect components. Ports are specifically declared as signals. The wire signals can be input or output of logic gates.

```
signal count: bit_vector (3 down to 0);
```

`Count` may represent the current state of a counter and it represents memory elements. The `:=` symbol represents immediate assignments and is used to indicate the initial value of the signal. For simulation, this initialization ensures that the signal is initialized as to 0101. It can be described as

```
signal count: bit_vector(3 down to 0):= "0101";
```

(c) Variables Variables are used only in processes and subprogrammes. They are declared in the declaration region of VHDL codes and are used for high-level modeling. Variables are declared in the following manner of VHDL codes:

```
variable result: std_logic:='0';
```

Variable assignments are immediately by not scheduled and variables do not project any output waveforms. Variables can be declared in process of the VHDL code by the following way:

```
architectute will_work of my_and is
    begin
   ending:process(a_bus)
   variable temp: bit;
    begin
         temp:='1';
           for i in 7 downto 0 loop
                 temp:=a_bus(i) and temp;
            end loop;
             X<=temp;
         end process;
     end will_work;
```

The variable '`temp`' is declared in the declarative region and initialized each time the process is activated, so that it will not retain its previous value.

(d) Files Files contain values of a specified type and are used to read stimulus and write out data using test benches.

(e) Aliases Aliases are not data objects. An alias is an alternate identifier for an existing object but it is not a new object. Making an assignment to the alias is equivalent to making an assignment to the original object. An alias is often used as a convenient way to identify a range of array types.

```
signal address: std_logic_vector(31 downto 0);
alias top_ad: std_logic_vector(3 downto 0) is address (31 downto 28);
alias bank: std_logic_vector(3 downto 0) is address(27 down to 24);
alias row_ad: std_logic_vector(23 downto 0) is address(23 downto 0);
```

(f) Data Types There are two data types—scalar types and composite types which are used for synthesis.

(i) Scalar types Scalar type is an order which allows relational operators to be used with them. There are four categories of scalar types—enumeration, integer, floating, and physical types. A scalar-type data object can hold one value at a time.

- **Enumeration Types** An enumeration type is a list of values that an object of that type may hold. These are defined as follows:

```
type bit is ('0','1');
     type is   states  (idle, preamble, data, jam, error);
     type std_logic is ('u','x',' θ','1','Z');
```

- **Integer Type** Integers and the relational and arithmetic operators for integers are predefined. A signal or variable that is an integer type and that is to be synthesized into logic should be constrained with a range.

```
variable  a; integer range -255 to 255;
```

An integer can be represented internal to a synthesis tool by a signal vector. The subtype natural can be represented in a synthesis tool by an unsigned vector.

- **Floating Type** Floating-point type values are used to approximate real numbers. Like integers, floating-point types can be constrained. The only predefined floating type is real which includes at minimum the range –1.0E38 to +1.0E38, inclusive.
- **Physical Type** Physical-type values are used as measurement units. The only predefined physical type is *time*. Its range includes, at minimum, the range of integers. Its primary unit is fs (femtoseconds) and is defined as follows:

```
type time is range -2147483647 to 2147483647
units
fs;
bs = 1000 fs;
ns = 1000 ps;
us = 1000 ns;
ms = 1000 us;
sec = 1000 ms;
min = 60 s;
hr = 60 min;
```

We can also create other physical types such as meters, grams, ounces, and so on.

(ii) Composite Types Data objects of scalar types can hold multiple values at a time. Composite types consist of array type and record type.

- **Array Type** An object of array type consists of multiple elements of the same type. The most commonly used array types are those predefined by the IEEE 1076 and 1164 standards.

```
type bit_vector is array (natural range <>) of bit;
type std_ulogic _vector is array (natural range <>) of std_ulogic;
```

These array types are unconstrained arrays. The arrays may be constrained. An example is

```
type word is array (15 downto 0) of bit;
signal b: word;
```

- **Record Type** An object of a record type has multiple elements of different types. The individual fields of a record can be referenced by the element name. For example,

```
type iocell is record
buffer_inp: bit_vector (7 downto 0);
enable: bit;
buffer_out: bit_vector (7 downto 0);
end record;
```

(g) Attributes An attribute provides information about items such as entities, architectures, types, and signals. The attribute values are left, right, high, low, and length. The attribute 'left' yields the left-most value of a type and 'right' gives the right most value whereas 'high' provides the greatest value of a type, and 'low' yields the lowest value of a type. The examples are given below:

```
count left = 0;
count right = 127;
count high = 127;
count low = 0;
```

The attribute yields the number of elements in a constrained array.

(h) Common Errors There are several common errors that may give syntax and semantics. These errors are located or identified in the line numbers specified in VHDL code. Common errors are

(i) Omission of semicolon (:) or wrong placement of semicolon

Example: end many errors

Errors: The semicolon is omitted

(ii) Missing keywords

(iii) Semantic error

The process sensitivity list is missing. Although, this is syntactically alright, yet it gives the process the wrong meaning.

Consider the VHDL code with these common errors as follows:

```
entity many errors is port  -  line 1
       a: bit_vector(3 to 0) -  line 2
       b: out std_logic_vector(0 to 3) - line 3
       c: in bit_vector (5 downto 0) - line 4
             end many errors - line 5
architecture not_so_good of many_errors  - line 6
      begin - line 7
          my_label: process - line 8
          begin - line 9
          if c = X "F" then - line 10
                 b <=a - line 11
          else - line 12
                 b<= '0101'; - line 13
          end if; - line 14
                          end process.  - line 15
      end not_so_good - line 16
```

The corrected form of VHDL code is given below.

```
                  library ieee;
                  use ieee.std_logic_1164.all;
entity many_errors is port (
        a: std_logic_vector (3 downto 0);
        b: out std_logic_vector (0 to 3);
        c: in bit_vector (5 down to 0);
      end many_errors;
architecture not_so_good of many_errors is
begin
my_label: process (c,a)
begin
if c = "01111" then
```

```
    b <= a;
else
    b <= "0101";
end if;
end process;
end not_so_good;
```

line 1: It requires parenthesis
line 2: ok
line 3: ok
line 4: wrong placement of semicolon
line 5: it requires semicolon
line 6: missing keyword which is after the name of entity
line 7: ok
line 8: semantic error in which process should be specifiec properly
line 9: ok
line 10: semantic error
line 11: it requires semicolon
line 12: ok
line 13: it requires " " instead of ""
line 14: it requires semicolon
line 15: ok
line 16: it requires semicolon

9.3 ▫ REPRESENTATION OF COMBINATIONAL LOGIC USING VHDL CODES

Combinational logics can be described in several ways—First In First Out (FIFO) design, concurrent statements, and sequential statements. In case of FIFO design, the combinational logic is described by the signals demux and *en*, as well as the three state buffers. The three state buffers are described with behavioral code using `if-then-else`. FIFO design is mostly used in the circuit consisting of both combinational logic and synchronous logic. So FIFO design will be discussed later on.

1. Concurrent Statement Design for Combinational Logic

Concurrent statements stay outside a process and are executed concurrently. There are mainly three types of concurrent statements which are used in dataflow descriptions—concurrent signal-assignment statements with Boolean equations, selective signal-assignment statements (with `select_when`), and conditional signal assignment (`when_else`) statements.

2. Concurrent Signal Assignment with Boolean Equations

Boolean equations can be used in both concurrent and sequential signal assignment statements. An example of this typed, concurrent statements for a 4-to-1 mutiplexer that multiplexes 4-bit buses is given below:

```
library ieee;
use ieee.std_logic_1164.all;
entity mux is port (
a,b,c,d: in std_logic_vector (3 downto 0);
  s: in std_logic_vector (1 downto 0);
  X: out std_logic_vector (3 downto 0);
end mux;
```

The architecture `archmux` of `mux` is

```
begin
X(3) <=(a(3) and not (s(1)) and not(s(0))
   or (b(3) and not (s(1)) and s(0))
   or (c(3) and s(1) and not (s(0)))
   or (d(3) and s(1) and s(0));
X(2) <= (a(2) and not (s(1)) and not (s(0))
  or (b(2) and not (s(1)) and s(0))
  or (c(2) and s(1) and not (s(0)))
  or (d(2) and s(1) and s(0));
X(1) <= (a(1) and not (s(1)) and not (s(0))
  or (b(1) and not (s(1)) and s(0))
  or (c(1) and s(1) and not (s(0)))
  or (d(1) and s(1) and s(0));
X(0) <= (a(0) and not (s(1)) and not (s(0))
  or (b(0) and not (s(1)) and s(0))
  or (c(0) and s(1) and not (s(0)))
  or (d(0) and s(1) and s(0));
 end archmux;
```

The block diagram of a 4-to-1 multiplexer is shown in Fig. 9.18.

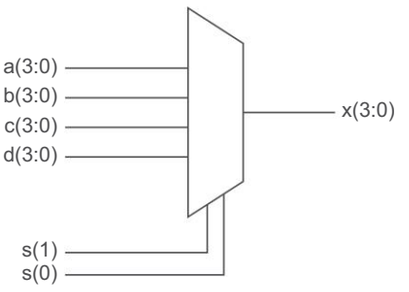


Fig. 9.18 4-to-1 multiplexer

Selective signal statement (`with _select_ when`):

The with_select_when statement provides selective signal assignment which means that a signal is assigned a value based on the value of the selection signal. It is written as follows:

```
With selection_signal select
signal_name <= value_a when value_1 of selection_signal,
            <= value_b when value_2 of selection_signal,
            <= value_c  when value_3 of selection_signal,
            <= value_d when last_signal of selection_signal ;
```

Signal_name is assigned a value based on the current value of selection_signal, where all values of selection_signal must be listed in the when clauses. Example of mux is given below:

```
 library ieee;
 use ieee.std_logic_1164.all;
 entity mux is
 port ( a,b,c,d: in std_logic_vector (3 downto 0);
        s: in std_logic_vector (3 downto 0);
        x: out std_logic_vector (3 downto 0);
 end mux;
```

The architecture archmux of mux is

```
    begin
    with s select
    x <= a when "00";
    b when "10";
    c when "11";
    d when others;
end archmux;
```

The metalogical value "__" can also be used to assign don't care value to *x*.

The architecture archmux of mux is

```
begin
  with s select
   x <= a when "00";
        b when "01";
        c when "10";
        d when "11";
        "__" when others;
   end archmux;
```

A selective signal assignment describes logic based on mutually exclusive combinations of values of the selection signal.

When_else: The when_else statement provides for a conditional signal assignment which means that a signal is assigned a value based on a condition.

In the construct,

```
Signal_name <= value_a when condition 1 else
               value_b when condition 2 else
               value_c when condition 3 else
               ..............................
               value_x;
```

signal_name is assigned the value based on the conditions evaluated in the circuit. Signals are assigned when the corresponding condition is valid.

Example 4.1 ▫ Multiplexer

```
library ieee;
use ieee.std_logic_1164.all;
entity mux is port(
          a,b,c,d: in std_logic_vector (3 downto 0);
                s: in std_logic_vector (3 downto 0);
                x: out std_logic_vector (3 downto 0);
end mux;
Architecture archmux of mux is
 begin
         x <= a when (s ="00")else
         b when (s="01") else
         c when (s="10") else
         d ;
  end archmux;
```

3. Priority Encoder

Figure 9.19 shows the priority encoder of AND and OR gate and the output *j* can be expressed in terms of the following equation:

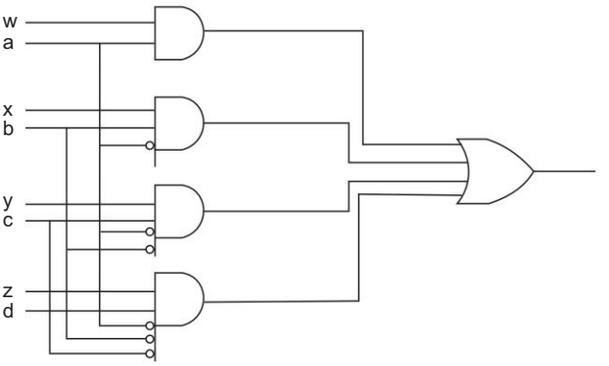


Fig. 9.19 Priority encoder

$$j = a*w + \overline{a}*b*x + \overline{a}*\overline{b}*c*y + \overline{a}*\overline{b}*\overline{c}*d*z$$

The above circuit can be described by using a `when_else` statement which indicates the priority that *j* is assigned the value of *w* when *a* is asserted even if *b*, *c* or *d* is asserted. Signal *b* holds priority over *c* and *d*. Signal *c* holds priority over *d*. So we can describe the above priority encoder as follows:

```
library ieee;
  use ieee.std_logic_1164.all;
  entity priority encoder is
  port( a,b,c,d,w,x,y,z: in std_logic;
                                  j: out std_logic);
  end priority encoder;
  architecture priority of priority is
   begin
         j <= w when a='1' else
                x when b='1' else
                y when c='1' else
                z when d='1' else
                '0';
             end priority;
```

By using Boolean equations, we can describe selection without priority encoder as given below:

```
Architecture priority of priority is
  begin
   j <= (a and w) or (b and x) or (c and y) or (d and z);
end;
```

4. Relational Operators

Relational operators are used for testing equality, inequality, and ordering. The magnitude operators (<, <=, > and >=) are defined for scalar types or an array with discrete range. Arrays are equivalent only if their lengths are equivalent and all corresponding elements of both arrays are equivalent. These operators may be overloaded. These operators are defined in numeric `std_all` of 1076.3 standard. Some operators are also defined in IEEE 1164 standards. The example of the operator is given below:

```
library ieee;
    use ieee.std_logic_1164.all;
   use work.numeric _std.all;
    entity compare is port(
         a: in unsigned (3 downto 0);
         x: out std_logic;
   end compare;
```

```
architecture compare of compare is
  begin
     x <= '1' when a=3 else '0';
end;
```

5. Component Instantiations

Component instantiations are concurrent statements that specify the interconnections of the signals in the design. Component instantiation can be used to implement combinational logic. The example of a 4-bit comparator is given below:

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_logic+_1164.all;
  entity compare is port(
      a,b: in std_logic_vector (3 downto 0);
      a eq b: out std_logic;
  end compare;
use work.gatesphg.all;
architecture compare is
   signal c: std_logic_vector(3 downto 0);
  begin
      X0: x nor2 portmap (a(0),b(0),c(0));
      X1: x nor2 portmap (a(1),b(1),c(1));
      X2: x nor2 portmap (a(2),b(2),c(2));
      X3: x nor2 portmap (c(0),c(1),c(2),c(3),a eq b);
end;
```

Gate components are not defined in VHDL standard. These components are defined in the `gates pkg. all` library. The synthesis tool may provide the components for instantiating. These components can be described in behavioral or structroral descriptions.

Sequential Statements for Combination Logic Sequential statements are the statements contained in the process. These statements are

```
if then else
          if then-elseif-else
case-when
```

The if-then-else statement can be written as

```
if condition then
          do some thing
    else
          do some thing different
    end if
```

The example is given below:

```
    library ieee
    use work.std_logic_1164.all;
    entity adder is part(
          signal step: std_logic;
          signal adder:std_logic_vector(7 down to 0)
    end entity;
    Architecture process of adder is
    begin
          if adder>x"0f" then
            step<='1';
          else
            step<='0';
          else if
    end process
```

To include the further conditions, `if-then-else` can be extended to add "`else if`" with it. It can be written as

```
if(condition 1) then
    x<=value1
else if (condition 2) then
    x<=value2
    ..else
    x<=last value
end if
```

An example of 16-bit address decoder using an `if-then-else if-else` is given below:

```
library ieee;
use ieee.std_logic_1164.all;
entity decode is part(
    address: in std_logic_vector(15 down to3);
    valid, boot_up: in std_logic;
    stram,prom,eeprom,shadow,periph1,perriph2:out_logic);
end decode;
Architecture mem_decode of decode is
begin
    mapper: process(address, valid, boot_up) is
          shadow <='0'
          prom <='0'
          periph1 <='0'
          periph2 <='0'
          stram <='0'
          eeprom <='0'
    if valid='1' ,then
          if address>=x"0000" and addres<x"4000" then
```

```
                if boot_up='1',then
                  shadow<='1';
                else
                  prom<='1';
                endif
          else if address>=x"4000" and address<=x"4008"then
                periph1 <='1';
          else if address>=x"4008" and address<=x"4010"then
                periph2 <='1';
          else if address>=x"8000" and address<=x"c000"then
                stram <='1';
          else if address>=x"c000"
                eeprom <='1';
          end if;
    end if
end process
end mem_decode;
```

Case-when: The 'case-when' statement is used to specify a set of statements to execute based on the value of a given selection signal. It is written as

```
Case selection_signal is
When value-1_of_selection_signal =>
     Do something
When value_2_of_selection_signal =>
     Do something different
When last value_of_selection_signal =>
     Do something different.
```

An example of an address decoder using `case-when` statement is given below:

```
Library ieee ;
Use ieee.std_logic_1164.all ;
Entity test_case is
        Port (address ; in std_logic_vector (2 down to 0) ;
                  Decode: out std_logic-vector (7 down to 0);
     End test_case ;
```

The architecture design of test-case is

```
      Begin
                Process (address)
      Begin
                Case address is
                  When "001" => decode    <= x"11" ;
```

```
                When "111" => decode    <= x"42" ;
                 When "010" => decode   <= x "44";
                 When "101" => decode   <= x"88" ;
          When others => decode   <= x"00" ;
        End case ;
        End process ;
        End  design ;
```

The `case-when` statement describes how decode is driven based on the input address. The word "others" is used to define behaviour of the output address for all possible values of address.

9.4 □ REPRESENTATION OF SYNCHRONOUS LOGIC USING VHDL CODES

Synchronous logic circuits are important components of digital circuits. Most device architectures have blocks of combinational logic connected to inputs of flip-flops as basic building blocks for CPLD macro cell. This section describes VHDL codes for synchronous logic. The following code describes *D*-type flip-flop (Fig. 9.20):

```
library ieee
use ieee std_logic_1164.all;
entity different_logic in part (
       d, clk: in std_logic;
       q: out std_logic ) ;
   and different_logic ;
   Architecture different of different_logic in
              begin
           process (clk) begin
            if (clk; event and clk='1') then
                        q<=d
                          and if
                     end process
                     end difference
```

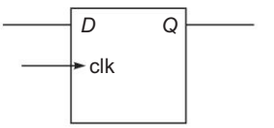


Fig. 9.20 *D*-type flip-flop

whenever the *clk* is at a high level, the output *q* is assigned the value of input *d*.

1. Example of T-type Flip-Flop

The *T*-type flip-flop or toggle flip-flop is shown in Fig. 9.21. All signal assignments in this process that occur after if (`clk 'event'` and `clk='1'`) are synchronous to the signal `clk`. The signal assignments in the process above `q<=not(q)` and `q<=q` are synchronous to the `clk`. The first signal assignment which occurs if *t* is asserted, indicates that on the rising edge of the `clk`, *q* will be assigned the opposite of it's current value. The second signal assignment which occurs if it is dissented, indicates that on the rising edge of the `clk`, *q* will retain its value.

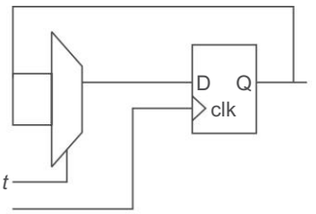


Fig. 9.21 *T*-type flip-flop

```
The VHDL code of flip-flop is given below:
library ieee;
use ieee.std_logic_1164.all;
entity if_logic is port(
    t,clk: in std_logic;
    q: buffer std_logic);
    end if_logic;
Architecture t_example of if_logic is
    if (clk'event and clk='1')then
    if (t='1'),then
    q<=not(q);
    else
    q<=q;
    end if
    end if;
    end process;
    And t_example;
```

The VHDL code of an 8-bit register is given below:

```
library ieee;
use ieee.std_logic_1164.all;
entity reg_logic is port(
    d:in std_logic_vector(0 to 7);
    clk:instd_logoc;
    q: out std_logic_vector(0 to 7);
    end reg_logic;
Architecture r_example of reg_logic is
begin
process(clk) begin
if (clk' event and clk='1')then
q<=d
end if
end process
end r_exmaple
```

`Rising_edge` and `falling_edge` Functions: The `std_logic_1164` package defines the functions `rising_edge` and `falling _edge` to detect rising and falling edges of signals. The *d* flip-flop using `rising_edge` function is described below:

```
Architecture example of d ff_logic is
begin
process(clk) begin
    if rising_edge(clk)then
    q<=d
end if;
end process;
end example;
```

The *d* flip-flop using `falling_edge` function is also described by considering `falling_edge (clk)` function instead of `rising_edge(clk)`.

VHDL codes for *D*-type flip-flop with Resets and Presets: The *D* type flip-flop with resets is given below:

The architecture example of different logics is

```
begin
process (clk, reset) begin
if (reset='1') then
q<='0';
else if rising_edge(clk)then
q<=d
end if
end process
end re_example
```

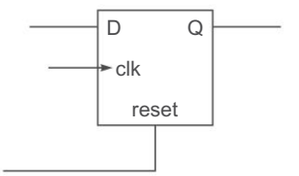


Fig. 9.22 *D*-type flip-flop with reset

The *D* type flip-flop with preset is represented by considering the `preset` function instead of `reset` function. The 8-bit register using *D* flip-flop with `reset` in Fig. 9.22 is represented with VHDL codes as follows:

```
    library ieee
    use iee.std_logic_1164.all;
    entity reg_logic is port(
          d: in std_logic_vector(0 to 7);
    reset, init, clk: is std_logic;
          q: out std_logic_vector(0 to 7);
    and reg_logic
```

The architecture example of `reg_logic` is

```
    begin
          process(clk, reset) begin
          if(reset='1') then
```

```
        q=b"00000000";
        else if (clk event and clk='1')then
        if(init='1')then
        q=b"11111111";
        else
        q<=d;
        else if
        end process
        end example
```

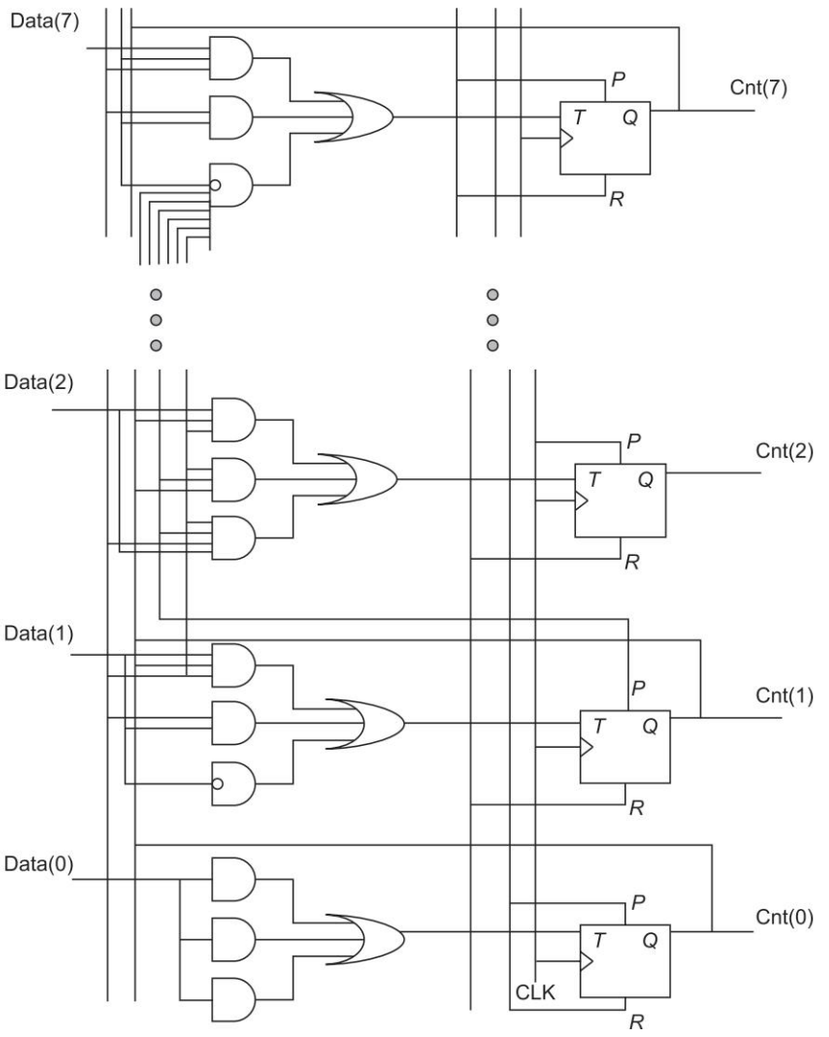


Fig. 9.23 8-bit register

2. Counter with Asynchronous Resets and Presets

In an 8-bit counter with asynchronous resets and presets, the process `count` is sensitive to `grst` and `txclk`. If the `grst` is asserted then some of the `cnt` flip-flops are asynchronously reset and others are preset. On the resting edge of `txclk`, the `cnt` registers are loaded if load is asserted, incremented if `enable` is asserted, and remain same if neither `load` nor `enable` is asserted.

The VHDL code for an 8-bit counter is given below:

```
library ieee;
 use ieee.std_logic_1164.all;
 use work.numeric_std.all;
 entity cnt8 is
    port (
    txclk, grst, enable, load: in std_logic;
    data_in: unsigned (7 downto 0);
    cnt: buffer unsignedv(7 downto 0));
 end cnt8;
```

The architecture `archcnt8` of `cnt` 8 is

```
begin
         count:process (grst, txclk)
         begin
           if grst='1' then
               cnt<="00111010";
                  else if (txclk'event and txclk='1') then
                      if load='1' then
                         cnt<=data;
                  else if (enable='1') then
                     cnt=cnt+1;
               end if;
            end if;
          end process count;
 end archcnt8;
```

9.5 □ REPRESENTATION OF THREE-STATE BUFFERS AND BIDIRECTIONAL SIGNALS

The three state buffer is used for counter. The process "`oes`" is used to describe the three-state output buffer. If '`oes`' is asserted, the value of `cnt` is assigned to `cnt_out`, (the output port of this design). If '`oe`' is not asserted, the output of the design is placed in the high-impedance state. '`oe`' is included as three-state control and `cnt_out` is acted as output.

Three-state control can be described with a `when-else` statement. We add an additional output collision which is assorted when load and enable are asserted in 8-bit counter. Three-state buffer is shown in Fig. 9.24.

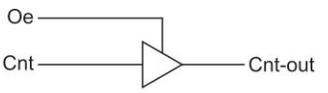


Fig. 9.24 Three-state buffer

```
Oes: process(oes,cnt)
begin
    If oe='0' then
    cnt_out<=(others=>'z');
               else
    cnt_out<=cnt;
end if
end process oes;
```

The bidirectional signals are described with little modification in an 8-bit counter with a three-state buffer. The counter is loaded with current value on the pins associated with counter outputs meaning that the value loaded when the load is asserted may be the counter's previous value or a value driven onto the pins from another device, depending upon the state of the output enable.

The architecture `arch cnt8` of `cnt8` is

```
signal cnt: std_logic_vector(7 downto 0);
begin
count: process (grst,txclk)
begin
if grst='1' then
cnt="00111010";
      else if (txclk'event and txclk='1') then
      if load='1' then
      cnt<=cnt+1;
      end if
                        end process(count)
 Oes: process (oe,cnt);
         begin
               if oe='0' then
                   cnt_out<=(others<='z');'
                       else
                             cnt_out<=cnt;
                     end if;
              end process oes;
end arch cnt8;
```

9.6 ☐ DESIGNING FIFO USING VHDL CODE

FIFO is introduced to design or create combination logic and synchronous logic. FIFO in full form is first in first out. Here, when a read signal, 'rd' is asserted, the data output of FIFO, data-out is enabled and when it is not asserted, the output is in high-impedance state. When write signal '`wr`' is asserted, the data is loaded into one of the output registers. The signals '`rdptr`' and `wrptr` are used to reset the registers to read and write, respectively. The signals '`rdine`' and '`wrine`' are used to increment the read and write pointer that indicate which register to read and write. Figure 9.25 shows FIFO diagram. The signal '`rst`' is used to reset the counters and registers.

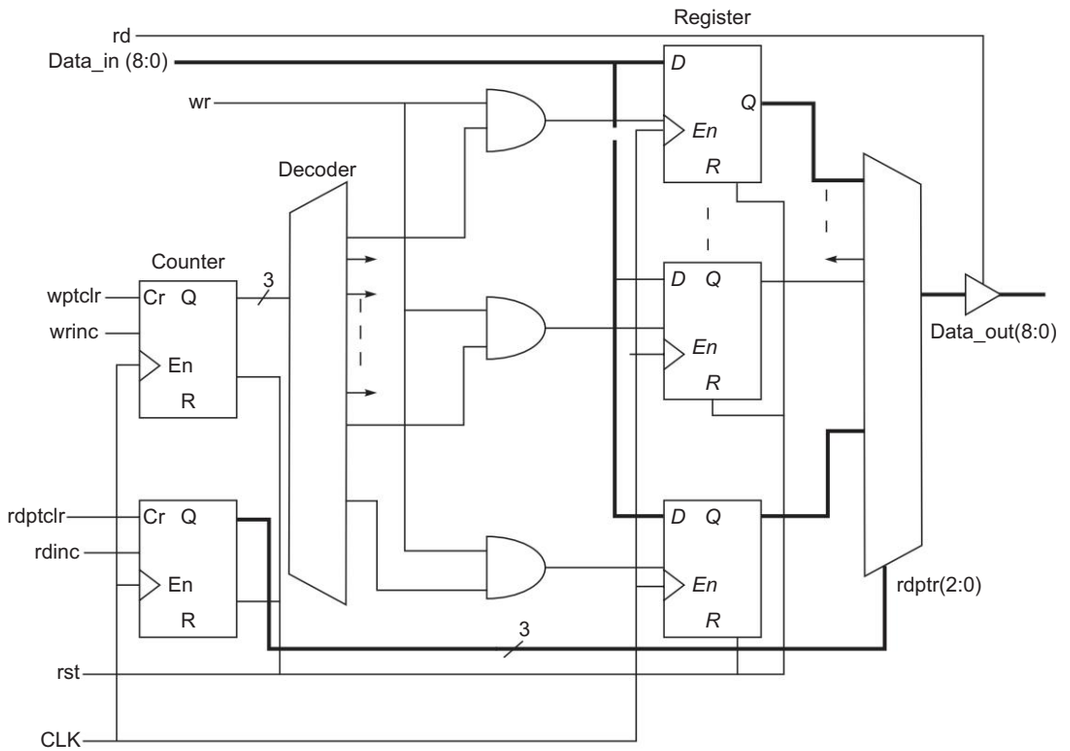


Fig. 9.25 FIFO block diagram

FIFO is implemented by using heirarchy and structural design. The VHDL codes of FIFO design is given below:

```
    library ieee;
        use ieee std_logic_1164.all;
        use ieee workstd_arith.all;
           entity fifoxbby is
                generic (circle; integer:=32)
                port (
```

```
                clk,rst, oe: in std_logic;
              rd, wrdine, rdinc, rdptrclr, wrptrclr: in std_logic;
                 data-in: in std_logic_vector(wide down to 0);
                 data_out: out std_logic_vector (wide down to 0));
          end fifoxbyy;
```

Architecture `archfifoxbby` of `fifoxbby` is

```
      constant deep: integer:=20;
      type fifo_array is array (deep down to 0) of std_logic_vector
(wide down to 0)
        signal fifo: fifo_array;
          signal: wrptr,rdptr: integer range 0 to deep;
          signal en: std_logic_vector(deep downto 0);
              begin
       reg_array: process (rst, clk)        --fifo register array
                        begin
                              if rst='i' then
                                    for i in fifo'range loop
                                          fifo(i) <=(others=>'0');
                                    end loop;
                              else if rising_edge (clk) the3n
                                  if wr='1' then
                                   fifo(wrptr)<=data_in;
                                  end if;
                             end process;
          read_count: process(rst,clk);          --reader pointer
                     begin
                         if rst='1' then
                          rdptr<='0';
                         elseif rising_edge(clk) then
                        if rdptr_ctr='1' then
                           rdptr<='0';
                         elseif
                          rdine<='1' then
                          rdptr=rdptr+1;
              endif;
          endif;
        end process;
   write_count: process(rst,clk)                    --writer pointer
                begin
                    if rst='1' then
                         wrptr<='0';
                        else if rising_edge(clk) then
                            if wrptrctr='1' then
```

```
                                wrptr<=0;
                               else if wrine='1' then
                               wrptr=wrptr+1;
                          end if;
                     end if;
                end process;
                                          -- Data output multiplexer
          three_state: process (oe, dmuxout)
                begin
                    if oe= '1' then
                            data_ out <= dmuxout;
                    else
                         data_out <= (others => 'Z');
              end if;
        end  process;
end archfifoxbyy;
```

1. Loops

Loop statements are used for implementations of repetitive operations. There are `for` loops and `while` loops. The for loops are used for execution of limited number of operations based on a controlling value, whereas the `while` statement is used to execulte an operation as long as a continued logical condition evaluates true.

(a) `For` Loop Statement The `for` loop statement example is given below. In the example of the FIFO register, when `rst` is asserted, all the FIFO registers are reset except for FIFO(4) register:

```
reg_array: process (rst, clk)
    begin
    if rst = '1' then
    for  i in 7 downto 0 loop
         if i=4 then
     next;
  else
     fifo( i ) <= (others => '0');
end loop;
```

(b) `while` Statement The `while` statement example is given below. In the example of the FIFO register, when `rst` is asserted, all the FIFO registers are reset except for FIFO(4) register:

```
  reg_array: process (rst, clk)
  variable i: integer:= 0;
  begin
           if rst = '1' then
```

```
            while  i = 7  loop
            if i=4 then
          next;
        else
            fifo( i ) <= (others => '0');
               i:= i +1;
end loop;
```

2. Exiting Loop

The exiting loop is used to exit a loop and it checks for an illegal condition. The example of a FIFO register is shown below:

This statement can terminate at compile time.

```
    reg_array: process (rst, clk)
                    begin
                         if rst = '1' then
                         loop1:  for  i in deep downto 0 loop
                                    if i >20 then
          exit  loop1;
        else
            fifo( i ) <= (others => '0');
                 end loop;
```

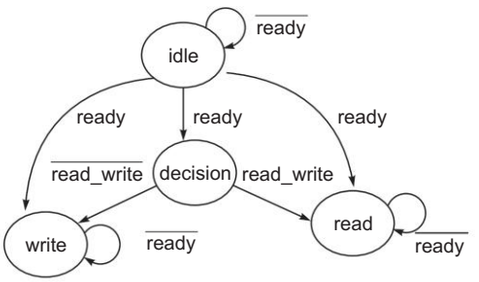
3. State Machine Design in VHDL

In traditional digital circuit design, the first step is to construct a state machine diagram from which we can derive a state table. In the state diagram, we can find the equivalent states by matching rows of the state table and we can generate next states corresponding to present states from this table. Figure 9.25 shows a simple state machine diagram consisting of read, write, decision, and idle states. The diagram presents that read and write transaction operates with the assertion of ready in which case the state machine transition is from idle state to the decision state. Depending on the value of `read_ write` during the next clock cycle, the transaction is either a read or written and the state machine transitions to the appropriate state. A transaction is completed when ready is asserted, placing the controller back in idle state. If ready is not asserted, placing then the controller back in the current state. All the states require different inputs to the transition to the next state. The combination of the inputs provide Karnaugh map, from which the logic equations are derived.

The state machine diagram of Fig. 9.26(a) can be translated into transition tables and from these tables, the output equations can be derived by using Karnaugh maps as shown in Fig. 9.26(b).

The state diagram can be described by VHDL codes without having to perform the state assignment, generate the state translation table or determine the next state equations based on the types of flip-flops available. In VHDL codes, each state can be described by using a `case-when` statement and

if-then-else statements are used for state transitions. We begin with state names and declaration of signals and represent state transitions by using `case-when` statements for an example of Fig. 9.25(a).



(a) State machine

State	q_0q_1	read_write / ready: 00	01	11	10	Outputs oe	we
idle	00	00	01	01	00	0	1
decision	01	11	11	10	10	0	0
write	11	11	00	00	11	0	1
read	10	10	00	00	10	1	0

q_0q_1	Q_0 read_write. ready 00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	0	0	1
00	1	0	0	1

$Q_0 = \bar{q}_0\, q_1 + q_1.\overline{ready}$

q_0q_1	Q_1 read_write. ready 00	01	11	10
00	0	1	1	0
01	1	1	0	0
11	1	0	0	1
00	0	0	0	0

$Q_0 = q_0\, q_1$. ready + $q^0q^1.\overline{read_write}$ + $q^0q^1.\overline{ready}$

Fig. 9.26 (a) State machine (b) Its transition table

```
entity example_ state_ machine is port (
       read_write, ready, clk:     in   bit;
       oe, we                :     out bit;
end example_ state_ machine;
```

The architecture state_machine is

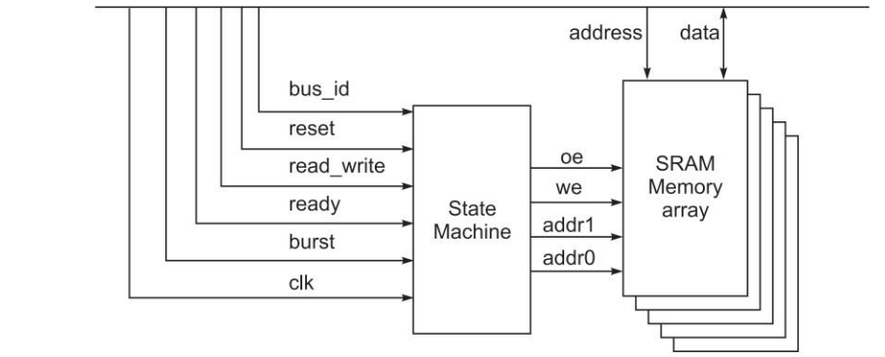
```
  type StateType is (idle, decision, read, write);
  signal present_state, next_state: StateType;
  begin
sate_comb: process (present_state, read_write, ready) begin
case present_state is
```

```
when idle =>    oe <= '0'; we <= '0';
           if ready ='1' then
           next_state <= decision;
            else  next_state <= idle;
            end if;
when decision =>    oe <= '0'; we <= '0';
              if read_write ='1' then
              next_state <= read;
              else  next_state <= write;
              end if;
when read =>    oe <= '1'; we <= '0';
     if ready ='1' then
     next_state <= idle;
     else  next_state <= read;
     end if;
     when write =>    oe <= '0'; we <= '1';
     if ready ='1' then
     next_state <= idle;
     else  next_state <= write;
     end if;
     end case
     end process state_comb;
     state_clocked: process(clk) begin
     if clk'event' and clk = '1' then
     present_ state <= next_state;
     end if;
     end process state_clocked;
     end architecture state_machine
```

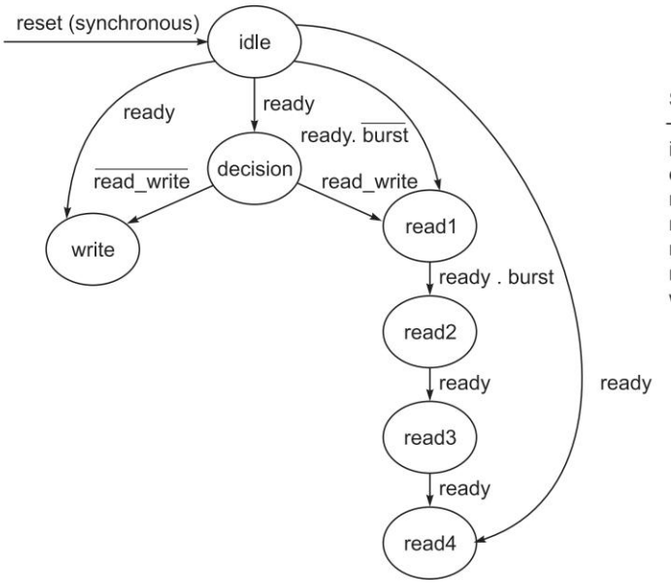
In the above VHDL code, there are two FSM processes—clock process and state process. So by using VHDL codes, we can have a system consisting of multi-FSM processes.

4. Representation of Memory Controller using FSM Processes

Figure 9.26 (a) shows block diagram of a memory controller consisting of a bus with `burst`, `read_write, clk` signals, etc. to access SRAM memory arrays. The `read_write` signal is asserted to indicate a read from the memory buffer and deasserted to indicate a write to the memory buffer. If the memory access is a read, the read may either be a single word read or a four-word burst read. A write to the buffer is always a single word write but never a burst. If the access is a write (`read_write =0`), the controller goes to the write state. Figure 9.27(b) represents state-machine flow diagram.



(a) Memory controller



(b) State machine flow diagram

State	oe	we	addr[1:0]
idle	0	0	00
decision	0	0	00
read1	1	0	00
read2	1	0	01
read3	1	0	10
read4	1	0	11
write	0	1	00

(c) Transition table

Fig. 9.27 Memory controller, state diagram, and transition table

The VHDL codes of memory controller with two FSM processes (shown in Fig. 9.26) are given below:

```
entity example_ state_ machine is port (
read_write, ready, bus_id, burst:     in   bit;
oe, we, addr1, addr2                      :     out bit;
end example_ state_ machine;
```

```
architecture state_machine_memory_controller is
type StateType is (idle, decision, read1, read2, read3,
read4, write);
     signal present_state, next_state: StateType;
     begin
     state_controller: process (present_state, bus_id, burst,
     read_write, ready) begin
     case present_state is
     when idle =>    oe <= '0'; we <= '0'; addr<= "00";
     if bus_id = '11110011' then
     next_state <= decision;
     else  next_state <= idle;
     end if;
     when decision =>    oe <= '0'; we <= '0'; addr<= "00";
     if read_write = '1' then
     next_state <= read1;
     elseif (burst= '0') then
     next_state <= idle;
     else
     next_state<=read2;
     end if;
     when read2 =>    oe <= '1'; we <= '0'; addr<= "01";
     if ready ='1' then
     next_state <= read3;
     else  next_state <= read2;
     end if;
     when read3 =>    oe <= '0'; we <= '0'; addr <= "10";
          if ready ='1' then
     next_state <= read4;
     else  next_state <= read3;
     end if;
     when read4 =>    oe <= '1'; we <= '0'; addr <= "11";
     if ready ='1' then
     next_state <= idle;
     else  next_state <= read4;
     end if;
     when write =>    oe <= '0'; we <= '1'; addr <= "00";
     if ready ='1' then
     next_state <= idle;
     else  next_state <= write;
     end if;
     end case
```

```
      end process state_controller;
      state_clocked: process(clk) begin
      if clk'event' and clk = '1' then
      present_ state <= next_state;
      end if;
      end process state_clocked;
      end architecture state_machine_memory_controller
```

9.7 □ HIERARCHY IN A LARGE DESIGN

To design or represent a large circuit, it is required to break the large design into manageable units. The breaking of the large design into manageable units creates hierarchy. The advantages of hierarchy are given below:

- Design of manageable circuit units in parallel with other engineers
- Verification of each component in design simultaneously
- Building the design in stages interfacing the constituent component at a time
- Fewer initial errors and quicker debugging time
- Reduction of design complexity

For this type of design, libraries, and packages are used to declare and store components. A library is a directory where design entity declaration and architecture body pairs are compiled. A example is given below:

```
library ieee;
```

A package is a design unit consisting of a package declaration and a package body. It can be used to make its type, component, and function, and other declarations cannot be made visible to other design units. The package body does not declare any function or a procedure that does not require an associated package body. Component declaration must be visible in a package. As for example, the component *D* flip-flop is described as given below:

```
       library ieee;
                 use ieee. std_logic_1164.all;
                 package dflop_pkg is
                      component dflop port (
                           clk, reset,d: in std_logic;
                           q: buffer std_logic);
end component;
end dflop_pkg;
                library ieee;
                 use ieee. std_logic_1164.all;
                 entity dflop port (
                           clk, d: in std_logic;
                           q: out std_logic);
                     end dflop;
```

```
architecture archdflop of dflop is
      begin
           process
              wait until clk = '1' ;
                 q <= d;
          end;
     end;
```

The entity declaration architecture body and package declaration are all design units for component the *D* flip flop.

We can group the component declarations for the design entities into packages. We consider three packages—`regs_pkg`, `counter_pkg`, and `synch_pkg`—and combine them in the same design file called as library basic. These packages are described below:

```
       library ieee;
                use ieee. std_logic_1164.all;
                use work.numeric_std.all
                package regs_pkg is
                     component rdff1 port (
                          clk, reset,d: in std_logic;
                          q: buffer std_logic);
end component
component rdff
generic(size:integer:=2);
port clk, rest: in std logic;
                d: in std_logic_vector(size- 1 down to 0);
                q: buffer std_logic_vector(size- 1 down to 0);
end component
component rregl port (
                      clk, rest, load: in std _logic;
                      d: in std_logic;
                q: buffer std_logic);
end component
component rreg
             generic(size: integer:=2);
             port(clk, reset, load: in std_logic;
                 d: in std_logic_vector(size- 1 down to 0);
                 q: buffer std_logic_ vector(size- 1 down to 0);
end component
component reg generic(size: integer:=2);
            port(clk, reset, load, pst: in std_logic;
                 d: in std_logic_vector(size- 1 down to 0);
                 q: buffer std_logic_ vector(size- 1 down to 0);
```

```
end component
component ureg
            generic (size: integer:=2);
            port (clk, reset, load: in std_logic;
                  d: unsigned(size- 1 down to 0);
                  q: buffer unsigned(size- 1 down to 0);
end component;
end regs_pkg;
library ieee;                                    -- counter package
use ieee_std_logic_1164.all;
package counter_pkg is
        component as count
        generic (counter size: intger:=2);
        port(clk, areset, sreset, enable: in std_logic;
        count: buffer std_logic_ vector(countersize- 1 down to 0));
end component;
end counter.pkg;
library ieee;
use ieee_std_logic_1164.all;
package synch_pkg is component port (
          clk, reset, d: in std_logic;
          q: buffer std_logic);
end component;
component psynch port (
          clk, preset, d: in std_logic;
          q: buffer std_logic);

end component;
end synch_pkg;
```

These packages are compiled into the main basic library and can be accessed via library and use clauses. The clause is given below:

```
library basic;
use basic.regs_pkg.rreg';
          or
library basic;
use basic_regs_pkg.all:
```

In this way, we can develop the packages. By this type of hierarchy, we can design and describe many devices such as RAM consisting of sixteen 4-bit wide registers. The codes are given below:

```
  library ieee;
  use iece.std_logic_1164.all;
  use work.numeric_std.all;
  use work.mnemerics_std.all;
  use basic.regs_pkg.all;
      entity ram_regs is port C
            clck, rst: in std_logic;
      a,bf:in unsigned(3 down to 0);
  dest_ctl:in std_logic_vector(2 down to 0);
  ram0,ram3:in out std_logic;
  ad, bd: buffer unsigned(3 down to 0);
  end ram_regs;
  architecture ram_regs of ram_regs is
      signal ram_en: std_logic;
      signal data: unsigned(3 down to 0);
      signal en: std_logic_vector(15 down to 0);
     type ram_array is array(15 down to 0) of unsigned(3 down to 0);
      signal ab_data: ram array;
  begin
      gen: i in 15 down to 0 generate       - - define register array;
            ram: urge generic map(4)
            port map(clk, rst, en(i), data, ab_data(i));
            end generate;
  with dest_ctl select   - - decode b to determine which register is
enable;
   ram_en b<=0 when greg1 nop;
                   'i' when others;

  decode_b: process(b)
  begin
            for i  in 0 to 15 loop
               if to_integer(b)=I then en(i)<=ram_en;
  else en(i)<='0';
  end if;
  end loop;
  end process;
                              - - define data input to register array:
  with dest_ctl select
  data <= (f(2), f(1), f(0), ram0) when ramqu | ramu, - - shift up
                        (ram3, f(3), f(2), f(1)) when ramqd | ramid,
                                                        - - shift down
                       f when rama | ramf;
                "----" when others;
```

```
- - define reg_array output for a and b regs;
            ad <= ab_data(to_integer(a));
            bd <= ab_data(to_integer(b));
- - define ram0 and ram3 inputs;
   ram3 <= f(3) when dest_ctl = ramu or dest_cll=ramqu) else 'z';
  ram0 <= f(0) when dest_cll = ramq or dest_cll=ramqd) else 'z';
end ram_regs;
```

Out Mux

The `out_Mux` component contains a two-to-one multiplexer with select links that are decoded by the destination-control microinstruction. The three-state input is also controlled by a dedicated output enable. The codes are given below:

```
Library ieee;
Use ieee.std_logic_1164.all;
Use work.numeric_std.all;
Use work.mnemerics_std.all;
entity out_mux is port C
          ad, f: in unsigned(3 down to 0);
          dest_ctl: in std_logic vector(2 down to 0);
    oe: in std_logic;
          y: buffer unsigned(3 down to 0);
end out_mux;
architecture out_mux is
signal y_int: unsigned(3 down to 0);            - - output before
tri-state buffer
begin
                  y_int <= ad when dest_ctl = rama else f;
           y <= y_int when oe = 'o' else "zzzz";
end out_mux;
```

9.8 □ FUNCTIONS AND PROCEDURES

Functions and procedures are high-level design constructs that compute values or define process for type conversions, operator overloading or as an alternative to component instantiation. Although all of these have application for synthesis, the functions of most have already been defined in IEEE 1076, 1164, and 1076.3 standards. The functions `to-integer`, `to-signed`, and `to-unsigned` are defined in numeric-standard when combining types in arithmetic operations. These are type-conversion functions. Below, Boolean to bit-type conversion function is shown.

```
Function b/2 bit (a: BOOLEAN) return bit is
begin
     if a then
          return '1';
     else
          return '0';
     end if;
end b/2;
```

The above type function is used to convert a signal of type Boolean to type bit. We can define the conversion function for `bit_vector` to integer as `bv2i`

```
Function bv2i (bv: bit_vector) return integer is
    Variable result, abit: integer:=0;
    Variable count        : integer:=0;
begin
               bits: for I in bv 'low to bv 'high loop
                      a bit:=0;
                If (bv(I) = '1') then
                abit: =  2** (I - bv'low);
                end if
                result: = result+abit;     -- add in bit if '1';
          count:= count + 1;
          exit bits when count =32;                -- 32 bits max;
    end loop bits;
return (result);
end bv2I;
```

The function '`bv2I`' takes a `bit_vector` as input. It performs a binary to decimal conversion and as an integer.

The function '`i2bv`' is used for conversion of integer to `bit_vector` as given below:

```
  function i2bv (VAL, width: INTEGER) return bit_vector is
  variable result: bit_vector (0 to width - 1): = (other => '0');
  variable bits: INTEGER:=width;
  begin
                       if (bits > 32) then
                  bits: = 32;
          else
                    ASSERT 2** bits > VAL REPORT
                         "value too big for bit_vector width"
               SVERITY WARNING;
               end if;
            for I in 0 to bits - 1 loop
```

```
            if (val/(2**i)) MOD 2 =1 then
                            result (i): ='1';
            end if;
            end loop;
```

The function takes an integer value as its inputs and the size (width) of the `bit_vector` that is to be returned. The function performs a decimal-to-binary conversion and returns the value of the integer as a `bit_vector`.

The function `Inc_bv` takes a `bit_vector` as its inputs and increments the value of that `bit_vector` and returns a `bit_vector` of the same size as the input `bit_vector`. The function `Inc_bv` is described by the following code.

```
      function Inc_bv (a: bit_vector) return bit_vector is
                        variable 3: bit_vector (a 'range);
                         variable carry: bit;
                  begin
                          carry: ='1';
                          for 1 in  a 'low to a 'high loop
                           s(i): = a (i) xor Carry;
                           carry: = a(i) and Carry;
                   end loop;
                   return(s);
         end Inc_bv;
```

All these type-conversion functions and increment functions can be declared in a package declaration and defined in the associated package body. There are some standard functions which are defined in standard packages like 1164-all and 1076.3-all.

Procedures

Like function, procedures are high-level design constructs to compute values or define processes that we can use for conversion function. Procedures can return more than one value. All statements in procedures cannot declare signals. But variables can be declared in the declarative region of a procedure.

```
       Librmary IEEE ;
       Use IEEE.std_logic_1164.all ;
       Package my flip flops is
       Procedure dff (signal d: bit_vector ; signal clk:bit ;
      signal q: our.bit_vector );
    Procedure dff ( signal d: bit_vector ; signal clk, rst,pst:bit ;
  signal q, q_bar: our.bit_vector );
    Procedure dff ( signal d: std_logic_vector ; signal clk,
rst,pst:std_logic ;
```

```
   signal q,q_bar: our.std_logic_vector );
          end my flip flop;
          Package body my flip flops is
              Procedure dff (signal d: bit_vector ; signal clk:bit ;
  signal q: our.bit_vector) is
                    begin
            if clk event and clk = '1' then
       q <= d ;
        end if;
  end procedure;
                 Procedure dff (signal d: bit_vector ; signal clk,
rst,pst:bit;
  signal q, q_bar: our.bit_vector) is
            begin
            if rst ='1' then q <= (other => '0');
             elseif pst = '1' then q <= (other => '1');
       elseif clk event and clk = '1' then
           q <= d ; q_bar <=  not (d) ;
        end if ;
  end procedure ;

  Procedure dff (signal d: std_logic_vector ; signal clk, rst,pst:std_
logic ;
   signal q,q_bar: our.std_logic_vector) is
            begin
            if rst = '1' then q <= (other => '0');
            elseif pst = '1' then q <= (other => '1');
                   elseif clk 'event and clk =1' then
            q <= d ; q <= not(d) ;
            end if ;
  end procedure ;
```

9.9 □ PIPELINING

Pipelining is used to take data path operation that is currently executed over one clock cycle and break it up into smaller operations that are executed in multiple shorter clock cycles. Figure 9.28 shows a large data-path operation broken into smaller data-path operations. If the time for large data-path operation is '*x*', and we break the large data path into three equal parts, then each part has a `tpd` of '*x*/3' which permits system frequency '3/*x*'—three times larger than that of the large data-path frequency '1/*x*' . The advantage of a pipeline is to increase the performance with expense of clock cycles and additional logic resources for registering.

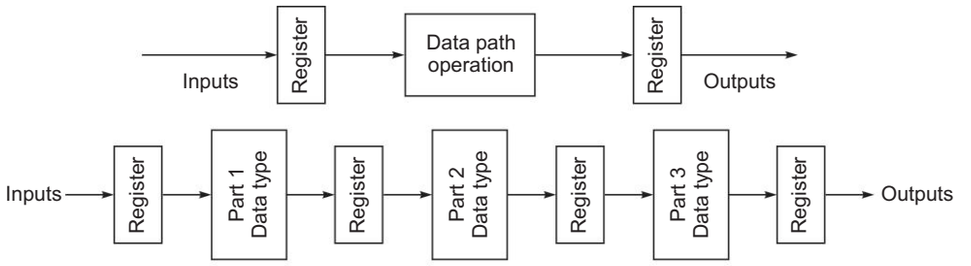


Fig. 9.28 Pipelining of data path

The pipelining is used in a DRAM controller to decrease clock-to-output timer. A pipeline stage can be introduced into DRAM controller as shown in Fig. 9.29.

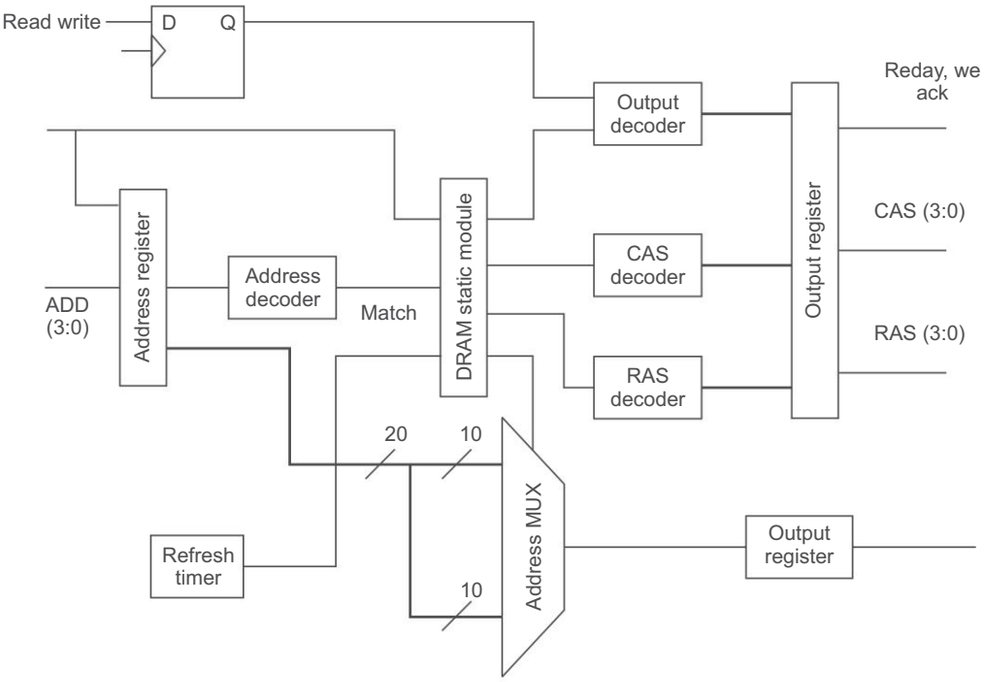


Fig. 9.29 DRAM controller with output register using pipelining

For pipelining of the output signal of a DRAM controller, the following signals are described for feeding the output register:

```
ack_q: out std_logic;                    ..... Acknowledgement
we_q : out std_logic;                    ..... Write enable
ready_q : out std_logic;                 ..... Data ready for latching
dram_q: out std_logic_vector (9 down to 0); .....DRAM address
```

```
  ras_q: out std_logic_vector (1 down to 0);  .....Row address
strobe
  cas_q: out std_logic_vector (3 down to 0);  .....Column address
strobe
```

There are output signals in the architecture which are described in the following manner.

```
  Signal ack, we, ready: std_logic;
  Signal dram: std_logic_vector (9 down to 0);
  Signal ras: std_logic_vector (1 down to 0);
  Signal cas: std_logic_vector (3 down to 0);
```

Within the body of the architecture, we can describe the output register with pipelining:

```
  Output register
      regd: process (reset, clock);
      begin
            if reset = '1' then
                  ack_q <= '1'; we_q <= '1'; ready_q <= '1';
  cas_q <= '1111'; ras_q <= '11';
  dram_q <= (other => '0');
            else if clock 'event and clock = '1' then
                  ack_q <= ack ; we_q <= we ; ready _q <= ready;
                  cas_q <= cas ; ras_q <=ras; dram_q<= dram;
            end if
      end process ;
```

Carry Look-Ahead Adder

Carry look-ahead adder schemes are used to reduce delays associated with carry chains. Figure 9.30 shows the construction of an adder of x_n bits. The size of the n-bit group adder affects the area and performance of the carry look-ahead logic. Each n-bit group adder has three outputs—a sum, carry generate, and carry propagate. A **carry generate** indicates that a group adder produceds a carry-out in the next stage, regardless of the value of the carry into this group. A `carry propagate` indicates that group adder produces a carry-out to the next stage only if the value of the carry into this stage is '1'. The equation of a 1-bit carry generate and propagate (G and P respectively) and the carry-out to the next are given below:

$$G = A * B$$
$$P = A + B$$

and

$$C_{out} = G + G_{in} * P$$

We distinguish a 1-bit carry generate and propagate terms from n-bit carry generate and propagate terms by using the letters E and R for group generate and propagate. The following code describes a carry look-ahead adder having an 8-bit adder with a 2-bit group adder.

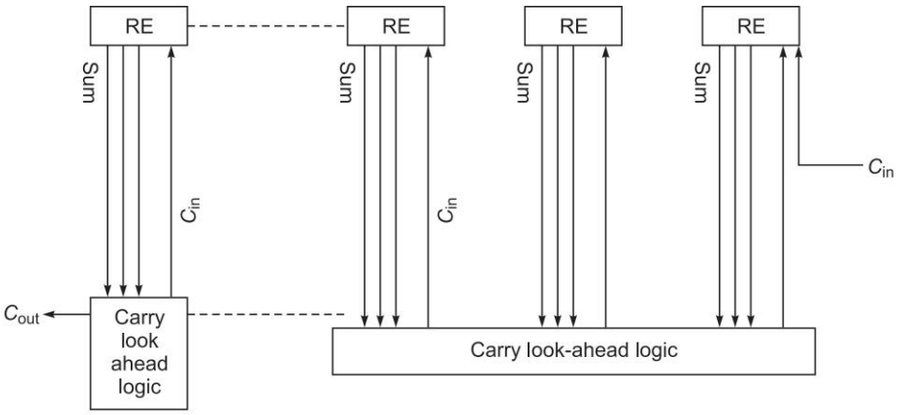


Fig. 9.30 Carry look-ahead adder

```
  library ieee ;
      Use ieee std_logic_1164.all;
      entity fcadd2 is
      port (Cin, a0, b1, b0 : bit;
      sum1, sum0, e, r:out bit);
      end fcadd2 ;
  architecture archfcadd2 of fcadd2 is
            begin
                  sum0 <= a0 xor b0 xor Cin;
                  sum1 <= a1 xor b1 xor (a0 and b0) or (a0 and Cin) or
(b0 and Cin);
                  e <= (a1 and b1) or (a1 or b1) and (a0 and b0) ;
                  r <= (a1 or b1) and (a0 or b0) ;
  end archfcadd2;
  entity fcadd8_2 is
      port (Cin: in bit ; a7, a6, a5, a4, a3, a2, a1, a0: in bit;
                     b7, b6, b5, b4, b3, b2, b1, b0: in bit;
                      sum7, sum6, sum5, sum4, sum3, sum2, sum1, sum0, C0:
out bit);
  end fcadd8_2;
  architecture archfcadd8_2 of fcadd8_2 is
      component fcadd2 port (Cin , a1, a0, b1, b0: in bit;
                              sum1, sum0, e, r: out bit);
      end component;
      signal C2,C4,C6:bit;
      attribute sysnthesis_off of e1, e2, e3:signal is true;
```

```
    signal r0, r1, r2 , r3: bit ;
          attribute snthesis_off of r1, r2 , r3: signal is true ;
    begin
    U1: fcadd2 port (Cin , a1, a0, b1, b0, sum1, sum0, e0, r0);
    U2: fcadd2 port (C2, a3, a2, b3, b2, sum3, sum2, e1, r1);
    U3: fcadd2 port (C4, a5, a4, b5, b4, sum5, sum4, e2, r2);
    U2: fcadd2 port (C6, a7, a6, b7, b6, sum7, sum6, e3, r3);
                C2 <= e0 or (r0 and Cin);
                C4 <= e1 or (r1 and e0) or (r1 and r0 and Cin);
                C6 <= e2 or (r2 and e1) or (r2 and r1 and e0)
or (r2 and r1 and r0 and Cin);
C0 <= e3 or (r3 and e2) or (r3 and r2 and e1) or (r3 and r2 and r1 and
          e0) or (r3 and r2 and r1 and r0 and Cin);
    end archfcadd8_2;
```

REFERENCES

9.1 Aoshima, A ., and E. Kuh, *Multi-Channel Optimization in Gate Array LSI Layout*, in International Conference on Computer-Aided Design, IEEE, 1983.

9.2 Brown, S., J. Rose, and Z. Vranesic, A *Detail Router for Field-Programmable Gate Arrays*, in International Conference on Computer-Aided Design, pp. 382–385, IEEE/ACM, 1990.

9.3 Brown, S., J. Rose, and Z. Vranesic, *A Detailed Router for Field-Programmable Gate Arrays*, IEEE Transactions on Computer Aided Design 11:620-628, 1992.

9.4 Brown, S., J. Rose, and Z. Vranesic, *A Stochastic Model to Predict the Routability of Field-Programmable Gate Arrays*, IEEE Transactions on Computer Aided Design 12:1827–1838, 1993.

9.5 Farrahi, A.H., and M. Sarrafzadeh, *Complexity of the Look-up Table Minimization Problem for FPGA Technology Mapping*, IEEE Transactions on Computer Aided Design 13(11):1319–1332, 1994 .

9.6 Farrahi, A.H., and M. Sarrafzadeh, *FPGA Technology Mapping for Power Minimization*, in International Workshop on Field-Programmable Logic and Applications, pp. 66–77, Springer-Verlag, Berlin, 1994.

9.7 Murgai, R., N. Shenoy, R.K. Brayton, and A. Sangiovanni-Vincentelli, *Performance Directed Synthesis for Table Look Up Programmable Gate Arrays,* in International Conference on Computer-Aided Design, pp. 572–575, IEEE, 1991.

9.8 Pedram, M., B. Nobandegani, and B. Preas, *Design and Analysis of Segmented Routing Channels for Row-Based FPGAs,* IEEE Transactions on Computer Aided Design 13:1470–1479, 1994.

9.9 Rose, J., and S. Brown, *Flexibility of Interconnection Structures in Field-Programmable Gate Arrays*, IEEE Journal of Solid-State Circuits 26:277–282, 1991.

9.10 Wu, Y., and M. Marek-Sadowska, *An Efficient Router for 2-D Field-Programmable Gate Arrays,* in European Design Automation Conference, 1994.
9.11 Yeap, G., M. Sarrafzadeh, and C. K. Wong, *A Global Routing Algorithm for Array Architecture FPGA Environment*, unpublished paper, 1993.
9.12 Zhu, K., and D. F. Wong, *On Channel Segmentation Design for Row-Based FPGA's,* in International Conference on Computer-Aided Design, pp. 26–29, IEEE, 1992.
9.13 Ashenden, Petre, *The designer's guide to VHDL,* San Fransisco Morgan Kaufmann, 1995
9.14 Dewey, A, and Anthony, G., *VHDL motivation,* IEEE design and test of computers, New work, IEEE, 1986.
9.15 Lowenstein, A., and Winter, G., *VHDL's impact on test*, IEEE design and test of computers, New work, IEEE, 1986
9.16 Saunders, L., *The IBM VHDL design systems,* Proceedings of the 24th degn automatic conference, 1987.
9.17 Leung, S., and Michael, S., *ASIC system design with VHDL: a Paradigm*, Boston, Kluwer academic, 1989.
9.18 Coppola, A., and Lewis, J., *VHDL for programmable logic devices,* 1993 programmable logic devices conference, Santa Clara, March, 1993
9.19 Brown, S., *Field programmable devices*, Los Gotos Calif, Stan Baker Associates, 1995.
9.20 Bhasker, J., *A VHDL premier*, Eaglewood Cliff, N. J., Prentice Hall, 1992.
9.21 Cypress Semiconductor Corp., *Programmable logic data book,* San Jose Calif, 1996.

EXERCISES

9.1 Explain different steps of design-tool flow for implementation of digital circuit in FPGA with VHDL tool.
9.2 What is CPLD? How can the digital circuit be implemented in a CPLD?
9.3 What is FPGA? Describe architecture of different types of FPGA. Mention advantages and disadvantages of circuit implementation in FPGA.
9.4 Explain operation of multiplexer-based configurable logic block (CLB) and look-up-table-based CLB.
9.5 Describe SRAM based FPGA with issues.
9.6 List the features that are used to classify a device as a PAL, CPLD, FPGA, or an ASIC.
9.7 Write the entity declaration for a 2 bit equality comparator.
9.8 Write the entity declaration of the following architecture, assuming that all signals in the architecture are ports:

```
architecture archmux of mux is
 begin
   with s select
    x <= a when "00";
```

```
        b when "10";
        c when "11";
        d when others;
    end archmux;
```

9.9 Differentiate behavioral data flow and structural description for architecture body of VHDL code with an example.

9.10 Which description can be preferred to describe the following circuit (Fig. P9.1)?. Write the VHDL code of the circuit with behavioral description.

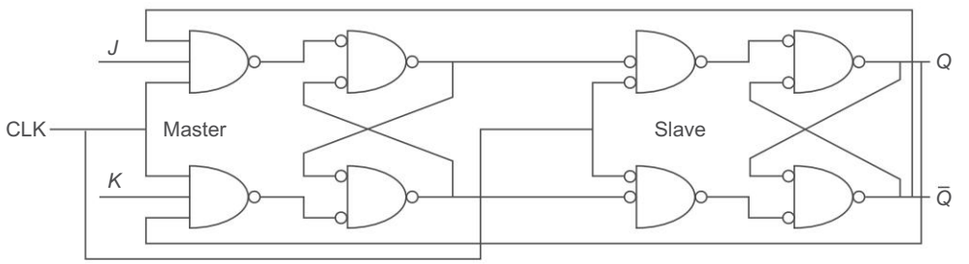


Fig. P9.1

9.11 Write the VHDL code of the above circuit (Fig. P9.1) with dataflow description.

9.12 Identify errors in the following code:

```
entity priority encoder is
  port(a,b,c,d,w,x,y,z: in std_logic;
                                  j: out std_logic);
  end;
  architecture priority of priority is
   begin
         j <= w when a='1' else
                x when b='1' else
                y when c='1' else
                z when d='1' else
                '0';
              end priority;
```

9.13 Write the VHDL code of a 4-bit magnitude comparator with three outputs (equals, less than, and greater than) by using

(a) logical operators
(b) relational operators
(c) structural installation
(d) a `when-else` statement
(e) an `if-then-else` statement

9.14 Describe the following circuit with structural description of VHDL.

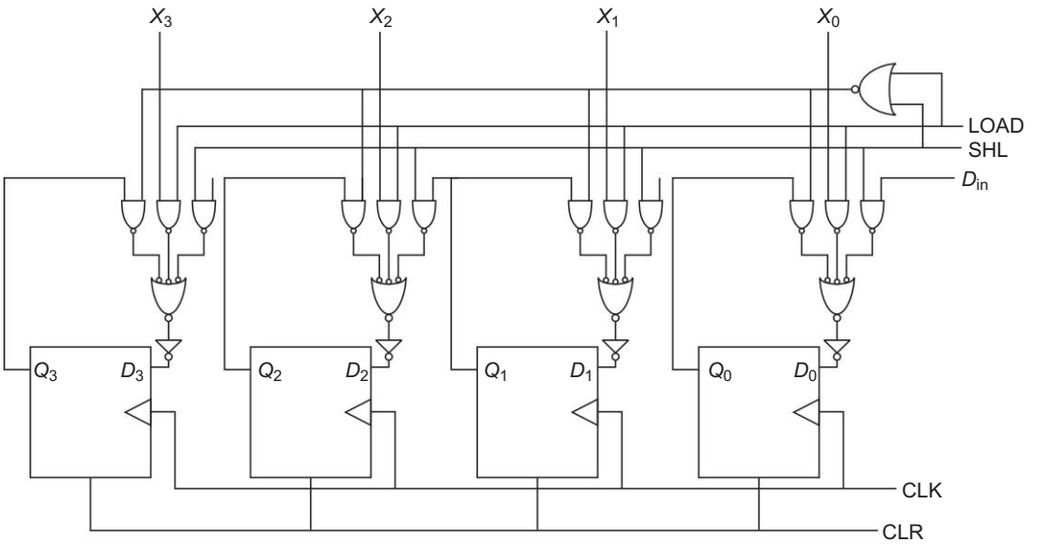


Fig. P9.2

9. 15 Write the VHDL codes of a 16-bit downcounter with synchronous loads and asynchronous reset. The outputs are three-state outputs, controlled by two separate signals—one for the lower 8-bits and one for the upper 8-bit **inputs:**

```
clk, reset, load, data [16:1] and upper_ enable, and lower_enable
output: count[15:0].
```

9.16 Design a state machine that detects—starting with the leftmost bit—the sequence "1110101101". Then, perform the following:

(a) Draw the state-flow diagram.

(b) Code the design as a two-process FSM using an enumeration type to define the states.

(c) Compile and synthesize the design targeting Spartan-III.

9.17 Design and compile 3:1 clock free four-bit word multiplexer by using VHDL codes, targeting Spartan-III.

9.18 Write VHDL codes of DRAM controller (shown in Fig. P9.3), considering that it refreshes by 10 ns. Modify the codes to refresh the DRAM every 4 ns.

9.19 Convert the following expression to a canonical sum of the products:

```
X<= (a or b) xor (c and d)
```

9.20 Find the errors in the following VHDL codes:

```
Use ieee.std_logic_1164.all;
Use work.numeric_std.all;
Use work.mnemerics_std.all;
```

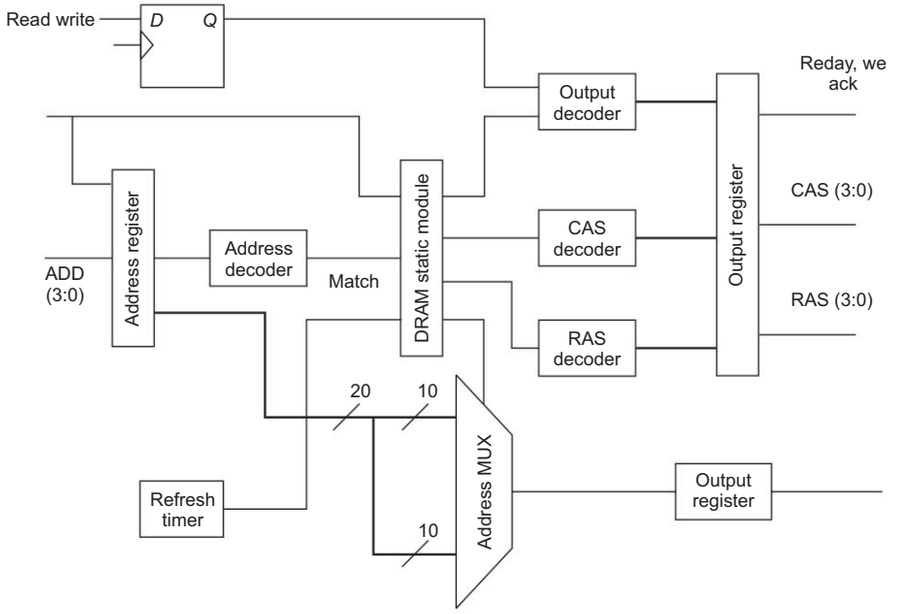


Fig. P9.3

```
        entity out_mux is port C
        ad, f: in unsigned(3 down to 0);
        dest_ctl: in std_logic vector (2 down to 0);
    oe: in std_logic;
        y: buffer unsigned(3 down to 0);
  end ;
  architecture out_mux is
  signal y_int: unsigned(3 down to 0);            --output before
                                                  tri-state buffer
  begin
     y_int <= ad when dest_ctl = rama else f;
    y <= y_int when oe = 'o' else "zzzz";
end out_mux;
```

9.21 Create VHDL codes of a 16-bit pipelined carry look-ahead adder using 4-bit group adders.

9.22 Create VHDL codes of a 16-bit pipelined carry select adder using 4-bit group adders.

9.23 Create VHDL codes of test fixture for a 16-bit pipelined carry look-ahead adder.

10 Top-level System Design: CPU

A simple example of top-level system design is the design of Central Processing Unit (CPU). In this chapter, we will discuss CPU design starting from VHDL representation to verification of its functionality synthesis with VHDL programming. In order to do that, we will use a 16-bit microprocessor.

10.1 CPU : 16-BIT MICROPROCESSOR

A block diagram of a 16-bit microprocessor is shown in Fig. 10.1. The processor contains a number of basic pieces. There is a register array of eight 16-bit registers, an arithmetic logic unit (ALU), a shifter, a program counter, an instruction register, a comparator, an address register, and a control unit. All of these units communicate through a common 16-bit tri-state data bus.

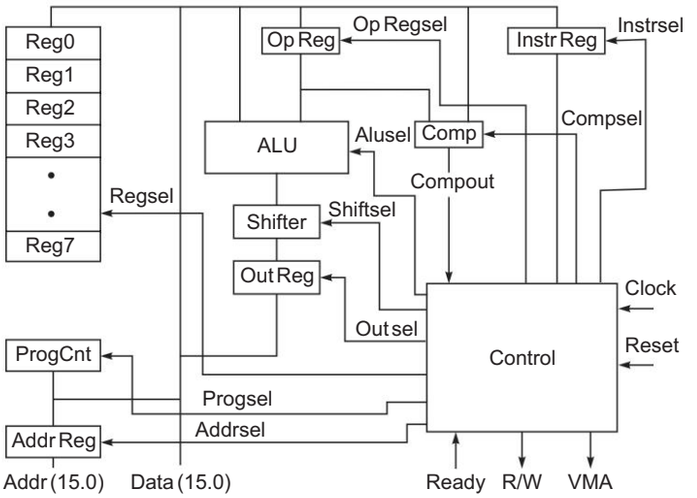


Fig. 10.1 Block diagram of 16-bit microprocessor CPU

The top-level design consists of the processor block and a memory block communicating through a bidirectional databus, an address bus, and a few control lines. The processor fetches instructions from

the external memory and executes these instructions to run a program. These instructions are stored in the instruction register and decoded by the control unit. The control unit causes the appropriate signal interactions to make the processor unit execute the instruction. If the instruction is an add of two registers, the control unit would cause the first register value to be written to register `OpReg` for temporary storage. The second register value would then be placed on the data bus. The ALU would be placed in add mode and the result would be stored in register `OutReg`. Register `OutReg` would store the resulting value until it is copied to the final destination. When executing an instruction, a number of steps take place. The program counter holds the address in memory of the current instruction. After an instruction has finished execution, the program counter is advanced to where the next instruction is located. If the processor is executing a linear stream of instructions, this is the next instruction. If a branch was taken, the program counter is loaded with the next instruction location directly. The control unit copies the program-counter value to the address register, which outputs the new address on the address bus. At the same time, the control unit sets the R/W (read-write signal) to a '0' value for a read operation and sets signal valid memory address (VMA) to a '1', signaling the memory that the address is now valid. The memory decodes the address and places the memory data on the data bus. When the data has been placed on the data bus, the memory has set the READY signal to a '1' value indicating that the memory data is ready for consumption. The control unit causes the memory data to be written into the instruction register. The control unit now has access to the instruction and decodes the instruction. The decoded instruction executes, and the process starts over again.

10.2 □ INSTRUCTIONS

Instructions can be divided into a number of different types as follows:

- **Load** These instructions load register values from other registers, memory locations, or with immediate values given in the instruction.
- **Store** These instructions store register values to memory locations.
- **Branch** These instructions cause the processor to go to another location in the instruction stream. Some branch instructions test values before branching; others branch without testing.
- **ALU** These instruction perform arithmetic and logical operations such as ADD, SUBTRACT, OR, AND, and NOT.
- **Shift** These instructions use the shift unit to perform shift operations on the data passed to it.

Sample Instruction Representation

Instructions share common attributes. Sample instructions are shown in Table 10.1. All instructions contain the opcode in the five most significant bits of the instruction. Single-word instructions also contain two 3-bit register fields in the lowest 6 bits of the instruction. Some instructions, such as INC (Increment), only use one of the fields, but other instructions, such as MOV (Move), use both register fields to specify the `From` register and the `To` register. In double-word instructions, the first word contains the opcode and destination register address, and the second word contains the immediate instruction location or data value to be loaded. For instance, a `LoadI` (Load Immediate) instruction would look like this:

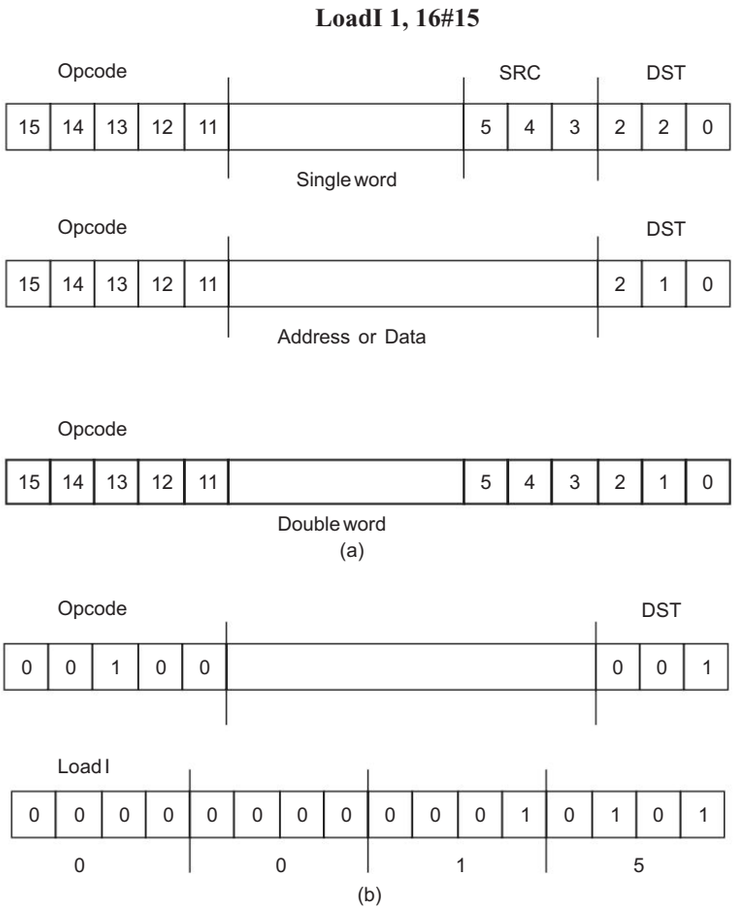


Fig. 10.2 Instructions and their opcodes

This instruction loads the hex value 15 into the register 1. The instruction opcodes of this is shown in Fig. 10.2. When the control unit decodes the opcode of the first word, it determines that the instruction is two words long and loads the second word to complete the instruction. The instructions implemented in the processor and their opcodes are listed in Table 10.1. By using this table, all of the possible instructions may not be implemented in this processor example. Typical commercial processors are much more complicated and have pipelined instruction streams for faster execution. To reduce complexity, this example is not pipelined.

Table 10.1 Instruction sets of a 16-bit microprocessor and its opcodes

Opcode	Instruction	Note
00000	NOP	No operation
00001	Load	Load register
00010	STORE	Store register

Contd..

00011	MOVE	MOVE value to register
00100	LOADI	LOAD register with immediate value
00101	BRANCHI	BRANCH immediate address
00110	BRANCHGTI	Branch greater than to immediate address
00111	INC	Increment
01000	DEC	Decrement
01001	AND	And two register
01010	OR	Or two register
01011	XOR	Xor two register
01100	NOT	NOT register
01101	ADD	Add two registers
01110	SUB	Substract two regeister
01111	ZERO	Zero register
10000	BRANC?HLTI	Branch less than to immediate value
10001	BRANCHLT	Branch less than
10010	BRANCHNEQ	Branch not equal
!0011	BRANCHNEQI	Branch not equal to immediate address
10100	BRANCHGT	Branch grater than
10101	BRANCH	Branch all the time
10110	BRANCHEQ	Branch if equal
10111	BRANCHEQI	Branch if equal to immediate address
11000	BRANCHLTEQ	Branch if less or equal to immediate address
11001	BRANCHLTE	Branch if less or equal
11010	SHL	Shift left
11011	SHR	Shift right
11100	ROTR	Rotate right
11101	ROT L	Rotate left

Before VHDL description for each of the CPU components, a top-level package `cpu_lib.vhd` is needed to describe the signal types that are used to communicate between the CPU components. Following is this package:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
package cpu_lib is
   type t_shift is (shftpass, shl, shr, rotl, rotr);
   subtype t_alu is unsigned(3 downto 0);
   constant alupass : unsigned(3 downto 0) := "0000";
   constant andOp : unsigned(3 downto 0) := "0001";
constant orOp : unsigned(3 downto 0) := "0010";
constant notOp : unsigned(3 downto 0) := "0011";
constant xorOp : unsigned(3 downto 0) := "0100";
```

```
constant plus : unsigned(3 downto 0) := "0101";
constant alusub : unsigned(3 downto 0) := "0110";
constant inc : unsigned(3 downto 0) := "0111";
constant dec : unsigned(3 downto 0) := "1000";
constant zero : unsigned(3 downto 0) := "1001";
type t_comp is (eq, neq, gt, gte, lt, lte);
subtype t_reg is std_logic_vector(2 downto 0);
type state is (reset1, reset2, reset3, reset4, reset5,
               reset6, execute, nop, load, store, move,
               load2, load3, load4, store2, store3,
               store4, move2, move3, move4,incPc, incPc2,
               incPc3, incPc4, incPc5, incPc6, loadPc,
               loadPc2,loadPc3, loadPc4, bgtI2, bgtI3,
               bgtI4, bgtI5, bgtI6, bgtI7,bgtI8, bgtI9,
               bgtI10, braI2, braI3, braI4, braI5, braI6,
               loadI2,loadI3, loadI4, loadI5, loadI6,
               inc2, inc3, inc4);
               subtype bit16 is std_logic_vector(15 downto 0);
end cpu_lib;
```

This package describes a number of types that are used to specify the ALU functionality, the shifter operation, and the states needed for the control of the CPU. It is described by the file `top.vhd` as given below:

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.cpu_lib.all;
entity top is
end top;
architecture behave of top is
   component mem
   port (addr : in bit16;
      sel, rw : in std_logic;
      ready : out std_logic;
      data : inout bit16);
   end component;
   component cpu
   port(clock, reset, ready : in std_logic;
     addr : out bit16;
     rw, vma : out std_logic;
     data : inout bit16);
end component;
signal addr, data : bit16;
```

```
   signal vma, rw, ready : std_logic;
   signal clock, reset : std_logic := '0';
begin
   clock <= not clock after 50 ns;
   reset <= '1', '0' after 100 ns;
   m1 : mem port map (addr, vma, rw, ready, data);
u1 : cpu port map (clock, reset, ready, addr, rw, vma, data);
   end behave of top;
```

This model also has components `cpu` and `mem` and specifies the necessary signals to connect the components, as shown in Fig. 10.3. Component `mem` is a memory device which contains the instructions/data for the CPU to execute. Component `cpu` is an RTL implementation of the CPU device that is simulated for correctness and synthesized to implement the design. The memory is described in the file `mem.vhd` as shown below:

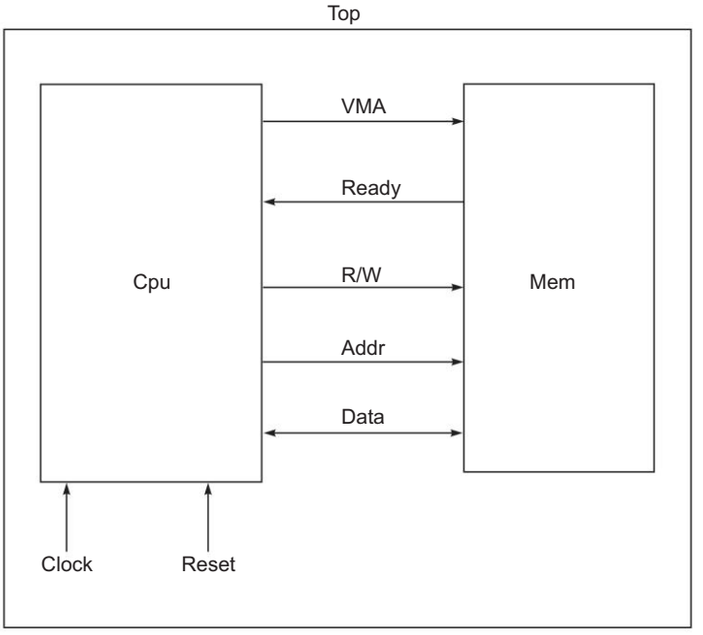


Fig. 10.3 CPU design model

```
library IEEE;
    use IEEE.std_logic_1164.all;
    use IEEE.std_logic_arith.all;
    use IEEE.std_logic_unsigned.all;
    use work.cpu_lib.all;
entity mem is
port (addr : in bit16;
```

```
      sel, rw : in std_logic;
      ready : out std_logic;
      data : inout bit16);
end mem;
architecture behave of mem is
begin
memproc: process(addr, sel, rw)
type t_mem is array(0 to 63) of bit16;
variable mem_data : t_mem :=
("0010000000000001",  --- 0  loadI 1, # -- load source address
"0000000000010000",  --- 1  10
"0010000000000010", --- 2  loadI 2, #   -- load destination address
"0000000000110000", --- 3  30
"0010000000000110", --- 4  loadI 6, # -- load data end address
"0000000000101111", --- 5  2F
"0000100000001011",  --- 6  load 1, 3 -- load reg3 with source element
"0001000000011010", --- 7  store 3, 2  -- store reg3 at destination
"0011000000001110", --- 8  bgtI 1, 6, # -- compare to
                                        see if at end of data
"0000000000000000", --- 9  00        -- if so just start over
"0011100000000001", --- A  inc 1     -- move source address to next
"0011100000000010", --- B  inc 2   -- move destination address
                                      to next
"0010100000001111", --- C  braI #    -- go to the next element
                                      to copy
"0000000000000110",  --- D  06
"0000000000000000",  --- E
"0000000000000000",  --- F
"0000000000000001",  --- 10       --- Start of source array
"0000000000000010",  --- 11
"0000000000000011",  --- 12
"0000000000000100",  --- 13
"0000000000000101",  --- 14
"0000000000000110",  --- 15
"0000000000000111",  --- 16
"0000000000001000",  --- 17
"0000000000001001",  --- 18
"0000000000001010",  --- 19
"0000000000001011",  --- 1A
"0000000000001100",  --- 1B
"0000000000001101",  --- 1C
"0000000000001110",  --- 1D
    "0000000000001111",  --- 1E
```

```
"0000000000010000",  --- 1F
"0000000000000000",  --- 20
"0000000000000000",  --- 21
"0000000000000000",  --- 22
"0000000000000000",  --- 23
"0000000000000000",  --- 24
"0000000000000000",  --- 25
"0000000000000000",  --- 26
"0000000000000000",  --- 27
"0000000000000000",  --- 28
"0000000000000000",  --- 29
"0000000000000000",  --- 2A
"0000000000000000",  --- 2B
"0000000000000000",  --- 2C
"0000000000000000",  --- 2D
"0000000000000000",  --- 2E
"0000000000000000",  --- 2F
"0000000000000000",  --- 30    --- start of destination array
"0000000000000000",  --- 31
"0000000000000000",  --- 32
"0000000000000000",  --- 33
"0000000000000000",  --- 34
"0000000000000000",  --- 35
"0000000000000000",  --- 36
"0000000000000000",  --- 37
"0000000000000000",  --- 38
"0000000000000000",  --- 39
"0000000000000000",  --- 3A
"0000000000000000",  --- 3B
"0000000000000000",  --- 3C
"0000000000000000",  --- 3D
"0000000000000000",  --- 3E
"0000000000000000"); --- 3F
begin
data <= "ZZZZZZZZZZZZZZZZ";
ready <= '0';
if sel = '1' then
   if rw = '0' then
   data <= mem_data(CONV_INTEGER(addr(15 downto 0)))
after 1 ns;
ready <= '1';
elsif rw = '1' then
```

```
mem_data(CONV_INTEGER(addr(15 downto 0))) := data;
end if;
else
data <= "ZZZZZZZZZZZZZZZ" after 1 ns;
end if;
end process;
end behave;
```

Entity `mem` is a large array with a simple bus interface to allow reading and writing to the memory. A memory location is selected for read by placing the appropriate address of the location on signal addr, setting. Input rw (read write) to a '0' and putting the value '1' on signal sel (select). The value of the memory location appears on signal data, and signal ready is set to a '1' value signaling that the memory information is available.

To write a location in the memory, the address is placed on signal addr, setting signal rw to a '1' value, setting signal sel to a '1' value, and putting the data to be written on input data. The memory is divided into two separate sections. The first section is the instruction area, and the second is the data area. The instruction area contains the instructions to be executed, and the second section contains the data area for the instructions to manipulate. The CPU instructions start at location 00 and end at location 0D. The data area starts at location 10 and ends at location 3F, the end of the array. The instructions stored in the memory device are a simple algorithm for moving a block of data from one location to another. This type of program could also be considered a block-copy operation.

10.3 □ BLOCK-COPY OPERATION

A block-copy operation is shown in Fig. 10.4. The copy operation starts when the CPU gets a reset signal. A reset signal causes the CPU to reset its internal state and start processing instructions at location 00 of the memory. The first set of instructions are used to set up the appropriate CPU registers so that the block-copy operation can be done. Register 1 contains the starting address, or the address of the first element of the memory block to be copied. Register 2 contains the starting address for the destination of the memory block. Register 6 contains the ending address of the memory block to be copied. The first instruction at location 00 loads register 1 with the starting address of the memory block to be copied. The actual address is contained in mem location 01. The value is hexadecimal 10 or 16 decimal. The block-copy program starts the copy operation from location 10. The first instruction is a double-word instruction. The first word specifies the instruction opcode and the registers to be used in the instruction. The second word contains the absolute address to be used in the operation. The next instruction is at the memory location 02. The first instruction advanced the program counter past location 01, which contained the starting address. This instruction loads the register 2 with the destination address for the block copy. The destination is at 30 hex or 48 decimal. These load instructions are load-immediate instructions, which load the next memory location into the register specified in the instruction. The final set-up instruction is at mem location 04. This instruction loads the last address of the memory block to be copied. This register signals the block-copy routine when to stop the copy operation. After this instruction has been executed, all of the registers have been set up for the block-copy operation, and the copy loop can start. Instruction 6 is the start of a loop of instructions that perform the copy operation. Instruction 6 copies the contents of the memory location whose address is

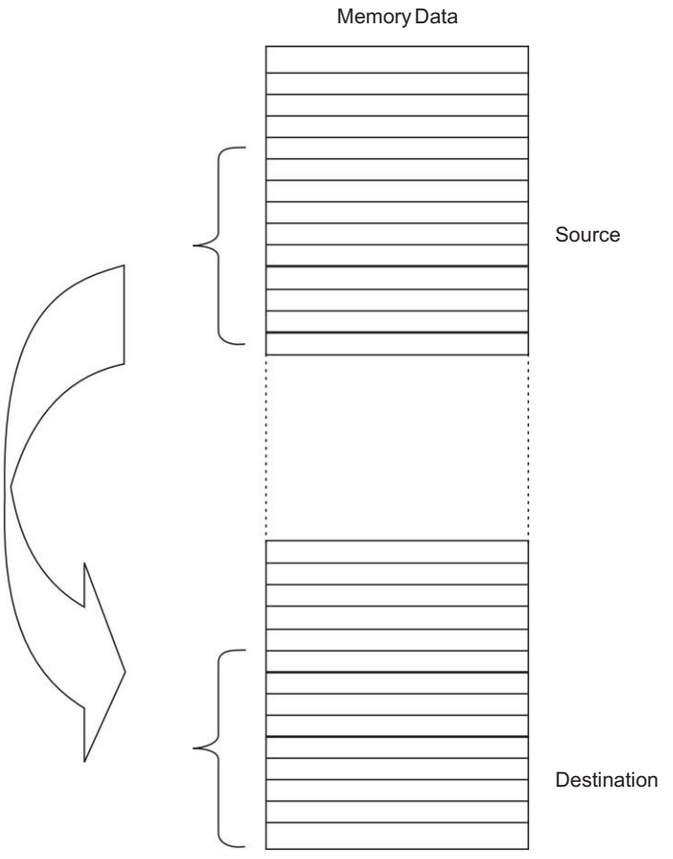


Fig. 10.4 Block-copy operation

contained in register 1 to register 3. Instruction 7 copies the data in register 3 to the memory location specified in the register 2. After these two instructions have executed the first time, the first memory element of the block will have been copied. After the copy operation, the program needs to check if it is done. Instruction 8 compares the address in the register 1 versus the end address in the register 6 to see if the copy operation has completed the last element. If so, the program should exit because the copy operation has completed. However, in this simple example, there is no other program to execute, so this program branches to the instruction 00 and starts the process over again. If the copy operation is not completed, the CPU executes the instruction at 0A, which increments register 1. This instruction increments register 1 so that it points to the next element to be copied. Instruction 0B increments the register 2, which moves the destination address to the next location. Finally, instruction 0C branches back to instruction 06 and continues the next copy operation.

So far, we have discussed different instructions used in a 16-bit microprocessor and using these instructions, the library file `cpu.lib.vhd` is made. It is also essential to describe CPU with Register Transfer Level (RTL). The CPU is described by a number of lower-level data abstraction and

components that are instantiated to form the CPU design. At the top, the CPU is an architecture that instantiates all of the lower-level components to form reference and value types as arguments. The CPU block diagram is shown in Fig. 10.1. Following is an implementation of this block diagram, shown by the file `cpu.vhd`:

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.cpu_lib.all;

entity cpu is
     port(clock, reset, ready : in std_logic;
     addr : out bit16;
     rw, vma : out std_logic;
     data : inout bit16);
end cpu;

architecture rtl of cpu is
     component regarray
     port( data : in bit16;
          sel : in t_reg;
          en : in std_logic;
          clk : in std_logic;
          q : out bit16);
     end component;
               component reg
port( a : in bit16;
          clk : in std_logic;
          q : out bit16);
end component;

component trireg
     port( a : in bit16;
          en : in std_logic;
          clk : in std_logic;
          q : out bit16);
end component;
component control
port( clock : in std_logic;
          reset : in std_logic;
          instrReg : in bit16;
          compout : in std_logic;
          ready : in std_logic;
          progCntrWr : out std_logic;
```

```
          progCntrRd : out std_logic;
          addrRegWr : out std_logic;
          outRegWr : out std_logic;
          outRegRd : out std_logic;
          shiftSel : out t_shift;
          aluSel : out t_alu;
          compSel : out t_comp;
          opRegRd : out std_logic;
          opRegWr : out std_logic;
          instrWr : out std_logic;
          regSel : out t_reg;
          regRd : out std_logic;
          regWr : out std_logic;
          rw : out std_logic;
          vma : out std_logic
          );
end component;

component alu
   port(a, b : in bit16;
          sel : in t_alu;
          c : out bit16);
end component;

component shift
port ( a : in bit16;
          sel : in t_shift;
          y : out bit16);
end component;
  component comp
  port( a, b : in bit16;
  sel : in t_comp;
  compout : out std_logic);
  end component;
  signal  opdata, aluout, shiftout, instrregOut : bit16;
  signal regsel : t_reg;
  signal regRd, regWr, opregRd, opregWr, outregRd, outregWr,
          addrregWr, instrregWr, progcntrRd, progcntrWr,
          compout : std_logic;
  signal alusel : t_alu;
  signal shiftsel : t_shift;
  signal compsel : t_comp;
begin
```

```
ra1 : regarray port map(data, regsel, regRd, regWr, data);
opreg: trireg port map (data, opregRd, opregWr, opdata);
alu1: alu port map (data, opdata, alusel, aluout);
shift1: shift port map (aluout, shiftsel, shiftout);
outreg: trireg port map (shiftout, outregRd, outregWr,data);
addrreg: reg port map (data, addrregWr, addr);
progcntr: trireg port map (data, progcntrRd, progcntrWr,data);
comp1: comp port map (opdata, data, compsel, compout);
instr1: reg port map (data, instrregWr, instrregOut);
con1: control port map (clock, reset, instrregOut, com
  pout, ready, progcntrWr, progcntrRd, addrregWr, out
regWr, outregRd, shiftsel, alusel, compsel, opre
gRd, opregWr, instrregWr, regsel, regRd, regWr, rw,
vma);
end rtl;
```

Architecture `rtl` of entity `cpu` is a structural implementation of the block diagram. Architecture `rtl` contains the component declarations of all the components used to build the design, the signals used to connect the components, and the component instantiations to create the functionality. After the component and signal declarations are the component instantiation statements that instance the components and connect the appropriate signals. In the next few sections, each of the VHDL component descriptions is described in more detail.

10.4 □ ALU

The first component in the CPU is described as ALU. This component entity performs a number of arithmetic or logical operations on one or more input buses. A symbol for the ALU is shown in Fig. 10.5. Inputs `a` and `b` are the two input buseres upon which the ALU operations (Table 10.2) are performed. Output bus `c` returns the result of the ALU operation. Input `sel` determines which operation is performed as specified by Fig. 10.5. As we can see, the ALU can perform a number of arithmetic operations, such as add and subtract, and some logical operations such as AND, OR, and XOR. Following is a VHDL description of the ALU component entity:

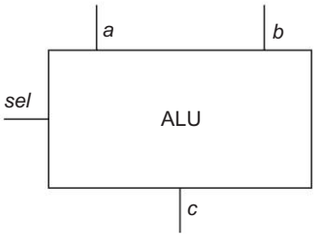


Fig. 10.5 ALU with inputs and outputs

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.cpu_lib.all;

entity alu is
  port( a, b : in bit16;
          sel : in t_alu;
          c : out bit16);
end alu;
architecture rtl of alu is
begin
   aluproc: process(a, b, sel)
   begin
   case sel is
   when alupass =>
   c <= a after 1 ns;

   when andOp =>
   c <= a and b after 1 ns;

when orOp =>
   c <= a or b after 1 ns;

when xorOp =>
   c <= a xor b after 1 ns;
when notOp =>
   c <= not a after 1 ns;

when plus =>
   c <= a + b after 1 ns;

when alusub =>
   c <= a - b after 1 ns;
   when inc =>
   c <= a +  "0000000000000001" after 1 ns;

   when dec =>
c <= a - "0000000000000001" after 1 ns;

   when zero =>
c <= "0000000000000000" after 1 ns;
```

```
   when others =>
c <= "0000000000000000" after 1 ns;

   end case;
   end process;
end rtl;
```

Table 10.2 ALU operations

Sel Input	Operation
0000	C = A
0001	C = A AND B
0010	C = A OR B
0011	C = NOT A
0100	C = A XOR B
0101	C = A + B
0110	C = A – B
0111	C = A + 1
1000	C = A – 1
1001	C = 0

The architecture uses a large `case` statement on input `sel` to determine which of the arithmetic or logical operations to perform. The possible values of signal `sel` are determined by type `t_alu` described in the package `cpu_lib` in the file `cpulib.vhd`. After the new value for output `c` is calculated, all of the resulting values are assigned with a 1-nanosecond time delay to eliminate delta-delay problems during RTL simulation.

10.5 □ COMPARATOR

The comparator entity is an important component in CPU as shown in Fig. 10.1. This entity compares two values and returns either a **'1'** or **'0'** depending on the type of comparison requested and the values being compared. A symbol showing the ports of the comparator is shown in Fig. 10.6. The comparison

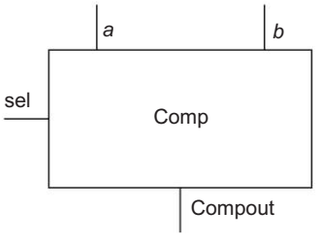


Fig. 10.6 Comp interface with inputs and outputs

type is determined by the value on input port `sel`. For instance, to compare if inputs `a` and `b` are equal, apply the value `eq` to port `sel`. If ports `a` and `b` have the same value, port `compout` returns **'1'**. If the values are not equal, **'0'** is returned. The types of comparisons allowed are described by type `t_comp` in the package `cpu_lib` in the file `cpulib.vhd` described earlier. The comparison types and values are given in Table 10.3 All operations work on two input values and return a single-bit result. This bit is used to control the flow of operation within the processor while executing instructions. Following is a VHDL description of the `comp` entity:

```
library IEEE;
        use IEEE.std_logic_1164.all;
        use IEEE.std_logic_arith.all;
        use work.cpu_lib.all;
            entity comp is
        port( a, b : in bit16;
    sel : in t_comp;
    compout : out std_logic);
              end comp;
    architecture rtl of comp is
                  begin
    compproc: process(a, b, sel)
                  begin
                 case sel is
               when eq =>
              if a = b then
    compout <= '1' after 1 ns;
                 else
    compout <= '0' after 1 ns;
             end if;
             when neq =>
               if a /= b then
    compout <= '1' after 1 ns;
                   else
                  compout <= '0' after 1 ns;
               end if;
                    when gt =>
               if a > b then
    compout <= '1' after 1 ns;
              else
    compout <= '0' after 1 ns;
              end if;
                   when gte =>
               if a >= b then
```

```
                compout <= '1' after 1 ns;
                           else
                compout <= '0' after 1 ns;
                           end if;
                               when lt =>
                          if a < b then
                compout <= '1' after 1 ns;
                              else
                compout <= '0' after 1 ns;
                            end if;
                               when lte =>
                           if a <= b then
                compout <= '1' after 1 ns;
                              else
                compout <= '0' after 1 ns;
                           end if;
              end case;
              end process;
              end rtl;
```

Table 10.3 Comp operations

Sel input value	Comparison
EQ	Compout = 1 when a equals b
NEQ	Compout = 1 when a is not equals to b
GT	Compout = 1 when a is greater then b
GTE	Compout = 1 when a is greater than or equal to b
LT	Compout = 1 when a is less than b
LTE	Compout = 1 when a is less than or equal to b

The comparator consists of a large `case` statement where each branch of the `case` statement contains an `IF`. If the condition tested is true, a **'1'** value is assigned; otherwise, a **'0'** is assigned. Again, each assignment occurs after 1 nanosecond to remove delta delay problems.

10.6 □ CONTROL

The **control** entity provides the necessary signal interactions to make the data flow properly through the CPU and perform the expected functions. Architecture `rtl` has a state machine that causes all appropriate signal values to update based on the current state and input signals and produce a next state for the state machine. A symbol for the control block is shown in Fig. 10.7. The control symbol has only

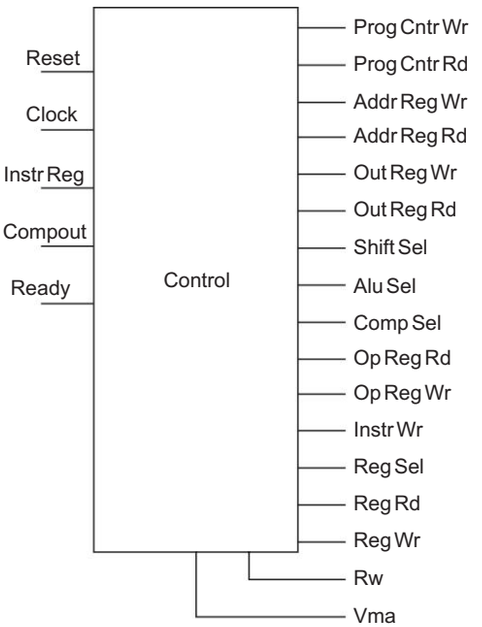


Fig. 10.7 Control symbols with clock

a few inputs, but a lot of outputs. The control block provides all of the control signals to regulate data traffic for the CPU. Following is the VHDL description for the CPU:

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.cpu_lib.all;

entity control is
port( clock : in std_logic;
          reset : in std_logic;
          instrReg : in bit16;
          compout : in std_logic;
          ready : in std_logic;
          progCntrWr : out std_logic;
          progCntrRd : out std_logic;
          addrRegWr : out std_logic;
          addrRegRd : out std_logic;
          outRegWr : out std_logic;
          outRegRd : out std_logic;
          shiftSel : out t_shift;
          aluSel : out t_alu;
          compSel : out t_comp;
```

```
            opRegRd : out std_logic;
            opRegWr : out std_logic;
            instrWr : out std_logic;
            regSel : out t_reg;
            regRd : out std_logic;
            regWr : out std_logic;
            rw : out std_logic;
            vma : out std_logic
    );
end control;
architecture rtl of control is
signal current_state, next_state : state;
begin
nxtstateproc: process( current_state, instrReg, compout, ready)
begin
progCntrWr <= '0';
progCntrRd <= '0';
addrRegWr <= '0';
outRegWr <= '0';
outRegRd <= '0';
shiftSel <= shftpass;
aluSel <= alupass;
compSel <= eq;
opRegRd <= '0';
opRegWr <= '0';
instrWr <= '0';
regSel <= "000";
regRd <= '0';
regWr <= '0';
rw <= '0';
vma <= '0';
case current_state is
when reset1 =>
aluSel <= zero after 1 ns;
shiftSel <= shftpass;
next_state <= reset2;

when reset2 =>
aluSel <= zero;
shiftSel <= shftpass;
outRegWr <= '1';
next_state <= reset3;
```

```
when reset3 =>
outRegRd <= '1';
next_state <= reset4;

when reset4 =>
outRegRd <= '1';
progCntrWr <= '1';
addrRegWr <= '1';
next_state <= reset5;

when reset5 =>
vma <= '1';
rw <= '0';
next_state <= reset6;

when reset6 =>
vma <= '1';
rw <= '0';
if ready = '1' then
instrWr <= '1';
next_state <= execute;
else
next_state <= reset6;
end if;

when execute =>
case instrReg(15 downto 11) is
when "00000" =>                  --- nop
next_state <= incPc;

when "00001" =>                --- load
regSel <= instrReg(5 downto 3);
regRd <= '1';
next_state <= load2;

when "00010" =>                --- store
regSel <= instrReg(2 downto 0);
regRd <= '1';
next_state <= store2;
                when "00011" =>            ----- move
regSel <= instrReg(5 downto 3);
regRd <= '1';
```

```
aluSel <= alupass;
shiftSel <= shftpass;
next_state <= move2;

when "00100" =>          ---- loadI
progcntrRd <= '1';
alusel <= inc;
shiftsel <= shftpass;
next_state <= loadI2;

when "00101" =>          ---- BranchImm
progcntrRd <= '1';
alusel <= inc;
shiftsel <= shftpass;
next_state <= braI2;

when "00110" =>          ---- BranchGTImm
regSel <= instrReg(5 downto 3);
regRd <= '1';
next_state <= bgtI2;
when "00111" =>         ------- inc
regSel <= instrReg(2 downto 0);
regRd <= '1';
alusel <= inc;
shiftsel <= shftpass;
next_state <= inc2;

when others =>
next_state <= incPc;

end case;

when load2 =>
regSel <= instrReg(5 downto 3);
regRd <= '1';
addrregWr <= '1';
next_state <= load3;

when load3 =>
vma <= '1';
rw <= '0';
```

```
next_state <= load4;
when load4 =>
vma <= '1';
rw <= '0';
regSel <= instrReg(2 downto 0);
regWr <= '1';
                                    next_state <= incPc;

when store2 =>
regSel <= instrReg(2 downto 0);
regRd <= '1';
addrregWr <= '1';
next_state <= store3;

when store3 =>
regSel <= instrReg(5 downto 3);
regRd <= '1';
next_state <= store4;

when store4 =>
regSel <= instrReg(5 downto 3);
regRd <= '1';
vma <= '1';
rw <= '1';
next_state <= incPc;

when move2 =>
regSel <= instrReg(5 downto 3);
regRd <= '1';
aluSel <= alupass;
shiftsel <= shftpass;
outRegWr <= '1';
next_state <= move3;

when move3 =>
outRegRd <= '1';
next_state <= move4;

when move4 =>
outRegRd <= '1';
regSel <= instrReg(2 downto 0);
regWr <= '1';
next_state <= incPc;
```

```
when loadI2 =>
progcntrRd <= '1';
alusel <= inc;
shiftsel <= shftpass;
outregWr <= '1';
next_state <= loadI3;

when loadI3 =>
outregRd <= '1';
next_state <= loadI4;

when loadI4 =>
outregRd <= '1';
progcntrWr <= '1';
addrregWr <= '1';
      next_state <= loadI5;
      when loadI5 =>
vma <= '1';
 rw <= '0';
next_state <= loadI6;
when loadI6 =>
       vma <= '1';
        rw <= '0';
   if ready = '1' then
regSel <= instrReg(2 downto 0);
regWr <= '1';
next_state <= incPc;
else
next_state <= loadI6;
end if;

when braI2 =>
progcntrRd <= '1';
alusel <= inc;
shiftsel <= shftpass;
outregWr <= '1';
next_state <= braI3;
when braI3 =>
outregRd <= '1';
next_state <= braI4;

when braI4 =>
```

```
outregRd <= '1';
progcntrWr <= '1';
addrregWr <= '1';
next_state <= braI5;

when braI5 =>
vma <= '1';
rw <= '0';
next_state <= braI6;

when braI6 =>
vma <= '1';
rw <= '0';
if ready = '1' then
progcntrWr <= '1';
next_state <= loadPc;
else
next_state <= braI6;
end if;

when bgtI2 =>
regSel <= instrReg(5 downto 3);
regRd <= '1';
opRegWr <= '1';
next_state <= bgtI3;

else
if clock'event and clock = '1' then
current_state <= next_state after 1 ns;
end if;
end process;
end rtl;
```

Architecture `rtl` contains two processes. The first is a combinational process (not clocked) that examines the current state and all inputs and produces output control values and next state output. The second process is a sequential process (has a clock) that is used to store the current state and copy the next state to the current state. The next state transitions occur on rising edges of the clock input. The control block is a very large state machine that contains a number of states for each instruction. Executing all of the states for an instruction performs the necessary steps to complete the instruction. If the `reset` signal is high, the sequential process labeled `controlffproc` sets signal `current_state` to state value `reset1`. This is the first state of the reset sequence for the CPU. This state starts the process of getting the CPU ready to execute instructions. If the `reset` signal is not **'1'** and there

is a rising edge on the clock signal, then the `controlffproc` process copies the `next_state` signal generated by the combinational process to signal `current_state`. This is the method for the state machine to advance from one state to another. After the `reset` signal is set to a value other than **'1'**, the state machine is in state `reset1`. This state causes the `alu` entity to output the value **0**, the shift entity to pass the value with no modification, and the next state signal to be updated with the value `reset2`. This can be seen in the VHDL description for entity `control` in the case statement starting at the `when` clause for state `reset1`. At the next clock edge, the state machine advances to state `reset2`. State `reset2` leaves the control signals for the `alu` and `shift` entities as before, but also sets the `OutRegWr` signal to a **'1'**, causing the **0** value on the data bus to be written to register `OutReg`. The goal of the reset sequence is to set up the program counter to start reading instructions from memory. After state `reset2`, the state machine next goes to state `reset3` on the next clock edge. This state sets signal `OutRegRd` to a **'1'**, causing entity `OutReg` to output its value to the data bus. The state machine then advances to state `reset4`. During `reset4`, the value from `OutReg` is copied into register `ProgCntr` and also to register `AddrReg`. The state machine advances to state `reset5`, sets output signal `RW` (read write) to **'0'** (read mode), and signals VMA (Valid Memory Address) to a **'1'**. This causes memory entity `mem` to output the data at location **0** to the data bus. The state machine advances to state `reset6` and, depending on the value of the ready signal from the memory, either stays in `reset6` or writes the memory data value to register `InstrReg` and goes to state `execute`. At this point, the state machine has reset the state of the CPU to a known state and loaded the first instruction into register `InstrReg`. From this point forward, the state machine changes state depending on the instructions encountered. The `reset` of the description for the state machine contains the state transitions for the rest of the instructions that have been implemented. As mentioned previously, not all of the instructions have been implemented and are left as an exercise for the reader.

10.7 □ REG

The `reg` entity is used for the address register and the instruction register. These registers need to be able to capture the input data on a rising edge of the `clk` input and drive output `q` with the captured data. The value of input `a` is assigned to output `q` when a rising edge occurs on input `clk`. The assignment is delayed by 1 nanosecond to remove delta delay problems during simulation. A symbol for the `reg` entity is shown in Fig. 10.8. The `reg` symbol contains three ports. Port `a` is the data input port, `q` is the data output port, and port `clk` controls when the data is stored in the `reg` entity. Following is the VHDL description for the entity `reg`:

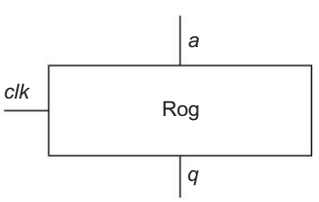


Fig. 10.8 Register with inputs and outputs

```
library IEEE;
      use IEEE.std_logic_1164.all;
      use work.cpu_lib.all;
    entity reg is
    port( a : in bit16;
clk : in std_logic;
q : out bit16);
    end reg;
      architecture rtl of reg is
     begin
        regproc: process
      begin
          wait until clk'event and clk = '1';
          q <= a after 1 ns;
     end process;
  end rtl;
```

Process `regproc` is triggered when a rising edge occurs on input `clk`. When the process is triggered, input `a` is copied to output `q`.

10.8 □ REGARRAY

In a CPU, the `regarray` entity is used to model the set of registers that are used to store intermediate values during instruction processing. These registers are read from and written to during the execution of instructions. The set of registers is modeled as a RAM of eight 16-bit words. The `regarray` component is shown in Fig. 10.9. To write a location in the `regarray`, we have to set input `sel` to the location to be written. Then, we have to set input data with the data to be written, and finally we have to put a rising edge on input `clk`. To read a location from `regarray`, we set input `sel` to the location to be read and set input `en` to a '1'. The data is output on port `q`. The register array is described as two separate processes as shown in the following:

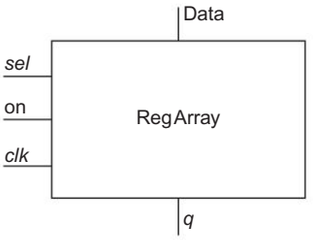


Fig. 10.9 Reg Array with inputs and outputs

```
library IEEE;
                  use IEEE.std_logic_1164.all;
                  use IEEE.std_logic_unsigned.all;
```

```
                    use work.cpu_lib.all;
                     entity regarray is
                   port( data : in bit16;
                       sel : in t_reg;
                       en : in std_logic;
                       clk : in std_logic;
                        q : out bit16);
                      end regarray;
                           architecture rtl of regarray is
type t_ram is array (0 to 7) of bit16;
signal temp_data : bit16;
begin
process(clk,sel)
variable ramdata : t_ram;
begin
if clk'event and clk = '1' then
ramdata(conv_integer(sel)) := data;
end if;
temp_data <= ramdata(conv_integer(sel)) after 1 ns;
end process;

process(en, temp_data)
begin
if en = '1' then
q <= temp_data after 1 ns;
else
q <= "ZZZZZZZZZZZZZZZ" after 1 ns;
end if;
end process;
end rtl;
```

The first process describes the part of the RAM that stores the data. The process has a local variable `ramdata` that is used to store the data written to the `regarray` entity. When the `clk` signal has a rising edge, the location selected by input `sel` is updated with the new value. This process also writes the location to a signal called `temp_data` to pass the value to the second process. The reason being that this model was written using VHDL, and variables cannot be shared between processes. Sharing variables between processes is legal but has other synthesis ramifications. The second process is used to read data from the `regarray`. Whenever input `sel` changes, the first process updates the value of `temp_data`. Signal `temp_data` is passed to the second process to pass the memory data.

The second process provides outputs for the value of `temp_data` if the `en` signal is '1'; otherwise, it puts out `Z` values. The `Z` values show that the `regarray` entity is not driving the output when the `en` input is unasserted. A synthesis tool reading this design states that the `regarray` entity can be implemented by a RAM device in the target technology and provides the proper mapping. For instance,

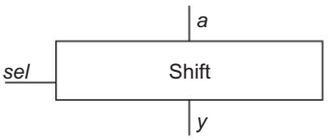


Fig. 10.10 Shift with inputs and outputs

if the design were to be mapped to an FPGA technology that included RAM in the architecture, the synthesis tool could map the `regarray` entity to an onboard RAM device. Using such an implementation instead of a set of flip-flops and gates, it is possible to get a smaller and faster implementation.

10.9 □ SHIFT

The `shift` entity is used to perform shifting and rotation operations within the CPU. The `shift` entity contains a 16-bit input bus, a 16-bit output bus, and a `sel` input that determines which shift operation to perform. This is shown in Fig. 10.10. The shift operations can be obtained by the `shift` entity as shown in Fig. 10.11. As seen in the figures, the `shift` entity can do a shift left, shift right, rotate left, and rotate right operation. Apart from the above operations shown in the figures, it provides a pass-through operation in which all input bits are passed through to the output unchanged. Following is an entity that performs these operations:

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.cpu_lib.all;

entity shift is
port ( a : in bit16;
     sel : in t_shift;
     y : out bit16);
end shift;

architecture rtl of shift is
begin
shftproc: process(a, sel)
begin
case sel is
when shftpass =>
y <= a after 1 ns;

when shl =>
y <= a(14 downto 0) & '0' after 1 ns;

when shr =>
y <= '0' & a(15 downto 1) after 1 ns;
```

```
when rotl =>
       y <= a(14 downto 0) & a(15) after 1 ns;
      when rotr =>
      y <= a(0) & a(15 downto 1) after 1 ns;
end case;
      end process;
      end rtl;
```

The shftpass mode allows the shifter to pass the input data to the output without any shift operations. This mode is quite common because all of the ALU operations flow through the shift entity, and very few instructions are actually performing a shift operation. The shl and shr selections perform shift left and shift right operations, respectively. The rotl and rotr selections perform rotate-left and rotate-right operations, respectively.

10.10 □ TRIREG

In CPU, the tri-state register component, trireg, is one of the important components and it is connected to the main data bus and can store information from the data bus as well as drive information to the data bus. In trireg, input a is the data input to the register, port q is the data output from the register and input clk is used to store a new value into the register. When a rising edge is applied to input clk, the data on input a is stored in the register. Input en is used to control output q. When en is a **'1'** value, the register state is driven to output q. When en is a **'0'**, output q is a high impedance value and not driving. This functionality is implemented by entity trireg shown in the following:

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.cpu_lib.all;

entity trireg is
port( a : in bit16;
          en : in std_logic;
          clk : in std_logic;
          q : out bit16);
end trireg;

architecture rtl of trireg is
signal val : bit16;
begin
triregdata: process
begin
wait until clk'event and clk = '1';
```

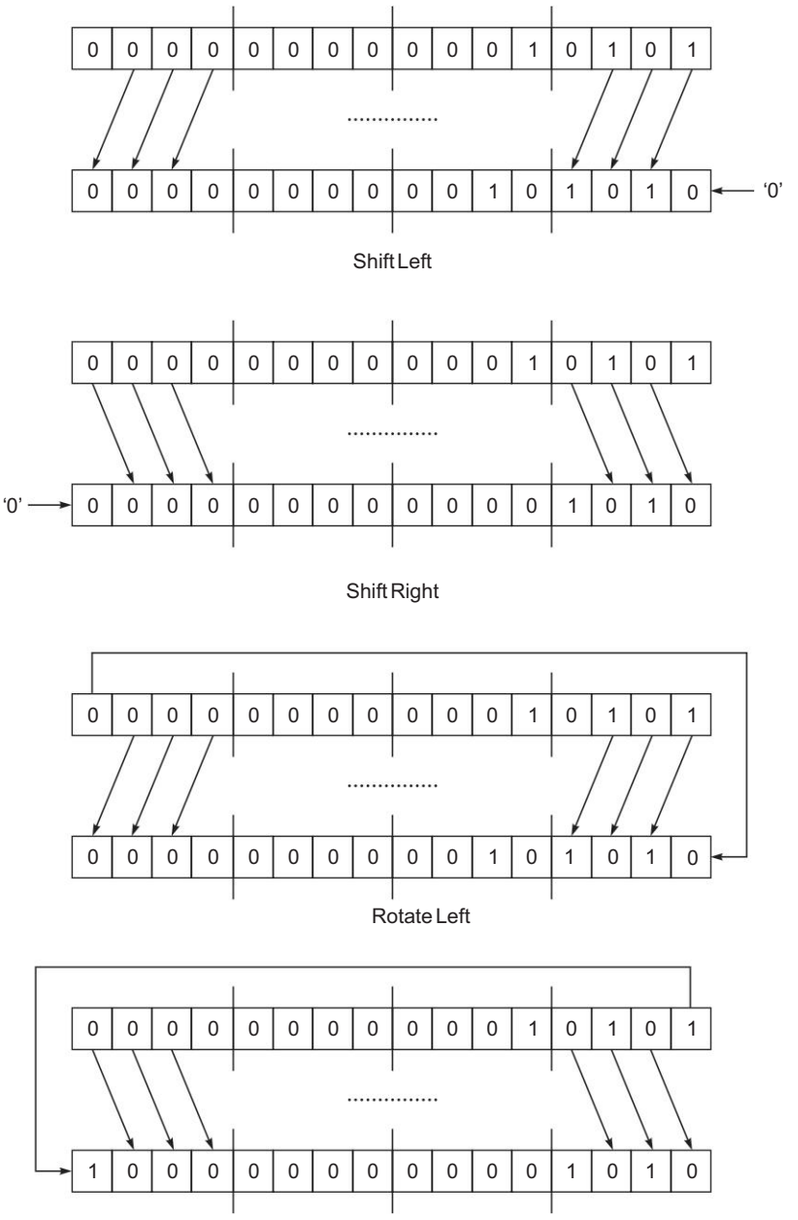


Fig. 10.11 Shift operations: Shift left, shift right, rotate left, rotate right

```
val <= a;
end process;

trireg3st: process(en, val)
begin
if en = '1' then
q <= val after 1 ns;
elsif en = '0' then
q <= "ZZZZZZZZZZZZZZZ" after 1 ns;
-- exemplar_translate_off
else
q <= "XXXXXXXXXXXXXXX" after 1 ns;
 -- exemplar_translate_on
    end if;
    end process;
    end rtl;
```

The functionality is represented by two processes that use a signal to communicate with the `regarray` entity. The first process does the controls when signal `val` is in the input `a`. Signal `val` is written only on the rising edge of input `clk`. The second process transfers the value of signal `val` only when input `en` is a '1' value; otherwise, a value of `'Z'` is output. When all of these entities are connected together correctly, the functionality of the CPU is obtained.

10.11 □ VERIFICATION OF RTL DESCRIPTION

A VHDL simulator is used to verify the functionality of the CPU VHDL RTL description. The VHDL RTL description of the CPU is simulated with a standard VHDL simulator to verify that the description is correct. A simulator needs two inputs—the description of the design and stimulus to drive the design. Sometimes designs are self-stimulating and do not need any external stimulus, but in most cases, VHDL designers use a VHDL testbench of one kind or another to drive the design being tested. The structure of the design looks like Fig. 10.12. The top-level design description instantiates two components—the first being the Design Under Test (DUT) and the second the stimulus driver.

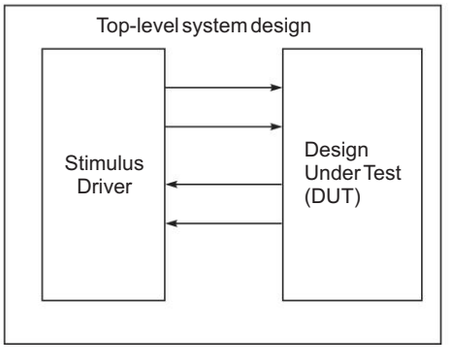


Fig. 10.12 Design structure for top-level system

A testbench is used to perform the analysis for the functionality of a design. When the designer makes a small change to fix an error, the change can be tested to make sure that it did not affect other parts of the design. Normally, the testbench provides the necessary input stimulus to the DUT and examines the output from the DUT. The testbench encapsulates the stimulus driver, known good results, and DUT, and contains internal signals to make the proper connections. The stimulus driver drives inputs into the DUT. The DUT responds to the input signals and produces output results. Finally, a compare function within the testbench compares the results from the DUT against those known good results and reports any discrepancies. There are different ways to make the testbench as given below:

- *Stimulus Only*—Contains only the stimulus driver and DUT; does not contain any results verification.
- *Full Testbench*—Contains stimulus driver, known good results, and results comparison.
- *Simulator Specific*—Testbench is written in a simulator-specific format.
- *Hybrid Testbench*—Combines techniques from more than one testbench style.
- *Fast Testbench*—Testbench written to get ultimate speed from simulation.

10.11.1 Stimulus-only Testbench

The stimulus only testbench contains the stimulus driver and DUT blocks of a testbench. The verification process is left to the designer. This type of testbench is useful at the beginning of a design project when no known good vectors exist, or for a quick check of an entity. Following is an example of stimulus only-testbench:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE std.textio.ALL;
USE ieee.std_logic_textio.all;
USE WORK.count_types.all;
ARCHITECTURE stimonly OF testbench IS


-- component declaration for counter

COMPONENT count
PORT (clk    : IN std_logic;
ld    : IN std_logic;
up_dwn : IN std_logic;
clk_en : IN std_logic;
din  : IN bit8;
qout : INOUT bit8);
END COMPONENT;
SIGNAL clk, ld, up_dwn, clk_en : std_logic;
SIGNAL qout, din : bit8;
```

```
BEGIN
-- instantiate the component
uut: count PORT MAP(clk => clk,
ld => ld,
up_dwn => up_dwn,
clk_en => clk_en,
din => din,
qout => qout);

--provide stimulus and check the result
test: PROCESS

VARIABLE tmpclk, tmpld, tmpup_dwn, tmpclk_en :
std_logic;
VARIABLE tmpdin : integer;

FILE vector_file : text IS IN "counter.txt";
VARIABLE l : line;
VARIABLE vector_time : time;
VARIABLE r : real;
VARIABLE good_number, good_val : boolean;
VARIABLE space : character;
BEGIN
WHILE NOT endfile(vector_file) LOOP
readline(vector_file, l);

-- read the time from the beginning of the line
-- skip the line if it doesn't start with a number
read(l, r, good => good_number);
NEXT WHEN NOT good_number;

vector_time := r * 1 ns;        -- convert real number to time
IF (now < vector_time) THEN     -- wait until the vector time
WAIT FOR vector_time - now;
END IF;

read(l, space);  --- skip a space

-- read clk value
read(l, tmpclk, good_val);
assert good_val REPORT "bad clk value ";
```

```
-- read ld value
read(l,tmpld, good_val);
assert good_val REPORT "bad ld value ";

-- read up_dwn value
read(l,tmpup_dwn, good_val);
assert good_val REPORT "bad up_dwn value ";

-- read clk_en value
read(l,tmpclk_en, good_val);
assert good_val REPORT "bad clk_en value ";

read(l, space);  --- skip a space
END;
                 -- read din value
read(l, tmpdin, good_val);
assert good_val REPORT "bad din value ";
clk <= tmpclk;
ld <= tmpld;
up_dwn <= tmpup_dwn;
clk_en <= tmpclk_en;
din <= tmpdin;

END LOOP;
ASSERT false REPORT "Test complete";
WAIT;
END PROCESS;
```

The beginning of the testbench declares entity `testbench` as an entity with no ports. This is completely legal as the testbench is the topmost entity and does not interact with any other entities. Next is the architecture declaration. The architecture uses a number of packages including IEEE standard packages and counter. The next section in the model declares the component for the device under test (DUT), and the counter. The ports and types on this component should match the DUT. Next, the local interconnect signals are declared. After the architecture-declaration section, the DUT component is instantiated and connected to the local interconnect signals. A process called `test` is declared which contains the stimulus generation capability. First, a number of local variables are declared to receive data from the `TextIO` procedures used to read the stimulus information from a file. `TextIO` can only be assigned to variable objects and not to signals; therefore, local variables are assigned by the `TextIO` procedures, and these variables are assigned to the internal interconnect signals. Inside the process is a single `while` loop that reads data from the stimulus file until an end-of-file condition is reached. Each pass through the loop reads another line from the file and reads the appropriate data from that line. The first data read from the line is the time that this vector is to be applied. The process checks to make sure that the value read is a valid number. If not, the line is discarded because it does

not represent a valid stimulus line. This allows comment lines to be inserted in the vector files. If a valid number was not read, the process skips this iteration through the loop and goes to the next iteration using the `next` clause. If the value read was a good number then the vector is assumed to be valid. The process reads each data value from the vector and applies the values to the locally declared variables.

In the counter example, the first value read is the `clk` signal. The `TextIO` statement reads the `STD_LOGIC` value from line l and assigns the value `read` to the variable `tmpclk`. Later, the `tmpclk` variable is assigned to the signal `clk`. The process continues to read a line, read a time value, wait until that time value occurs, read all vector values, and apply vector values until the end of the file is reached. When the end of the file is reached, the loop terminates, an assertion message is written to standard output, and the process waits forever. The `WAIT` statement after the assertion at the end of the loop does not have a termination condition and, therefore, waits forever, effectively stopping execution of this process. The `Textio` readline statement inside the `while` loop reads a vector line from a vector file. Following is an example of a vector file:

```
-- vector file for counter
-- time clk ld up_dwn clk_en din
   10 0001 0
   20 1101 50
   30 0001 0
   40 1001 0
   50 0001 0
   60 1001 0
   70 0001 0
   80 1001 0
   90 0001 0
   100 1101 10
  110 0001 0
  120 1001 0
  130 0001 0
  140 1001 0
  150 0001 0
  160 1001 0
```

The first two lines of the vector file do not start with valid numbers and are treated as comment lines. Comment lines can be embedded anywhere in the file. Comments can also be placed at the end of a vector because any data after the last field of the vector are ignored. Each vector line starts with a time value and then contains a string of values to be assigned to the DUT at that time. Spaces can be embedded between vector values if a corresponding read function exists in the `while` loop to skip the space. For the stimulus-only testbench, the test process reads a vector from the file and applies the stimulus to the DUT. The stimulus-only testbench does not check the output results of the DUT in reaction to the applied stimulus. The stimulus-only testbench is most useful for a quick check of a piece of a design that is easy for the designer to verify manually or for early in the design process when no known good results exist to verify against. When the results are verified, these results become the known good results to verify future versions or minor changes to the design.

10.11.2 Full Testbench

A full testbench is very similar to a stimulus-only testbench except that the full testbench also includes the capability to check the output of the DUT. The full testbench applies the stimulus to the design and then examines the outputs of the design to see if the output results of the DUT match known good test results. Following is a full testbench for the counter:

```
ENTITY testbench IS END;
-- FULL TESTBENCH
-- testbench for counter
-- reads from file "counter.txt"
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE std.textio.ALL;
USE ieee.std_logic_textio.all;
USE WORK.count_types.all;
ARCHITECTURE full OF testbench IS

-----------------------------------
-- component declaration for counter
-----------------------------------

COMPONENT count
PORT (clk    : IN std_logic;
            ld    : IN std_logic;
            up_dwn : IN std_logic;
            clk_en : IN std_logic;
            din  : IN bit8;
            qout : INOUT bit8);
END COMPONENT;

        SIGNAL clk, ld, up_dwn, clk_en : std_logic;
        SIGNAL qout, din : bit8;


BEGIN
-- instantiate the component
uut: count
PORT MAP(clk => clk,
         ld => ld,
                              up_dwn => up_dwn,
                              clk_en => clk_en,
                              din => din,
                              qout => qout);
```

```
-- provide stimulus and check the result
test: PROCESS

VARIABLE tmpclk, tmpld, tmpup_dwn, tmpclk_en :
std_logic;
VARIABLE tmpqout, tmpdin : bit8;

FILE vector_file : text IS IN "counter.txt";
VARIABLE l : line;
VARIABLE vector_time : time;
VARIABLE r : real;
VARIABLE good_number, good_val : boolean;
VARIABLE space : character;

BEGIN
WHILE NOT endfile(vector_file) LOOP
readline(vector_file, l);

-- read the time from the beginning of the line
-- skip the line if it doesn't start with a number
read(l, r, good => good_number);
NEXT WHEN NOT good_number;

vector_time := r * 1 ns;        -- convert real number to time
IF (now < vector_time) THEN     -- wait until the vector time
WAIT FOR vector_time - now;
END IF;

read(l, space);  --- skip a space

-- read clk value
read(l, tmpclk, good_val);
assert good_val REPORT "bad clk value";

-- read ld value
read(l, tmpld, good_val);
assert good_val REPORT "bad ld value";

-- read up_dwn value
read(l, tmpup_dwn, good_val);
assert good_val REPORT "bad up_dwn value";
```

```
  -- read clk_en value
  read(l, tmpclk_en, good_val);
  assert good_val REPORT "bad clk_en value";

read(l, space);  --- skip a space
         -- read din value
read(l, tmpdin, good_val);
        assert good_val REPORT "bad din value";
        read(l, space);  --- skip a space
            ---- the difference in the file is below

  -- read good output value
  read(l, tmpqout, good_val);
  assert good_val REPORT "bad qout value";

  -- Compare outputs
  assert tmpqout = qout REPORT "vector mismatch";

  clk <= tmpclk;
  ld <= tmpld;
  up_dwn <= tmpup_dwn;
  clk_en <= tmpclk_en;
  din <= tmpdin;

  END LOOP;
  ASSERT false REPORT "Test complete";
  WAIT;
  END PROCESS;
  END full;
```

The full testbench looks exactly the same as the stimulus only testbench for most of the file. The full testbench has a top-level entity with no ports, an architecture that instantiates the DUT, and a `while` loop that reads a vector file. The differences are in the `while` loop itself. The first part of the `while` loop is exactly the same. The process reads a time value and waits for that time value to occur. The full testbench is different in that, not only does the full testbench read the input values, but it also reads the output values and then performs a compare operation between the output values from the DUT versus the values read from the file. If a mismatch *n* is found, an assertion message is generated to let the designer know that the output results did not match the known good results. The full testbench also reads from a vector file to get the stimulus for the design and the expected results. The vector file contains a time value, the input values, and the expected output values. Following is the full testbench vector file:

```
  --- vector file for counter
  -- time clk ld up_dwn clk_en din dout
       0 0001  0 0
```

```
10 1001 0 255
20  0101 10 255
30 1001 0 10
40 0001 0 10
50 1001 0 8
60 0001 0 8
70 1001 0 7
80 0001 0 7
90 1001 0 6
100 0101 100 100
110 1001 0 100
120 0001 0 100
130 1001 0 98
140 0001 0 98
150 1001 0 97
160 0001 0 97
```

The full testbench can be used to test whether a DUT matches a specification or not. For that, we should have the proper specification with good test results that the testbench can match against. The full testbench can also verify that a small change or deviation in input still matches the known good results or not. The designer can make the change and rerun the testbench to make sure that the change did not affect the rest of the design. After the design matches the testbench results, the design is ready to be put into production. The stimulus-only and full testbench should have a couple of examples of the many ways that a testbench can be written.

10.11.3 Stimulator-Specific Full Testbench

The simulator-specific testbench is written specifically for one brand of simulator. Most simulators include a command language that allows the designer to control the simulator. The designer can compile designs, load designs, create libraries, set breakpoints, run the simulation, and lots of other tasks using the simulator command language. Most of these simulators also allow the designer to set signals to new values. Using command languages, the designer can write a testbench. Following is an example of a simulator-specific testbench:

```
--- setup the clock
force -repeat 20 clk 0 0, 1 10

-- log the results to a file
list *

-- setup initial signal conditions
force ld 0
force up_dwn 0
force clk_en 1
```

```
force din 16#00

-- run the simulation
run 100

--- set next signal conditions
force ld 1
force up_dwn 0
force clk_en 1
force din 16#AA

--- run the simulation
run 200

--- set next signal conditions
force ld 1
force up_dwn 0
force clk_en 1
force din 16#55
--- run the simulation
run 200
write list data.out
quit -f
```

The command for this testbench uses ModelSim simulator command language that allows the designer to perform all of the necessary operations to compile designs, load designs, debug designs, save designs, and so on. The ModelSim simulator also has the capability to generate repeating clock signals to drive the design. The first command in the testbench file creates a repeating clock which repeats every 20 time units and is set to a '0' value at time 0 and a '1' value at time 10. The next command (`list*`) allows the designer to write all the signal values to an output file. The next commands in the file set up stimulus values on the counter-input signals. The `force` command sets the signal to a value until it is changed by another `force` command. The input signals are all set to an initial value and the `run` command advances simulation time and runs the simulation. All of the input values are propagated appropriately through the design. After the `run` command has finished, the new input stimulus values are set up with more `force` commands, and the simulation is run again. This process continues until all stimulus has been run through the design. The `write` command near the end of the file writes the results of the simulation to a file. The designer can analyze the output file to determine if the design is correct or use a file compare facility to automatically compare the DUT results with known good results. The advantages of a simulator-specific testbench are high speed and easy to generate. The disadvantage of the simulator-specific testbench is that the testbench is specific to one simulator and cannot be easily migrated.

10.11.4 Hybrid Testbenches

Hybrid testbench uses a combination of a number of techniques. Normally, hybrid testbench can use a full testbench approach and some of the stimulus data is generated in the testbench rather than read from a file. The following is a package for hybrid testbench.

```
    ENTITY testbench IS END;
            -- HYBRID Testbench
            -- testbench for 8-bit loadable updown counter
            -- reads from file "counter.txt"

       LIBRARY ieee;
       USE ieee.std_logic_1164.ALL;
       USE std.textio.ALL;
       USE ieee.std_logic_textio.all;
       USE WORK.count_types.all;
ARCHITECTURE hybrid OF testbench IS
-----------------------------------
-- component declaration for counter
-----------------------------------
COMPONENT count
PORT (clk    : IN std_logic;
ld    : IN std_logic;
up_dwn : IN std_logic;
clk_en : IN std_logic;
din  : IN bit8;
qout : INOUT bit8);
END COMPONENT;

SIGNAL ld, up_dwn, clk_en : std_logic;
SIGNAL clk : std_logic := '0';
SIGNAL qout, din : bit8;

BEGIN
-- instantiate the component
uut: count
PORT MAP(clk => clk,
ld => ld,
up_dwn => up_dwn,
clk_en => clk_en,
din => din,
qout => qout);

-- Generate the system clock
clk <= not clk after 10 ns;
```

```
-- provide stimulus and check the result
test: PROCESS

VARIABLE tmpclk, tmpld, tmpup_dwn, tmpclk_en :
std_logic;
VARIABLE tmpqout, tmpdin : bit8;

FILE vector_file : text IS IN "counter.txt";
VARIABLE l : line;
VARIABLE vector_time : time;
VARIABLE r : real;
VARIABLE good_number, good_val : boolean;
VARIABLE space : character;

BEGIN
WHILE NOT endfile(vector_file) LOOP
readline(vector_file, l);

-- read the time from the beginning of the line
-- skip the line if it doesn't start with a number

    END;
read(l, r, good => good_number);
NEXT WHEN NOT good_number;

vector_time := r * 1 ns;        -- convert real
number to time
IF (now < vector_time) THEN     -- wait until the
vector time
WAIT FOR vector_time - now;
END IF;

read(l, space);  --- skip a space

-- read ld value
read(l,tmpld, good_val);
assert good_val REPORT "bad ld value";

-- read up_dwn value
read(l,tmpup_dwn, good_val);
assert good_val REPORT "bad up_dwn value";
```

```
-- read clk_en value
read(l,tmpclk_en, good_val);
assert good_val REPORT "bad clk_en value";

read(l, space);  --- skip a space

-- read din value
read(l, tmpdin, good_val);
assert good_val REPORT "bad din value";

ld <= tmpld;
up_dwn <= tmpup_dwn;
clk_en <= tmpclk_en;
din <= tmpdin;

END LOOP;
ASSERT false REPORT "Test complete";
WAIT;
END PROCESS;
```

After the `counter` component instantiation, the system clock is generated by a signal assignment statement.

Signal `clk` is assigned the value of `not clk` after 10 nanoseconds. This statement creates a periodic waveform with a period of 20 nanoseconds. The testbench does not read signal `clock` from the vector file. The vector file contains changes only on signals other than `clock`. This results in a much smaller file that can be read much faster. Following is the hybrid vector file:

```
--- vector file for counter
-- time ld up_dwn clk_en din
10 001 0
20 101 50
30 001 0
100 101 0
110 001 0
250 101 35
260 001 0
```

The advantage of the hybrid testbench is that less data needs to be read from a vector file. Stimulus data is instead provided by either simulator command language commands or generated in the testbench. The disadvantage of the hybrid testbench is that it is more difficult to change data from run to run when the hybrid testbench generates the stimulus in the testbench. This testbench is less portable.

REFERENCES

10.1 Ashenden, Petre, The designer's guide to VHDL, San Fransisco Morgan Kaufmann, 1995

10.2 Dewey, A., and Anthony, G., *VHDL motivation*, IEEE design and test of computers, New York, IEEE, 1986.

10.3 Lowenstein, A., and Winter, G., *VHDL's impact on test*, IEEE design and test of computers, New York, IEEE, 1986

10.4 Saunders, L., *The IBM VHDL design systems*, Proceedings of the 24th design automatic conference, 1987.

10.5 Leung, S., and Michael, S., *ASIC system design with VHDL: a Paradigm*, Boston, Kluwer academic, 1989.

10.6 Coppola, A., and Lewis, J., *VHDL for programmable logic devices*, 1993 programmable logic devices conference, Santa Clara, March, 1993

10.7 Bhasker, J., *A VHDL premier*, Eaglewood Cliff, N. J., Prentice Hall, 1992.

EXERCISES

10.1 Draw the diagram of 8-bit micro-processor and write VHDL representation of the same.

10.2 Write the VHDL codes of CPU controller performing functions as shown in Fig. 10.7.

10.3 Write the VHDL codes of ALU performing functions as shown in Table 10.1E.

Table 10.1E ALU operations

Sel Input	Operation
0000	C = A
0001	C = A AND B
0010	C = A OR B
0011	C = NOT A
0100	C = A XOR B
0101	C = A + B
0110	C = A – B
0111	A = A + 1
1000	C = A – 1
1001	C = 0
1010	C = A/B
1010	C = A NXOR B

10.4 Write the VHDL codes of comparator performing functions as shown in Table 10.2E.

Table 10.2E Comp operations

Sel input value	Comparison
EQ	Compout = 1 when a equals b
NEQ	Compout = 1 when a is not equal to b
GT	Compout = 1 when a is greater then b
GTE	Compout = 1 when a is greater than or equal to b
LT	Compout = 1 when a is less than b
LTE	Compout = 1 when a is less than or equal to b

10.5 Write the VHDL codes of full testbench and stimulus-specific testbench for verification of 8-bit microprocessor.

10.6 Write the VHDL codes of hybrid testbench for verification of 8-bit microprocessor.

11

VLSI Process Technology

Silicon VLSI IC processes include basic processes involved such as silicon wafer preparation, wafer cleaning, oxidation of silicon, diffusion, ion implantation, epitaxial growth, lithography, metallization, and etching. In this chapter, we have discussed their basic steps.

11.1 ◻ SILICON-WAFER PREPARATION

Silicon is abundant in nature in the form of silica and silicates (i.e., sand). Normally, in integrated circuits, silicon should be a single crystal of very high purity in wafer form obtained from quartzite. It can be refined by using simple purification and crystal-growth techniques. Silicon also exhibits desirable physical properties and good electrical characteristic when used to fabricate active and passive devices on the same substrate. Initially, the starting material silicon is chemically treated to form a high-purity polycrystalline semiconductor from which single crystals are grown. The single crystals are shaped to define the diameter of the material and then sawed into wafers. These wafers are polished to provide smooth and clean surfaces on which devices will be made and then be interconnected to form a monolithic IC.

11.1.1 Preparation of Metallurgical and Electronic Grade Silicon

There are three steps used for fabrication of highly pure silicon wafer:

Step-1: Metallurgical Grade Silicon (MGS) Preparation

Initially, the silicon is obtained from quartzite. The quartzite is reduced in a carbon arc furnace as shown in Fig. 11.1. The furnace consists of quartzite, coal, coke, and wood chips in a small quantity. When heated, the following reaction takes place:

$$SiC + SiO_2 \rightarrow Si + SiO + 3\ CO \tag{11.1}$$

The silicon is obtained in liquid form as shown in Fig. 11.1. SiO and CO are gaseous in form so they leave the furnace. Some amount of condensed layer of silicon oxide (SiO) is formed in the upper layer of the furnace. The liquid silicon is solidified to obtain metallurgical grade silicon (MGS). In the arc furnace, the reaction takes place at a very high temperature of about 1700°C. So the material of the furnace should be such that it is not affected at such high temperatures. The metallurgical grade silicon obtained has high purity of around 98%. The impurities present in MGS are iron, aluminum, and carbon.

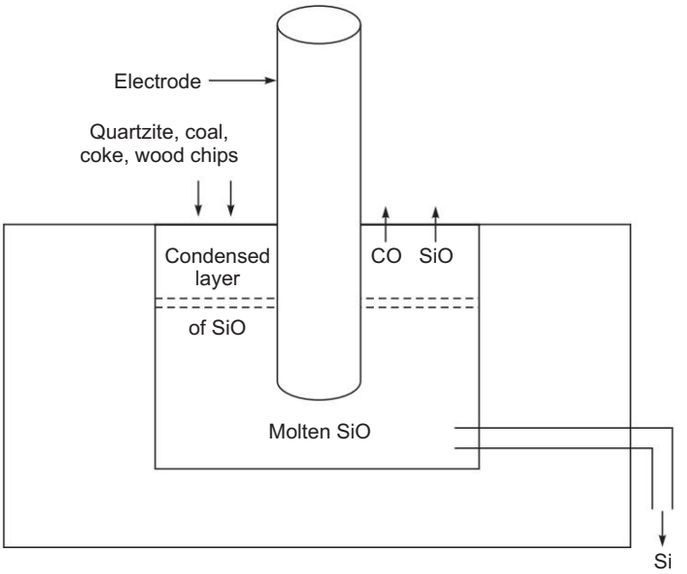


Fig. 11.1 MGS preparation

Step-2: Pyrolysis

MSG is further purified to obtain Electronic Grade Silicon (EGS) which is to be used in integrated circuits. Electronic grade silicon is a polycrystalline material and is highly pure. MGS is first made to powder form. Then, it is treated with anhydrous hydrogen chloride (HCl) in a fluidized bed reactor to form trichlorosilane $SiHCl_3$.

$$Si + 3HCl \rightarrow SiHCl_3 + H_2 \quad (11.2)$$

Trichlorosilane which is liquid in form at room temperature is fractionally distilled to obtain pure trichlorosilane and unwanted impurities (silicon tetrachloride and chlorides of impurities) are removed. The purified $SiHCl_3$ is reduced by hydrogen in a reactor to form EGS by Chemical Vapor Deposition (CVD) process. The chemical reaction involved is given below:

$$2SiHCl_3 + 2\ H_2 \rightarrow 2Si + 6\ HCl \quad (11.3)$$

The apparatus used to prepare EGS is shown in Fig. 11.2.

A resistance heated rod of silicon (slim rod) acts as the nucleation point for deposition of silicon. This results in rods of EGS, which are polycrystalline in structure of about 0.2 m in diameter and several meters in length. EGS is cut from these rods as single chunks or crushed into nuggets.

EGS can also be produced from pyrolysis of silane. In the CVD reactor of Fig. 11.2, silane is used instead of trichlorosilane and temperature is maintained around 900°C. The resulting equation is

$$SiH_4 + \text{heat} \rightarrow Si + 2\ H_2 \quad (11.4)$$

Production of EGS from silane is an easier process as compared to production from trichlorosilane.

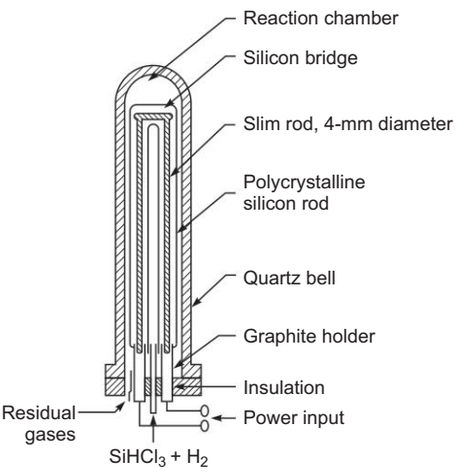


Fig. 11.2 CVD reactor for pyrolysis of silicon

Step 3: Crystal Growth

EGS has very small impurity levels measured in parts per million (ppm) to parts per billion (ppb). EGS is a polycrystalline material and is the raw material for the preparation of single crystal silicon. In crystal growth, the polycrystalline silicon rods (EGS) are converted to even purer and defect-free single crystal wafers. When crystals are grown, a phase change from solid, liquid, or gas to crystalline solid phase has to be made. There are many ways to get single-crystal silicon growth. The **Czochralski technique** is a widely used method. It uses liquid-solid silicon system where the solidification of silicon atoms takes place from a liquid phase at solid-liquid interface.

The growth speed is determined by the number of sites on the face of the crystal and the specifics of heat transfer at the interface. The apparatus used for CZ crystal growth is known as a **puller**. The CZ crystal-growth apparatus is shown in Fig. 11.3. The puller is very bulky and at a time, it can hold

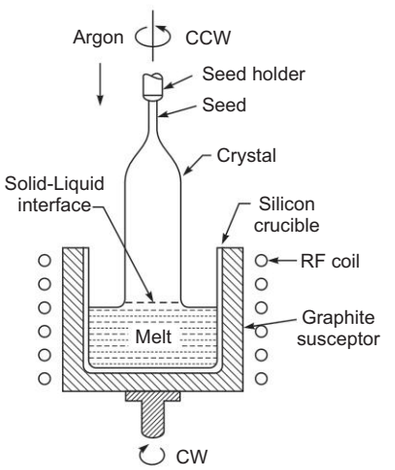


Fig. 11.3 Czochralski crystal-growth technique

about 60 kg of silicon from which, a crystal with 100 mm diameter and 3 m in length can be obtained. The puller consists of four main parts—furnace, crystal pulling system, ambient control, and control system.

The furnace has a fused silica crucible, a graphite susceptor, a rotation mechanism, a heating element, and a power supply. The melt is contained in the crucible which is chemically inactive with molten silicon since the electrical properties of silicon are highly sensitive to the presence of even small amounts of impurities. The melting point of the crucible should be very high for thermal stability and hardness. Fused silica also reacts with silicon, releasing silicon and oxygen into the melt. But the reaction is very slow. The purity of silica also affects the silicon purity. Another material for crucible is silicon nitride. This eliminates oxygen from crucible-grown crystals. The susceptor is used to support the silica crucible. It also provides better thermal conditions. Graphite is used to make the susceptor as it has high temperature properties. The graphite used should be highly pure to prevent contamination of the crystal from impurities. The susceptor is placed on a pedestal whose shaft is connected to a motor that provides rotation. Radio-frequency heating is used for small melts and resistance heating is used for large melts. Finally, the graphite heater is connected to a dc power supply.

The crystal pulling mechanism should control the pull rate and crystal rotation. Hence, it should have minimum vibration and great precision. Seed crystal holder and pulling mechanism should maintain the proper orientation perpendicular to the melt surface. Lead screws are provided to withdraw and rotate the crystal. The crystal leaves the furnace through a tube where ambient gas is directed along the surface of the crystal to reduce its temperature.

Ambient control includes an inert gas source (of argon), a flow control, and an exhaust system. The gas source must meet the purity requirement. An inert-gas environment is required because the hot graphite parts must be protected from oxygen to prevent erosion and the gas around the process should not react with silicon.

The control system includes control of process parameters like temperature, crystal diameter, pull rate, and rotation speed. Programmed process steps are provided through microprocessor-based control and infrared sensors are used for sensing any change of the optimum process parameter.

11.1.2 Crystal Growth

Polycrystalline silicon is placed in the crucible and the furnace is heated to melt silicon. A suitably oriented seed crystal is placed over the crucible in the seed holder. The seed is dipped into the melt. Some part of the seed melts and some touches the liquid surface. The seed is then slowly pulled up. So freezing at the solid-liquid interface occurs and a single crystal is formed. The seed crystal in this form is known as **ingot** or **silicon boule** as shown in Fig. 11.3. In the growth process, a known amount of dopant is added in the melt to obtain desired doping concentration in the grown crystal. Boron is usually added for *p-type* material and phosphorous for *n-type* material.

Two important parameters in CZ growth technique are the pull rate and the growth rate. The amount of silicon solidified on the seed is determined by the pull rate which is the rate at which the seed rod is pulled upwards as the silicon freezes on it. The growth rate is the instantaneous solidification rate. Temperature differentiates these two rates. If pull rate is large, the solidification occurs without single-crystal formation. If pull rate is slow, the re-melting may occur.

Two important principles are involved in the crystal-growth techniques:

- heat transfer at the solid-liquid interface, leading to the formation of single crystal silicon from electronic grade silicon.
- segregation.

11.1.3 Purification of Crystals

The addition of impurities to the melt accompanied by stirring results in a doped liquid from which crystal is grown. The concentration of the solute will be quite different in the solid and liquid phases of a crystal because of energy considerations. We consider C_s as the concentration of solute in the solid and C_L as the concentration of solute in the liquid phase, in the immediate vicinity of the interface. Distribution coefficient K_o may be defined as

$$K_0 = C_s / C_L \tag{11.5}$$

K_o is also known as **segregation coefficient** and the values of K_o for the commonly used dopants for silicon are shown in Table 11.1.

Table 11.1 Equilibrium distribution coefficients for dopants in silicon

Dopant	*P*	*As*	*Sb*	**B**	*Al*	*Ga*	***In***	***Au***
Distribution coefficient γ	0.32	0.27	0.02	0.72	1.8×10^{-3}	7.2×10^{-3}	3.6×10^{-4}	2.25×10^{-5}
Atomic weight	30.97	74.92	121.75	10.81	26.98	69.72	114.82	196.97
Type of Dopant	*n*-type			*p*-type			Deep lying	

11.2 □ WAFER ETCHING, POLISHING, AND CLEANING

Chemical etching of wafer is mainly of two types—acid etching and alkali etching. In the mechanical shaping operations, the surface and the edge of the wafer is damaged and contaminated. This damage and contamination can be removed by etching chemically the wafer as discussed below.

11.2.1 Wafer Etching

In **acid etching,** the wafers are introduced into a acid sink which consists of a tank to hold the etching solution. Wafer rinsing also takes place in the sink. The etchant used is a mixture of nitric acid, hydrofluoric, and acetic acid in the ratio of 4: 1: 3. Etching is an oxidation reduction process. The H^+, F^-, and NO_3 ions are the primary reacting species. The following reaction takes place:

$$3Si + 4HNO_3 + 18HF \rightarrow 3H_2SiF_6 + 4NO + 8H_2O$$

The limitation of nitric acid is that it oxidizes the surface of the wafer to form silicon dioxide. In high HF solutions, the reaction is limited by the oxidation step. Etching is anisotropic and the reaction is sensitive to doping, orientation, and defect structure of the crystal. Also all oxides formed are removed by HF. In high HNO_3 solution, isotropic etching takes place. The rate of reaction is diffusion limited, i.e., is transport of reactant products to the wafer surface across a stagnant boundary layer is the controlling

mechanism. Moreover in acid etching, the dimensional uniformity is not maintained for larger wafers, so alkali etching is preferred for larger wafers.

In **alkali etching**, a mixture of $NaOH/H_2O$ or KOH/H_2O is used. Alkali etching is limited by reaction rate and is intrinsically anisotropic and thus orientation-dependent. Excellent uniformity in the wafer surface can be achieved by alkali etching.

11.2.2 Polishing

The last step in wafer preparations is mechanical and chemical polishing of the wafer surface to obtain a smooth surface on which integrated circuits can be made. A schematic of polishing process is shown in Fig. 11.4. Wafer polishing can be done on a single wafer in a batch-processing manner. The polishing pad is made up of artificial fabric. The wafers are mounted on a fixture, processed against the pad under high pressure and rotated. A mixture of polishing slurry and wafer, dripped on to the pad does the polishing. The slurry consists of colloidal suspension of fine SiO_2 particles in an aqueous solution of sodium hydroxide. Sodium hydroxide oxidizes the silicon, and the silica particles in the slurry abrade the oxidized silicon away. The polishing rate depends on the properties of the pad, applied pressure, composition of slurry, and the rotation speed.

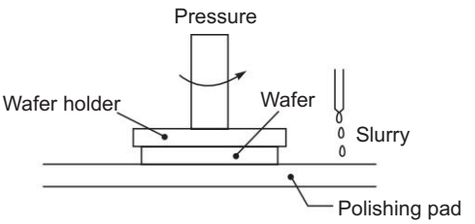


Fig. 11.4 Wafer polishing

11.2.3 Cleaning

The cleaning of wafer plays an important role in fabrication process. A thorough cleaning is necessary as dust or the impurities present may degrade the adhesion of the layers, which would be deposited further on the substrate. Cleaning process involves the following steps:

1. Initial Method

- 2–5 minute soak in deionised water (DI H_2O) with ultrasonic agitation
- 30-second rinse under free flowing DI H_2O
- Spin rinse dry for wafers; N_2 blow-off dry for tools and chucks
- For particularly troublesome grease, oil, or wax stains, the sample was soaked in 1, 1, 1-trichloroethane (TCA) for 2–5 minutes or trichloroethylene (TCE) with ultrasonic agitation prior to acetone cleaning

2. Acetone and Methanol Cleaning

- 2–5 minute soak in acetone with ultrasonic agitation
- 2–5 minute soak in methanol with ultrasonic agitation and dry in N_2 gas

3. RCA Clean

Step 1

- Dip the wafer in DI water: H_2O_2: NH_4OH in 5:1:1 ratio for 10 minutes at 75°C.
- Rinse in DI water.
- Rinse in mixture of DI water and HF in 50:1 ratio (this is done in a Teflon beaker).

Step 2

- Dip the wafer in DI + H_2O_2 + HCL in 6:1:1 ratio.
- Rinse in DI water.
- Rinse again in DI + HF in 50:1 ratio.

11.3 ◻ THERMAL OXIDATION AND OXIDATION SYSTEM

An oxide film can be grown on Si-wafer by using dry-oxidation or wet-oxidation ambient method (oxygen + water vapor) maintaining a high elevated temperature. Oxidation is necessary for growing an oxide layer throughout the fabrication of integrated circuits (IC). Thermally grown silicon dioxide reduces surface-state density of silicon. There are two types of oxide layer—intrinsic and extrinsic silicon dioxide layer. The **intrinsic silicon dioxide layer** consists of fused silicon dioxide having a melting point of 1732°C and highly stable crystalline form. **Extrinsic silicon dioxide layer** is made by the introduction of impurities for enhancing the properties of the layer. The impurities are mainly of substitution and interstitial type. Sometimes, water vapor may also be considered an impurity present in the layer.

Oxidation System

Thermal oxidation has been a principal technique in silicon IC technology. The schematic cross section of a resistance-heated oxidation furnace is shown in Fig. 11.5(a).

It is described below:

- A cylindrical fused quartz tube containing the wafers is held vertically in a slotted quartz boat.
- The wafers are exposed to a source of either pure dry oxygen or pure water vapour.
- The loading end of the furnace tube enters into a vertical flow hood where a filtered flow of air is maintained.
- The hood reduces dust and particulate matter in the air surrounding the wafers and minimizes the contamination at the time of wafer loading.
- The furnace temperature is maintained between 900 to 1200°C.
- Microprocessors are used to regulate the gas flow sequence to control the automatic insertion and removal of silicon wafers, to control temperature, etc.

SiO_2 can be grown very easily by heating the silicon to about 1000^oC in presence of either dry oxygen or oxygen with water vapor (wet oxygen). The resulting oxide has good uniformity, clings tenaciously to the silicon surface and can be grown to a thickness of several micrometers without cracking. The thermal oxidation of silicon is used for the fabrication of the undercladding layer of optical waveguide on silicon substrates. The layer, grown by this technique, exhibits good optical property with relatively low loss.

The experimental set-up consists of an open-tube furnace capable of providing temperature up to 1200°C with a control of ±5°C. The ambient used for oxygen is either dry oxygen or wet oxygen. To obtain wet-oxygen ambient, dry oxygen is passed through a water bubbler maintained at 95°C. The gas regulators are used to control the flow rates of oxygen and nitrogen where.

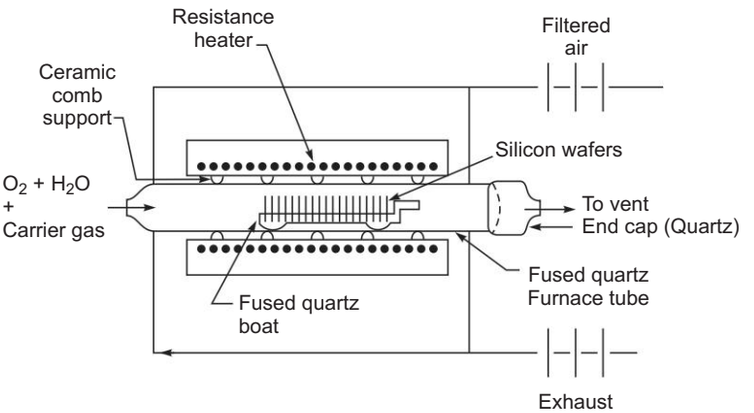


Fig. 11.5(a) Experimental set-up for dry or wet-oxidation system

Prior to oxidation, the silicon wafer is cleaned and finally washed in DI water. The wafers are kept in slotted quartz boat and inserted slowly to the furnace maintained at a particular desired temperature from 1000–1200°C to avoid thermal stress to the wafer. The gas-flow rate is adjusted within the range of 500 cc to 1 liter/min depending upon the quartz-tube diameter. The relation gives the oxidation time for making a sample

$$\frac{t}{x - x_0} = A + B\,(x - x_0) \tag{11.6}$$

where, t = Oxidation time, x = Oxide thickness, x_o = Oxide thickness at the beginning of oxide, and A and B are constants.

The oxidation rate can be written as

$$\frac{dx}{dt} = \frac{1}{A + 2B\,(x - x_0)} \tag{11.7}$$

For the growth of the layer, wet oxidation process is used because the growth rate in case of wet oxidation is ~ 3.5 times faster than that for dry oxidation. The growth rate for wet oxidation process is 0.00388 μm/min at the temperature 1100°C. Figure 11.5(b) shows the cross section of an oxidized layer formed on silicon substrate.

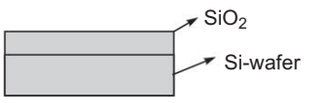


Fig. 11.5(b) Cross-sectional view of oxidized silicon wafer

11.4 ¤ DIFFUSION

Diffusion is the process of introduction of controlled amount of dopant atoms into the semiconductor. Diffusion alters the type of conductivity of the semiconductor. In silicon-integrated circuit processing, diffusion is an important step used to form base, emitter, collector, and resistors in bipolar technology, and source and drain regions of MOSFET's in MOS technology. A wide range of dopant atoms with varied concentrations can be diffused in silicon. Commonly used diffusion methods are—diffusion from a chemical source, diffusion from a doped oxide source, and diffusion and annealing from an ion-implanted layer. Though precise control of dopants in terms of concentration and selectivity can be achieved by using ion implementation but still in silicon IC processing, diffusion is normally used because of easy and cheaper method. In the diffusion process, the movement of atoms can be done by different ways inside the crystal lattice interstitial diffusion, substitution diffusion, and interchange diffusion.

1. Interstitial Diffusion

In this mechanism, the impurity atoms move through the crystal lattice by jumping from one interstitial site to the next as shown in Fig. 11.6. This type of diffusion can also occur by a dissociative mechanism. In this case, the impurity atom occupies both substitutional as well as interstitial sites. Silicon, copper, nickel, and gold are moved by this mechanism.

2. Substitutional Diffusion

In this case, the impurity atoms move through the crystal by jumping from one lattice site to the next with substitution of the original host atom. The presence of vacancies must be required for substitutional diffusion. This mechanism occurs at a much slower rate than interstitial diffusion since the equilibrium concentration of vacancies is low.

3. Interchange Diffusion

Two or more atoms diffuse by an interchange process to cause interchange diffusion. This is a direct process of interchange when two atoms are involved and is a cooperative interchange when larger numbers are involved.

Normally, the interstitial, substitutional, and interchange mechanisms can take place together in diffusion.

11.4.1 Diffusion (Impurity) Properties

Impurities of groups III and V participate in substitutional diffusion for silicon, and their motion is affected by the number and the charge state of lattice-point defects. These impurities are aluminum, boron, gallium, and indium of Group III and antimony, arsenic, and phosphorus of Group V. Impurities belonging to groups I and VIII move interstitially in silicon like the alkali metals, lithium, potassium and sodium, and gases. Transition elements like cobalt, copper, gold, iron, nickel, platinum, and silver are diffused through an interstitial and substitutional mechanism. Diffusion theory is based on two main approaches:

- **The atomistic approach**: It involves interactions between point defects, vacancies, interstitial atoms, and impurity atoms.
- **The continuum approach**: Atomic quantities are ignored and direct relations between the initial and final states can be obtained. It involves Fick's simple diffusion equation which is to be solved with proper boundary conditions and the diffusivity of the impurity atoms.

For higher impurity concentrations, diffusion profile deviates from results of simple diffusion theory (**Fick's laws**). To obtain the diffusion profile for higher concentrations, Fick's diffusion equation with concentration-dependent diffusivities is required. So to understand diffusion theory, we have to know the basic mechanics of diffusion along with the laws or equations that represent the diffusion of various materials in silicon.

1. Diffusion Mechanics

A pure single crystal of silicon is based on face-centered cubic lattice with a diamond structure. Each lattice point of the crystal has a basis of two silicon atoms. The lattice has an ample amount of voids or interstitial space between the lattice atoms. But there are always some structural imperfections or defects. The defects associated with lattice points are called **point defects**. Other types of defects are associated with lines, planes, or an array of points. Point defect is also known as **Schottky defect** and is an empty lattice point in the crystal, the vacancy being created by an atom that moves to the crystal surface or to an interstitial site. The combination of interstitial atom and a vacancy site thus formed is known as **Frenkel defect**. Larger number of Frenkel defects means smaller number of vacant sites for interstitial diffusion. With increase in temperature, the point defects are also generated. If there is a concentration gradient of impurity atoms then these point defects direct atom movement or diffusion. Diffusion in solids can be understood as atomic movement of the different atoms in the crystals by vacancies or interstitials as shown in Fig. 11.6.

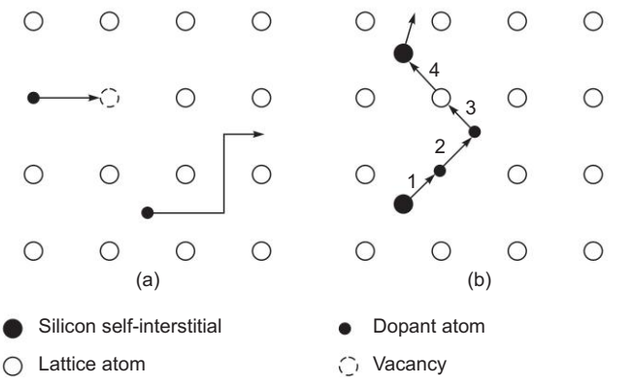


Fig. 11.6 Different atomic diffusion mechanism

The lattice atoms are represented by open circles whereas the solid circles are impurity atoms. At increased temperatures, the lattice atoms vibrate around their lattice sites. Some of the lattice atoms may gain sufficient energy to leave the lattice site and create a vacancy. When a neighboring atom moves to the vacancy site created by the above atoms, it is known as **diffusion by vacancy**. If the migrating atom is a host atom, it is known as self-diffusion. If the migrating atom is an impurity atom, it is known as **impurity diffusion**. An interstitial atom that moves from one place to another without occupying a

lattice site is known as **interstitial diffusion mechanism**. An atom which is smaller in size than the host atoms does not form a covalent bond with a host and it moves interstitially. The activation energy required for the diffusion of interstitial atoms are lower than those of the host atoms by a vacancy mechanism. In the figure, a self-interstitial atom displaces an impurity atom which in turn becomes an interstitial atom. The impurity atom displaces another host atom and this again becomes a self-interstitial atom. Phosphorous, boron, arsenic, and antimony diffuse in silicon through vacancy and interstitial mechanism. But in boron and phosphorus, the interstitial component is dominating and in arsenic and antimony, the vacancy mechanism is dominant. For each mechanism to operate, an imperfection or a defect is required to which the diffusing atom can move. The rate at which diffusion will take place is proportional to the number of imperfect ions present in crystal lattice.

2. *Measurement Techniques*

The result of diffusion can be obtained by conducting a few experiments to measure the junction depth and sheet resistance. Also, the measurement of diffusion profile of different dopants is important to study the way in which the dopants reside in the host lattice and in what concentration.

(a) Study of Junction Depth and Sheet Resistance

- *Measurement of Junction Depth*

The junction depth is measured on an angle-lapped sample chemical staining by using a mixture of 100 CC HF (49%), a few drops of $HN0_3$. and $A1_2O_3$ powder in water (used as the lapping mixture). This sample is put under a strong illumination for 1–2 minutes which would result in darker *p*-type region than the *n*-region. Then, with the help of interference fringe techniques, one can measure the junction depth up to a minimum 0.5 micron and a maximum of 100 microns. The result of staining is the formation of sub-oxide of silicon. In *p*-type region—since the majority carriers are the holes—the surface silicon atoms are oxidized into some form of SiO_x, ($x < 2$) that changes the surface reflectivity and gives a dark stain film of this sub-oxide. The stained junction is located at a place depending on the *p*-type concentration and the concentration gradient. A concentration in the range of mid-10^{17} atoms cm^3 is found at the stain boundary.

Another way is to grind a cylindrical groove into the wafer as shown in Fig. 11.7(a). This visually magnifies the junction region which is delineated by means of a selective etch.

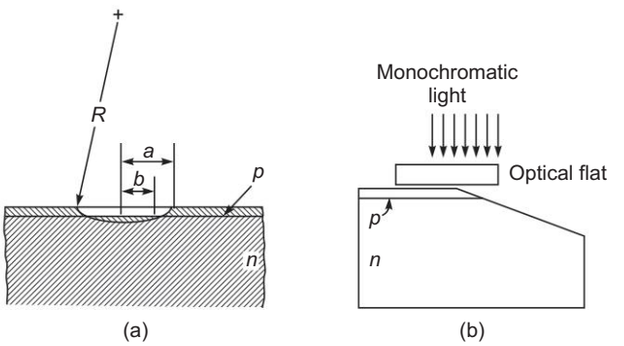


Fig. 11.7 (a) Grinding of cylindrical groove (b) Illumination on optical flat placed on chip

The junction depth can be given by

$$x_j = (R^2 - b^2)^{1/2} - (R^2 - a^2)^{1/2} \tag{11.8}$$

where R is the radius of the tool used for forming the groove, a and b as shown in the figure. For $R \gg a, b$, we can write,

$$x_j \approx (a^2 - b^2)/2R \tag{11.9}$$

The **interferometer method** is preferred with the upper surface of the chip serving as a reference plane when the lapping angle is not known accurately. An optical flat is placed on this chip and is vertically illuminated by collimated monochromatic light as shown in Fig. 11.7(b). The resulting fringe pattern gives a direct measure of the vertical depth in wavelengths of the illuminating source.

- *Sheet Resistance Measurement by Four-Probe Method*

Figure 11.8 shows the four-probe method consisting of four probe points. It is used to measure the sheet resistance of the diffused layer. The sheet resistance is expressed as

$$R_s = (V/I)F \tag{11.10}$$

where V is the dc voltage across the voltage probes, I is the constant dc current passing through the current probes, and F is the correction factor. These four probe points are in line with the dc current passing through the outer two probes and the voltage is measured across the inner two probes. The average resistivity of a diffused layer is R_s. X_j. The correction factors F are different for different kinds of samples (circular and rectangular types). If for a circular sample with a diameter d and a rectangular sample with the side perpendicular to the probe line is termed 'd' and 's' as the probe spacing is taken, then for large d/s (> 40), F is independent to sample types where CF = 4 .53. Table 11.2 shows the correction factor F for different samples.

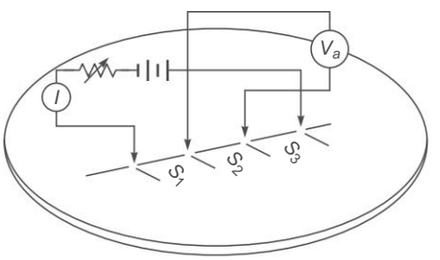


Fig. 11.8 Four-probe method

Table 11.2 Correction factors for square and rectangular samples

	square		rectangle		
d/s	Circle	a/d = 1	a/d = 2	a/d = 3	A/d ≥ 4
1.0				0.9988	0.9994
1.25				1.2467	1.2248
1.5			1.4788	1.4893	1.4893
1.75			1.7196	1.7238	1.7238
2.0			1.9475	1.9475	1.9475
2.5			2.3532	2.3541	2.3541

Contd.

3.0	2.2662	2.4575	2.7000	2.7005	2.7005
4.0	2.9289	3.1127	3.2246	3.2248	3.2248
5.0	3.3625	3.5098	3.5749	3.5750	3.5750
7.5	3.9273	4.0095	4.0361	4.0362	4.0362
10.0	4.1716	4.2209	4.2357	4.2357	4.2357
15.0	4.3646	4.3882	4.3947	4.3947	4.3947
20.0	4.4364	4.4516	4.4553	4.4553	4.4553
40.0	4.5076	4.5120	4.5129	4.5129	4.5129
∞	4.5324	4.5324	4.5325	4.5325	4.5324

For characterizing thin layers, sheet resistivity is measured using a four-point probe system. Four probe tips are arranged in a linear array. The probe force, probe travel, tip radius, and probe material must be selected with consideration for the resistivity hardness and thickness of the layer to be measured. The outer two probes carry current and the inner probes measure the resultant voltage.

(b) Measurement of Doping Profile

• *Differential Conductivity Technique* In this method, the diffusion profile is measured by electrical method. Repeated measurement of the sheet resistance of a diffused layer is done by a four-point probe technique after removing a thin layer of silicon by anodic oxidation and etching the oxide in HF solution. In anodic oxidation at room temperature, the impurity atoms do not move and there is no segregation effect. So, a doping profile can be determined.

• *Secondary Ion Mass Spectrometry Technique (SIMS)* The SIMS technique is used in impurity diffusion profile measurement in VLSI systems and is a very accurate technique.

• *Hall-Effect Measurement* Hall effect is used for measurement of doping profile. When a piece of conductor (sample) of width w (metal and semiconductor) carrying a current I is placed in a transverse magnetic field, an electric field E_H is produced inside the conductor in a direction normal to both the current and magnetic field as shown in Fig. 10.9. This is called **Hall effect** and the generated electric field is called a **Hall field** which gives rise to voltage called a **Hall voltage** (V_H). The Hall voltage across the sample of thickness d is obtained as

$$V_H = R_H . J.B.d = E_H.d \tag{11.11}$$

where R_H = Hall coefficient, J = Current density = $\dfrac{I}{d.w}$

$$R_H = \frac{V_H w}{I.B} \tag{11.12}$$

Measuring Hall coefficient by using the above equation, we can write the carrier concentration as:

$$n = \frac{1}{|eR_H|} \tag{11.13}$$

where e = Charge of electron.

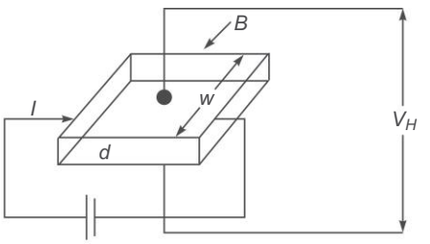


Fig. 11.9 Set-up for Hall-effect measurement

11.4.2 Diffusion Systems

Diffusion systems used in silicon technology are categorized as **predeposition**, where the dopant is to be transported into the wafer; and **redistribution**, where the impurity is redistributed within the wafer as a result of increased temperature. The basic requirement of any diffusion system is that a means should be provided for bringing the diffusion impurity in contact with a suitably prepared wafer. The surface concentration should be controllable over a wide range of values. Diffusion process should not damage the wafer surface. After diffusion, the material lying on the wafer surface should be easily removable. The diffusion system should give reproducible results from one diffusion run to the next and from slice to slice within a single run. The system should be capable of batch processing. There are two types of diffusion systems—sealed-tube type and open-tube type.

Here, the dopants and wafer are enclosed in a clean, evacuated quartz tube prior to heat treatment. After the diffusion process, the slices are removed by breaking the tube. This is a contamination-free diffusion. **Sealed-tube systems** operate by thermal evaporation of the dopant source, transport in the gas phase, adsorption on the surface of the semiconductor, and finally the diffusion of the dopant into the wafer/slices is made. Usually, the **open-tube method** is used where the wafer is kept in a highly temperature-controlled furnace. These furnaces have a ceramic core on which a Kanthal (alloy of AI, Cr and Ni) wire is wound as a heating element. The heating coil has three separate temperature-controlled zones. For temperature control, a Pt-PtRh thermocouple is used to sense the temperature in each of the zones. The central-zone thermocouple compares with a reference set temperature and the outer-zone thermocouple compares with additional thermocouples placed in the central zone. The quartz furnace tube is open at both ends. There are two ends—**source end** in rear side and **load end** in the front side through which wafers are loaded in a quartz boat.

The dopant is a gaseous, liquid, or a solid source. In **gas-source diffusion**, a gaseous compound of the dopant is mixed in appropriate proportions with a carrier gas like nitrogen, and introduced from the source end. In a **liquid-source diffusion**, a liquid compound of the dopant is kept at a fixed temperature in a bubbler (near source) and the carrier gas is bubbled through it. In **solid-source diffusion**, a solid compound of the dopant is taken. The choice of dopant source depends on the way the dopant is presented to the wafer.

1. Solid-Source Diffusion System

Figure 11.10(a) shows a solid-source diffusion system where a platinum boat is used to hold a source of dopant oxide upstream from the carrier with the semiconductor wafers. The carrier gas transports vapours from this source and deposits them on the wafers. Source shut-off is done by moving the dopant

source to a colder region of the furnace. Sometimes, a two-temperature furnace is also used where the source is maintained at a lower temperature than used for diffusion. Solid sources can also be directly placed on the semiconductor slice by chemical vapour deposition. Deposited solid sources allow the use of very dilute dopant concentrations by adjusting the dopant-to-binder ratio. Diffusion from these sources result in surface concentration which is controlled by the concentration of the dopant in the oxide and not by its solid solubility limit in the semiconductor.

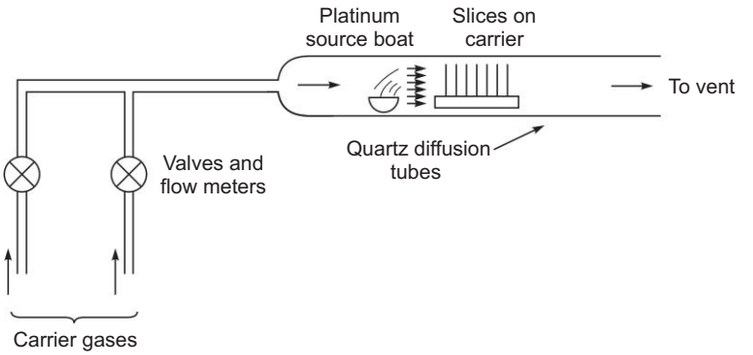


Fig. 11.10 (a) A solid-source diffusion system

2. *Liquid-Source Diffusion System*

Figure 11.10(b) shows a liquid-source diffusion system where a carrier gas is bubbled through the liquid which is transported in vapour form to the surface of the wafer. This gas is saturated with the vapour so that the concentration is relatively independent of gas flow. Thus, the surface concentration is entirely set by the temperature of the bubbler and of the diffusion systems. Liquid-source systems are convenient as compared to solid-source systems since the doping process can be readily initiated by control of the gas through the bubbler.

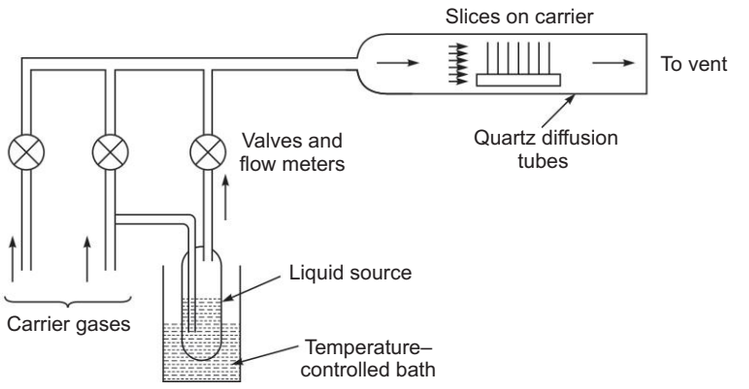


Fig. 11.10 (b) A liquid-source diffusion system

3. Gaseous-Source Diffusion System

Figure 11.10(c) shows a gaseous-source diffusion system where surface concentration can be controlled by adjusting the gas flow. Gaseous sources are more convenient than liquid-source systems. In gaseous-source diffusion system, provision is made for an ambient carrier gas in which the diffusion takes place. A chemical trap is often incorporated to dispose of unreacted dopant gases.

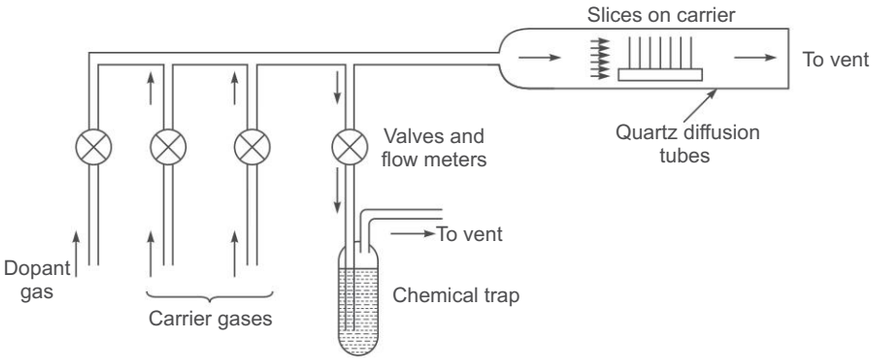


Fig. 11.10(c) A solid-source diffusion system

The dopant sources for diffusion of *p*-type and *n*-type impurity and their doping mechanism are given below:

(a) Boron Boron is a *p*-type impurity which is doped by using predisposition and the sources for boron doping are B_2O_3, BN, BBr_3, and B_2H_6. In open-tube method for B_2O_3, the source B_2O_3, is coated on to the diffusion tube and the wafers are placed in a jig with the polished surfaces to be doped, facing the walls. B_2O_3 is transported onto the wafer as HBO_2. At the wafer surface, it reacts with surface oxides to form borosilicate glass, which later acts as the source for predisposition. The diffusion reaction is given by

$$2B_2O_3 + 3\ Si = 4B\ + 3SiO_2$$

BBr_3 is the most useful source of boron diffusion in silicon as it covers most conditions in silicon processing. The BBr_3 transported to the wafer by the nitrogen gas—which is a carrier—encounters an O_2 stream which converts it to B_2O_3 in the diffusion tube. The B_2O_3 forms a boron-rich glass on the wafer surface. Here the reaction is:

$$4\ BBr_3 + 3\ O_2 \rightarrow 2\ B_2O_3 + 6Br_2$$

B_2H_6—the gaseous diborane is mixed with argon or nitrogen and a small percentage of oxygen to convert the diborane to 82°C which forms a boron-rich glass at the surface. The preliminary oxidizing reaction is

$$B_2H_6\ +\ 3\ O_2 \rightarrow B_2O_3 + 3\ H_2O$$

Higher surface concentrations can be obtained for low temperature (800–900°C) pre-deposition by using carbon dioxide rather than oxygen as

$$B_2H_6\ +\ 6CO_2 \rightarrow B_2O_3 + 6\ CO + 3\ H_2O$$

(b) Antimony Antimony is an n-type impurity which is doped by using predisposition. The diffusion is carried out by using Sb_2O_3 or elemental Sb as source. When using Sb_2O_3 as a source, the trioxide is loaded as a powder in a boat and kept in a different source furnace tube having the same axis as the main diffusion tube and kept at a temperature of 550–600°C. The wafers are kept downstream at a higher temperature (1250°C). Carrier gas used is N_2 and O_2. Antimony and silicon wafers can also be placed in a vacuum inside a silica capsule which is heated to the diffusion temperature. Antimony vapors are transported to the silicon wafers and diffuse onto it.

(c) Arsenic Arsenic is an n-type impurity which is doped by using predisposition. The diffusion sources for arsenic are arsine, arsenic trioxide, and arsenic capsules. In a open-tube gaseous source, diffusion is carried out at temperatures of 1100–l300°C. As_2O_3 is formed by reaction of arsine with oxygen in the presence of inert carrier gas of argon or nitrogen. As_2O_3 is transported to the wafer surface where it forms an arsenic-rich glass. In As_2O_3 process, the trioxide is evaporated in a source furnace and transported to the wafer by a carrier gas. The evaporation of the source takes place at around 350–450°C.

(d) Phosphorus Phosphorous is an n-type impurity which is doped by using predisposition. The sources for phosphorus diffusion are P_2O_5- phosphine, $POCl_3$, and phosphorus-doped silicon capsules. In $POCl_3$ systems, the source is kept in a glass bubbler at a low temperature. Nitrogen is bubbled through and then later mixed with an N_2-O_2 mixture. $POCl_3$ reacts with O_2 to form P_2O_5 which reacts with silicon to form phosphosilicate glass. Diffusion of phosphorus takes place in a temperature range of 900–1200°C. It is doped in a gaseous-source diffusion system.

11.5 □ IMPLANTATION SYSTEMS

In VLSI process technology, it is required to introduce small quantities of dopant atoms. For doping in a controlled manner, the doping should be reproducible and free from undesirable side effects. So to meet this need, the ion-implantation method has been developed. The ions are generated normally by ionization of gaseous chemical compounds and then these ion species required to be doped are selected by an appropriate magnetic field. These ion species are then focused to form an ion beam which is accelerated by application of electric field and targeted to the area to be doped.

The basic requirement for an ion-implantation system is to deliver a beam of ions of a particular type and energy to the silicon surface for doping of impurities. The ion-implantation machine has ion sources, accelerators, systems for ion species selection, and systems for manipulation of the ion beam over the semiconductor target or beam scanners. Since even a basic ion beam requires a large vacuum system, all the subsystems are housed within a number of interconnected vacuum systems as shown in Fig. 11.11(a). A heated filament requires the molecules to break up into charged fragments. The ion plasma contains the desired ion together with many other species. An excitation voltage accelerates the charged ions to move out of the ion source into the analyzer. The pressure is kept below 10^{-6} torr to minimize ion scattering by gas molecules (present in the tube). The magnetic field is so chosen that only ions with desired charge-to-mass ratio can travel through without being blocked by the analyzer walls. The ions continue to the acceleration tube for acceleration to the implantation energy as these move from high voltage to ground. Small apertures are used to keep the beam collimated. The beam is then scanned over the surface of the wafer using electrostatic deflection plates. A commercial ion

implanter is typically 6 m long, 3 m wide and 2 m high, consumes 45 kW of power, and can process 200 wafers per hour with ion dose of 1015 ion/cm in 100 mm wafers. A simplified block diagram of an ion-implantation system is shown in Fig. 11.11(b). Also, a quadruple lens system is placed after the accelerator column to focus the beam on to the target.

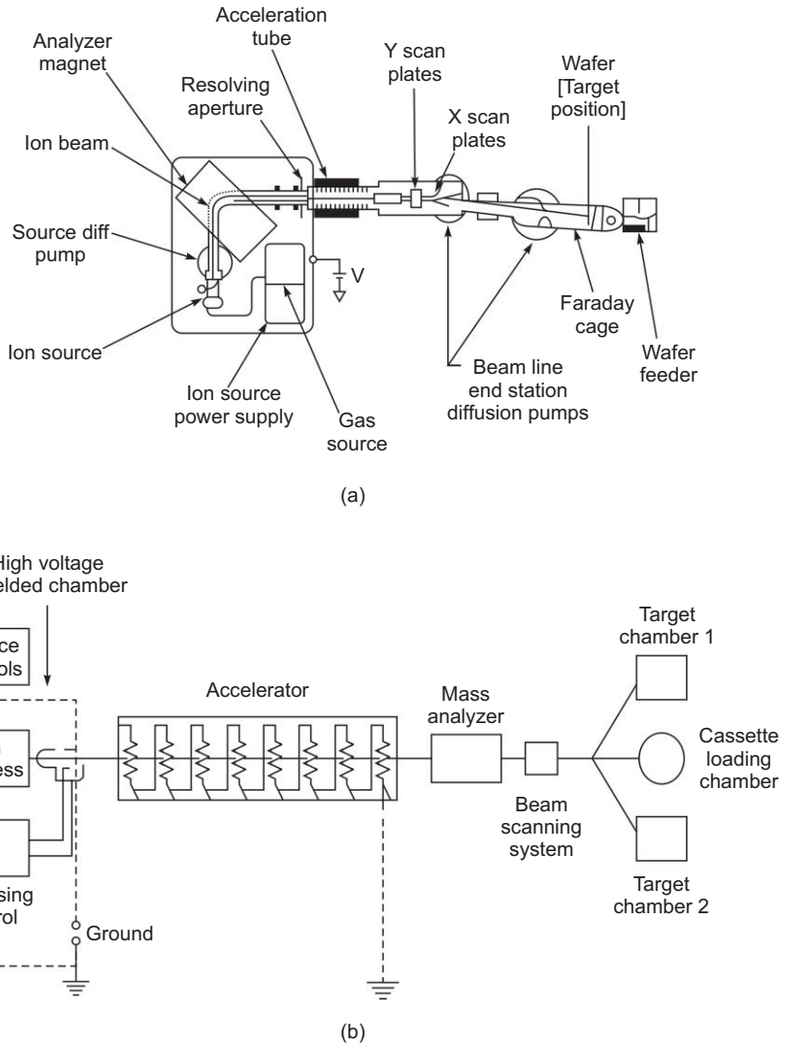


Fig. 11.11 Ion implantation (a) Ion-implementation system (b) Schematic block of simplified ion implantation

In order to get controlled doping to the silicon target, the following steps/systems are required:

- A wide variety of beam-scanning techniques such as electrostatic, magnetic, or mechanical systems are used. Also, hybrid systems consisting of all three techniques are sometimes used.
- The ion current is also monitored precisely, so that the dose of dopants introduced is also controlled.
- A variety of auxiliary subsystems like various vacuum techniques and high-voltage systems are used.

- The vacuum system is so designed that individual access to the target chambers is provided, while keeping the rest of the system under high vacuum. Oil-free pumping systems are used to avoid contamination due to cracking of hydrocarbons in residual oil vapours.

1. Ion Sources

Ion sources consist of compounds of the desired species which are ionized before delivery to the ion accelerator column. Any ionizable compound can be used as ion source. Gaseous materials are more convenient to use than solid ones since they avoid the necessity of having to use a vaporization chamber. The dopant material must be ionized. This is done by passing the vapour through a hot or cold-cathode electronic discharge. A magnetic field is provided so as to force the electrons to move in a spiral trajectory which increases the ionizing efficiency of the source. The Lorentz force due to the magnetic field B acting on the charged particles is balanced by the centrifugal force on the particles.

$$F_L = q\ (v\ x\ B) = mv^2/r$$

The accelerating potential V on the ions is responsible for the velocity of the ions.

$$qV = \tfrac{1}{2}\ mv^2$$

So the minimum magnetic field is

$$B_{\min} = \frac{1}{r_{\max}}\sqrt{\frac{2mV}{q}} \tag{11.14}$$

The radius of the spiralling ions (r) must be less than the chamber dimensions (r_{max}). The magnetic field B must exceed a minimum value to confine the ions and prevent them from striking the chamber walls. A means for extracting the ions from the discharge should also be provided which are then fed to the accelerator column. The outlet of the discharge tube is either a circular or rectangular slit and it defines the cross section of the ion beam. A schematic of a Nielson source is shown in Fig. 11.12.

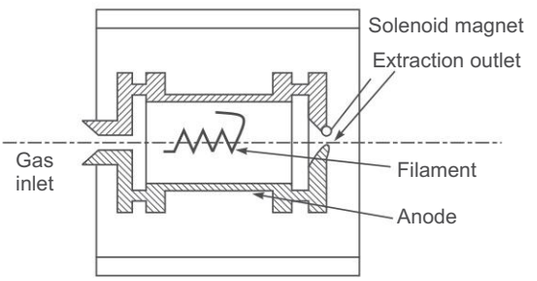


Fig. 11.12 A schematic of a Nielson ion source

2. Ion-Dose Measurement

The total number of ions N entering the target per unit area is known as dose. If the ion beam current is I, then for a beam swept over a target area A, the dose is given by

$$N = \frac{1}{qA}\int I.dt \tag{11.15}$$

The circuit between target and ion source is completed through a current-measuring device called Dosimeter which captures all secondary electrons. A schematic of a Dosimeter is given in Fig. 11.13. A large current is highly desirable to allow a large number of wafers to be handled. For accurate current measurements, all the secondary electrons emitted by the target by the incident ions must be recaptured.

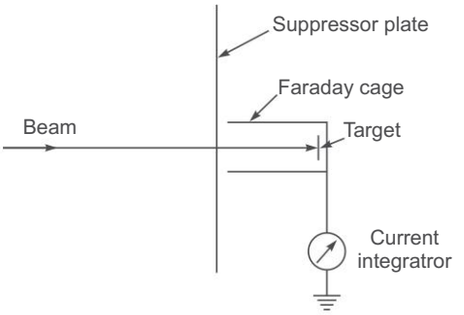


Fig. 11.13 Block diagram of dosimeter

3. Accelerator

Energy is given to the ion beam by passing it through a long column across by the accelerating potential. The output end of this tubular column is normally at ground potential. The energy of the beam determines the projected range of the ion. By using multiple charged ions, it is possible to increase the projected range.

4. Mass Separator

The ion source materials are usually compounds of the ion that is required for doping like BF_3, B_2H_6, and BCl_3, etc. By ionization, one or more charged species are generated. These ions may have other impurity contaminants. The ion separation on the basis of their masses is required for getting pure doping. By mass separation techniques, a unique distinction between ion implantation and diffusion is provided and due to this, a variety of dopants can be handled in a single machine with no effect from each other. A homogeneous-field magnetic analyzer is used with the dynamics of a charged particle of mass III and velocity v moving at right angles to an applied magnetic field of flux density B. The particle will experience a force F such that

$$F = q\,(v \times \mathrm{B})$$

This force tends to move the particle in a circular path of radius r given by

$$r = 1/\mathrm{B}\,(2mV/q)^{1/2} \tag{11.16}$$

The magnetic field B is so adjusted that radius r corresponds to the physical radius of the magnetic analyzer for the desired ion beam. So, only the desired ions will be accepted. Trajectories for different masses are shown in Fig. 11.14.

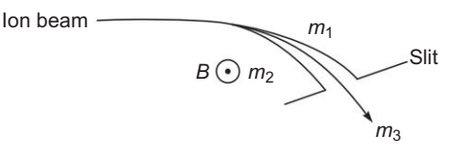


Fig. 11.14 Trajectories for ions of masses m_1, m_2, and m_3

Ions of any particular mass can be selected by the appropriate choice of the magnetic field B. The selected ions are, then, accelerated to the required implantation energy by the voltage applied to the accelerator column. An implanter requires uniform coverage and should be able to handle a number of wafers in a single pump-down. Hybrid diffraction systems are generally used where the beam is electronically scanned in one direction while the slices are mechanically moved in the other.

5. Beam Scanning and Ion Beam Heating

The transfer of energy from the incident beam takes place which heats up the wafer. Since ion implantation is carried out in a vacuum environment, radioactive cooling takes place. In case of radiative cooling, a temperature rise of higher order of 100°C may occur. In case of conductive cooling, the temperature rise can be limited to less than 5°C independent of the dose. So, the target wafers have a provision for conductor convective cooling. The target wafer is clamped to a sealing surface that has an opening at the back for conductive cooling. The heating effects impose limitations on the ion-implantation process by controlling the beam current density and hence on the implantation time required to achieve a desired dose.

Annealing

Ion implantation damages the target and displaces many atoms from the target lattice. The electrical behavior after implantation is dominated by deep-level electron and hole traps. Annealing is required to repair lattice damage and put the dopant at substitutional sites where these dopant atoms will be electrically active. In VLSI application along with repairing of damage and activate dopant, annealing should minimize diffusion so that shallow implants remain shallow. The different methods for annealing are the following:

1. Furnace Annealing

Annealing characteristics depend on the dopant type. Furnace annealing is mainly used for repairing of amorphous silicon. For amorphous silicon, regrowth takes place by solid-phase epitaxy which is done in furnace at high temperature. The amorphous/crystalline interface moves towards the surface at a fixed velocity that depends on temperature doping and crystal orientation. Since the activation energy for solid-phase epitaxy is 2.3 eV, the process involves bond breaking at the interface. If the implantation process does not create an amorphous layer then lattice repair occurs by the generation and diffusion of point defects which requires an activation energy of 5 eV and temperature of the order of 900°C to remove the defects.

Annealing an amorphous layer at high temperatures causes competition between solid-phase epitaxy and local diffusive rearrangement that can lead to polysilicon formation. So, a high-temperature step is preceded by a low-temperature regrowth. High-temperature defect diffusion can, then, repair

extended defects remaining after solid-phase expitaxy. Annealing a partially damaged layer at low temperatures can impair the process of lattice reconstruction since stable extended defects can be formed. When implantation does form an amorphous layer, there must be a transition region of partially damaged material at the tail of the implanted distribution. So in this region, stable defect may be formed. If the surface layer remains crystalline, solid-phase epitaxy can occur from both sides towards the middle and leaves a string of misfit dislocations at the center due to mismatch of the layers. Incomplete annealing results in a reduction in the fraction of active dopant. This is most apparent around the peak of the dopant distribution where damage is greatest.

2. *Rapid Thermal Annealing*

Rapid Thermal Annealing (RTA) is used for repairing of lattice damage in a single crystal silicon. The main aim of RTA is to repair lattice damage which is a process with an activation energy of 5 eV and diffusion should be minimized which is a process involving activation energy in the range to 3 to 4 eV. At higher temperatures, repair is faster than diffusion. Rapid thermal annealing covers various methods of heating wafers for periods from 100 seconds to a few nanoseconds allowing repair of damage with minimal diffusion. Rapid thermal annealing is done by three different ways:

(a) Adiabatic Here, the heating time is so short that only a thin surface film is affected. A high-energy laser pulse is used to melt the surface to a depth of less than 1 μm and the surface recrystallizes by liquid-phase epitaxy. Dopant diffusion in liquid state is very fast. By adjusting the pulse time and energy, shallow junctions can be obtained.

(b) Thermal Flux This kind of annealing occurs on time scales between 10^{-7} and 1 s, where heating from one side of the wafer with a laser, electron beam, or flash lamp gives a temperature gradient across the wafer thickness. The surface is not melted but surface damage can still be repaired by solid-phase epitaxy before any diffusion has time to occur.

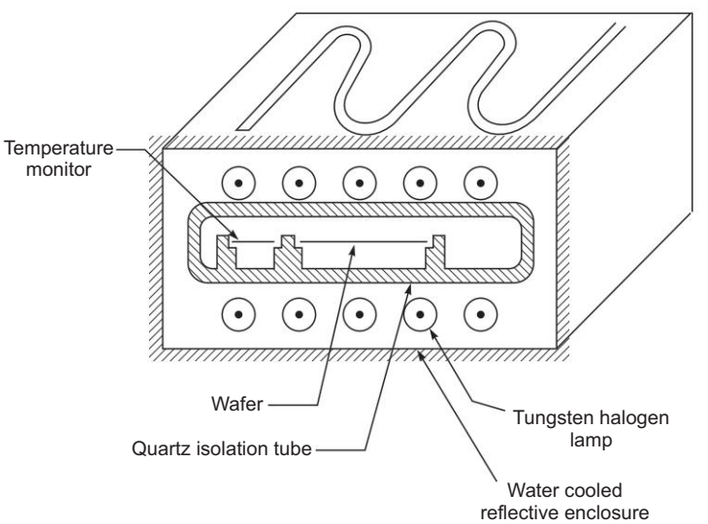


Fig.11.15 A schematic of an isothermal rapid annealing system

(c) Isothermal This covers heating process longer than 1 second. It uses tungsten-halogen lamps or graphite resistive strips to heat the wafer from one or both sides. A schematic of an isothermal rapid annealing system is shown in Fig. 11.15. In this case, good activation can be obtained with less diffusion than furnace annealing.

11.6 □ CHEMICAL VAPOUR DEPOSITION

Chemical processes are used for deposition of films or removal of material from the surface. Normally chemical deposition processes are carried out by Chemical Vapor Deposition (CVD) where the mass transport of the reactants are made in vapor phase and reacted and deposited on the substrate. Chemical vapor deposition technique is one of the most useful methods for the deposition of materials. In CVD, one or several gaseous species are thermally broken into their components. The components, then, impinge on the substrate. Some of the chemical vapor components nucleate on the substrate and form a thin film. The process of thermal breakdown is called **pyrolysis**. The deposition of CVD is controlled by mass transport, and reaction-limited process. The various transport and reaction processes are illustrated below:

1. Mass transport of reactants and diluent gases in the bulk gas-flow region from the reactor inlet to the deposition zone
2. Gas-phase reactions leading to the film precursors and by-products
3. Mass transport of film precursors and reactants to the growth surface
4. Absorption of film precursors and reactants to the growth surface
5. Surface migration of film formers to the growth sites
6. Desorption of by-products of the surface reactions

Some of the most common techniques are—Metallorganic CVD (MOCVD), Low-Pressure CVD (LPCVD), Atmospheric-Pressure CVD (APCVD), and Plasma-Enhanced CVD (PECVD). Figure 11.16 shows the different types of CVD.

11.6.1 MOCVD

This technique uses a thermally heated chamber. The sources used here are organometallic in nature. It provides thickness control within one atomic layer. Figure 11.16(a) shows the schematic set-up for MOCVD. The starting material (the respective accetyl acetonate) in the form of very fine powder is kept in an unheated receptacle and argon gas is bubbled through it. The argon flow rate is adjusted so

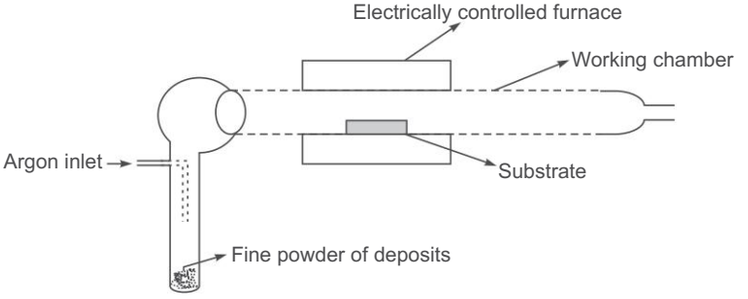


Fig. 11.16(a) Schematic of MOCVD

that the fine argon-borne particles are transported into the working chamber. Some of the fine particles settle on the substrate on the holder at the center of the furnace. The temperature of the furnace is generally kept to be at ~420°C. It is generally used for the fabrication of many optoelectronic devices with III–V compounds for solar cells, lasers, LEDs, HEMTs, etc.

11.6.2 APCVD

Figure 11.16(b) shows the schematic diagram for APCVD. This technique does not require any vacuum. At slightly reduced pressure to atmospheric (~ 100–10 kPa), it can be performed in continuous process with transporting a mass of deposited gas into the substrate, which is at a temperature between 300–400°C. The disadvantages of the processes are poor step coverage and particle contamination.

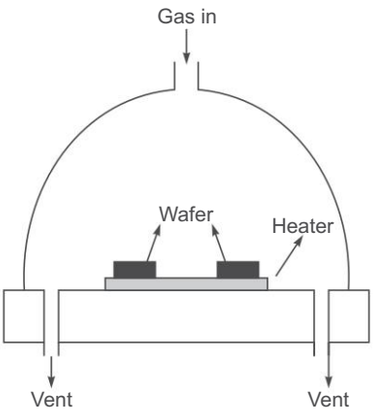


Fig. 11.16 (b) APCVD machine

11.6.3 LPCVD

It allows a large number of wafers to be fabricated simultaneously without any detrimental effects to film uniformity. This is the result of the large diffusion coefficient at low pressures (~100 Pa) leading to a growth limited by the rate of surface reactions rather than by the rate of mass transfer to the substrate. The surface reaction rate is very sensitive to the temperatures but the temperature is relatively easy to control. The deposition rate of 1μm/h has been achieved in a production environment. Figure 11.16(c) shows the schematic diagram of an LPCVD. The substrate is placed on a holder in a chamber, which is heated by a heater in a controlled manner. The substrate temperature is kept in between 500 –1000°C.

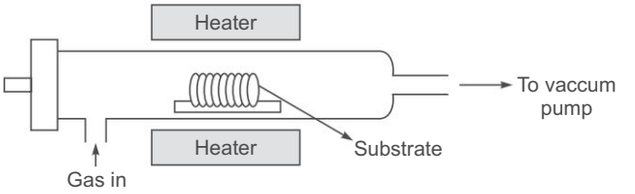


Fig. 11.16 (c) LPCVD machine

In this case, not only the wafers/substrates but also the reaction chamber walls get coated during processing. So it requires frequent cleaning to avoid contamination during deposition. It allows a large number of wafers to be fabricated simultaneously without any detrimental effects to the film uniformity. This is the result of the large diffusion coefficient at low pressures (~100 Pa) leading to a growth limited by the rate of surface reactions rather than by the rate of mass transfer to the substrate. The surface-reaction rate is very sensitive to the temperatures but the temperature is relatively easy to control. This method gives advantage of front and rear deposition. They find wide applications due to their economy, throughput, and uniformity. Two main disadvantages are low deposition rate and relatively high operating temperatures.

Deposition of Thin Film SiON Layer Using LPCVD

The deposition of SiON layer is made from SiH_2Cl_2, O_2, and NH_3 gases with LPCVD. The processed conditions are pressure = 100 mTorr, temperature = 900°C. The deposition rate depends on different flow rates of the reactant gases. The production of SiON film by LPCVD method with these specific parameters is mostly performed for the deposition of waveguides in higher refractive index range ($n \geq 1.7$).

$$SiH_2Cl_2 + O_2 + NH_3 \rightarrow Si\,O_xN_yH_z\,(solid) + H_2O(gas) + HCl(gas)$$
$$\rightarrow Si\,O_xN_y\,(solid) + H_2O(gas) + HCl(gas)$$

(after annealing at temperature 1150°C) where $x = \dfrac{X}{X+Y}$ = Concentration of oxygen, $y = \dfrac{X}{X+Y}$ = Concentration of nitrogen, X = Flow rate of O_2, and Y = Flow rate of NH_3.

11.6.4 PECVD

In this method, the gases used for depositing films are dissociated by electron impact in glow discharge plasma. The substrates are placed horizontally and remain at lower temperatures than that in APCVD and LPCVD. In this process, the RF induced plasma ions are bombarded and transfer energy into the reactant gases. These gas particles are forced to impinge on the substrate and get deposited on it, due to adhesive property. This process is suitable for the deposition of doped and undoped SiO_2 and Si_2N_4. A very high deposition rates (> 1 μm/min) and refractive index uniformity better than 1% and refractive index uniformity of ~ 10^{-4} can be achieved with this process. This allows the deposition of relatively dense oxide films at temperature of ≤ 300°C. Low RF frequency, low reactor pressure, and low deposition rate at high temperature may improve the quality of deposited film.

1. Deposition of SiO_2/SiON Layer by using PECVD

As stated earlier, the PECVD system is used for the deposition of SiO_2, Si_3N_4, and SiON layer due to lower temperature of operation and good uniformity of refractive index and thickness. The PECVD system used in processing of SiON layer is a parallel-plate-type reactor, as shown in Fig. 11.16(d). The glow discharge (plasma) generated by radio frequency, takes place between two electrodes which are separated by 2 cm. The plates are of 24 cm diameter and the RF power establishing the plasma is applied to the upper electrode while the samples are placed on the bottom grounded electrode which is temperature stabilized in the range from 200°C to 350°C. The system can be operated at pressure range of 0.01–10 torr and the applied RF power (13.56 MHz) can have values up to 300 watts. The films in

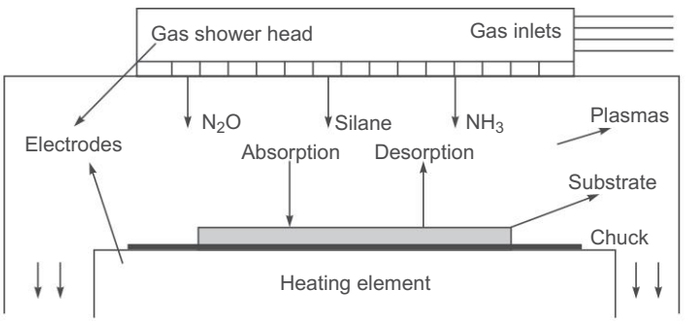


Fig. 11.16(d) PECVD parallel plate plasma reactor with capacitive coupling of rf power

the device structure are grown by making use of an intermediate-state glow discharge which contains mainly ionized gas. The ionized gas has positive ions, negative ions, and electrons. The electron energy takes values between 1–20 eV and density varies from 10^9 to 10^{12} /cm^3. The first step in the process of deposition is generation of reactant species via impact reactions with plasma electrons which are created by glow discharge. For example, electron impact reaction for SiH_4 is given below:

$$\begin{aligned} e^- + SiH_4 &= SiH_2 + H_2 + e^{-1} \\ &= SiH_3 + H_2 + e^{-1} \\ &= SiH + 2H_2 + e^{-1} \end{aligned}$$

This process is observed during the deposition as a glow emitted from the plasmas. The process of deposition of solid films on a substrate using gaseous precursors involves several complicated steps. There are two extreme cases of film-deposition conditions. The first case is observed at the conditions of low pressure and high RF power. In this case, the number of reactive species arriving at the surface is less than the reaction rate. This is called **mass-transfer limited reaction**. In the second case, the number of reactive species arriving at the surface is more than the reaction rate. This is called **reaction-limited process**, which is observed at the low RF power and low temperature. For efficient growth of the desired film, the number of reactive species arriving at the surface should be equal to the reaction rate. For that, a careful optimization of the reaction chamber has to be conducted.

2. *Silicon Dioxide (SiO_2)*

The basic reaction for the formation of the silica using SiH_4 and N_2O is given below:

$$SiH_4\,(gas) + 4\,N_2O = SiO_2\,(solid) + 2H_20(gas) + 4N_2\,(gas)$$

The reaction of SiH_4 with N_2O is based on the oxidation of silane by molecular oxygen produced by dissociation of N_2O. The external RF power source accelerates the electrons in the reaction chamber as shown in Fig. 11.16(d). The primary initial electron-impact reactions between electron and reactant gases form ions and radical reactive species, i.e, molecular oxygen. This molecular oxygen reacted with silane gas gives SiO_2. Thus, actually oxygen from N_2O produces SiO_2 for which refractive index becomes lower than that of SiON which is formed by N_2O and NH_3. The uniformity of the refractive index and thickness of the deposited layer are needed for small size of substrate. Silicon oxide layer is deposited on them in a single run using the following process parameters [59] for a process time of 20 minutes.

RF power @ 13.56 MHz --	10 W
Process pressure -----	1000 mTorr
Silane (2%SiH_4/ N_2) flow rate --	180 sccm
Substrate ---	Silicon

3. Silicon Nitride

Silicon nitride, other than oxide, is a material, the properties of which would be of interest as we expect it to be the other extreme for silicon oxynitride layers to be grown. The stochiometric ratio of silicon nitride is known to be 3/4 (Si_3N_4), however again PECVD silicon nitride films deviate from this because of the incorporation of hydrogen into the layers. The basic reaction mechanism of formation of silicon nitride (Si_3N_4) is given by

$$SiH_4\,(gas) + 4\,NH_3 = Si_3N_4\,(solid\,) + 7H_2\,(gas)$$

The NH_3 reacted with silane produces Si_3N_4.

4. SiON Layer

The optical and physical properties of SiON are expected to be between that of SiO_2 and Si_3N_4. Therefore, monitoring of the compositional characteristics of the mentioned layers should result in the better control of the SiON film properties. A general trend of decrease in the film refractive index with increasing oxygen is observed. It is because of oxygen's greater chemical reactivity compared to the nitrogen.

From the following chemical reactions, it can be concluded that both N_2O and NH_3 are involved in controlling refractive index. For the deposition of SiON film, both N_2O and NH_3 are required. The basic reaction for the formation of SiON is given below:

$$SiH_4 + N_2O + NH_3 \rightarrow Si\,O_xN_yH_z\,(solid) + H_2O(gas) + N_2\,(gas)$$
$$\rightarrow Si\,O_xN_y\,(solid) + H_2O(gas) + HCl(gas)\ \ (after\ annealing)$$

From the above discussion, it is clear that SiON films are obtained by using silane, ammonia, and nitrous oxide as reactant gases. The different process parameters reported for the deposition of SiON layer is shown in Table 11.3. The flow rate of silane (2% SiH_4/N_2: diluted in nitrogen because of being highly unstable at room temperature and tendency to burn in case of exposure to air) is fixed at 180 sccm and the flow rate of N_2O and NH_3 are varied to get the desired refractive index of SiON. The layer is grown at 350°C with application of RF power of 10 watts. The applied RF frequency is 13.56 MHz.

Table 11.3 Process parameters of SiON film

Parameters	Values
Si-Substrate temperature	350°0C
RF power @ 13.56 MHz	10 W
Pressure	1000 mTorr
N_2O flow rate	20–450 sccm
SiH_4 flow rate	180 sccm
NH_3 flow rate	15–30 sccm

The film thickness values of the samples varies roughly between 4300°A and 3000°A. It is found that increase in N_2O flow rate results in an increase of film growth rate too. Moreover, the deposition rate is observed to be decreasing with an increase of the ammonia flow rate. These properties are attributed again to oxygen's greater affinity for reacting with silane gas. Namely, in the first case, as the reactive oxygen concentration increases, it begins to dominate in the chemical reactions over nitrogen. As for the decrease of the growth rate with increase in nitrogen concentration in the film, the probability of the nitrogen-related bonding now has increased so that nitrogen's concentration in the film increases. Thus, the layer becomes more similar to silicon nitride structured film and the growth rate is also smaller than that of silicon oxide films. In fact, if the growth rates for these films are considered, they increase in the following order: silicon nitride, silicon oxynitride, silicon oxide. So, a smooth transition of the physical properties of silicon oxynitride from those of silicon oxide to silicon nitride occurs.

The PECVD deposited SiON layer contains a certain amount of O-H bonds, N-H bonds, and Si-H bonds that are known to be main cause of optical absorption at 1.38 μm, 1.48 μm and 1.51 μm respectively. These bonds are removed by annealing at temperature ~ 900–1000°C.

11.7 □ FLAME HYDROLYSIS DEPOSITION (FHD)

The origin of this process lies in the optical fiber manufacturing and can produce a thick layer (~100 μm) of doped silica at high deposition rates. In addition, the deposition and consolidation process is intrinsically plane rising, hence providing excellent cladding uniformity over closely spaced cores. The process of FHD is shown in Fig. 11.17. In this process, a mixture of gases are burnt in O_2/ H_2 torch to produce fine particles, which stick on to substrate fixed on rotating table. $SiCl_4$, $TiCl_4$, and $GeCl_4$ are used to produce SiO_2 doped with TiO_2 or GeO_2 respectively. A small amount of Cl_2 and BCl_3 are added to lower the softening temperature of synthesized glass particles. After deposition, the heating to a temperature of around 1200°C to 1300°C consolidates the material on substrate. This process has

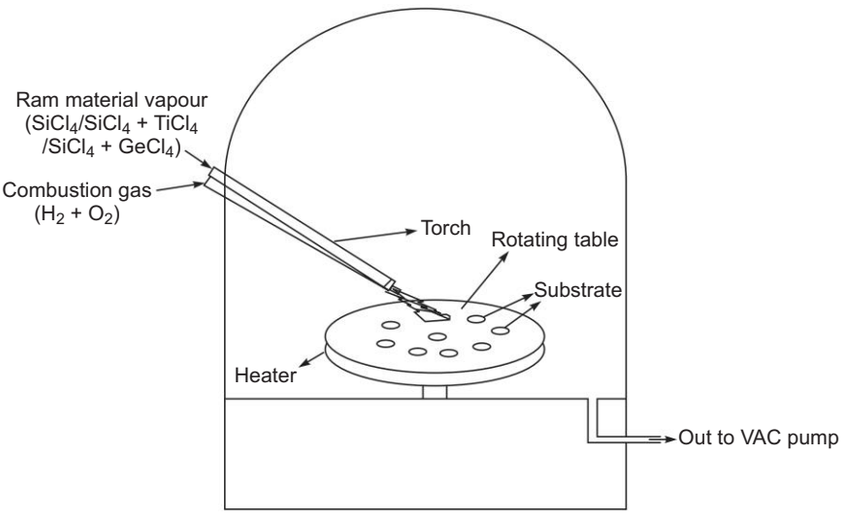


Fig. 11.17 Flame hydrolysis deposition

disadvantage of single side deposition only. The chemical reaction in FHD for deposition of SiO_2, $SiO_2 + GeO_2$ and $SiO_2 + TiO_2$ are given below:

SiO_2 deposition: $SiCl_4 + O_2 + 2H_2 = SiO_2 + 4HCl$
SiO_2 + GeO_2 deposition: $SiCl_4 + GeCl_4 + 2\ O_2 + 4H_2 = SiO_2 + GeO_2 + 8HCl$
SiO_2 + TiO_2 deposition: $SiCl_4 + TiCl_4 + 2\ O_2 + 4H_2 = SiO_2 + TiO_2 + 8HCl$

11.8 □ EPITAXY

Epitaxy is a Greek word consisting of—*epi* meaning upon and *taxy* meaning ordered. There are different ways of epitaxial processes—vapour-phase epitaxy, liquid-phase epitaxy, and molecular-beam epitaxy. These processes are used for the growth of crystalline layers. Epitaxial growth enhances the device performance. It provides a way to control the doping profile in a device structure beyond that available with diffusion and ion implantation.

11.8.1 Vapour Phase Epitaxy

Vapour Phase Epitaxial (VPE) growth is used mainly for fabrication in silicon device. The vapour deposition of a single-crystal silicon is performed in a reactor which consists of a quartz reaction chamber into which a susceptor is placed. The suceptor is used to give the physical support for the substrate wafers. VPE process is a high-temperature process at which deposition takes place after several chemical reactions. Figure 11.18 shows a pancake-shaped susceptor made of graphite. It provides uniform deposition of layers. The reaction tube/susceptor is relatively cool made by air cooling or water circulation. Energy of the reactor is supplied by the inductive coil. For silicon deposition, normally the reactant used is silicon tetrachloride ($SiCl_4$) but other reactant sources are dichlorosilane (SiH_2Cl_2), silane (SiH_4), etc. The reaction kinematics for source $SiCl_4$ and H_2 is given by

$$SiCl_4(g) + 2H_2\ (g) \rightarrow Si\ (s) + 4HCl\ (g)$$
$$SiCl_4(g) + Si \rightarrow SiCl_2\ (g)$$
$$SiCl_2(g) + H_2\ (g) \rightarrow Si\ (s) + 2HCl\ (g)$$

For growing *p*-type or *n*-type silicon layer, the dopants used are gaseous diborone (B_2H_6) and phosphine (PH_3) or arsine AsH_3 respectively. Gaseous mixtures are used with hydrogen as diluent to control the flow rates to obtain the desired amount of doping concentration.

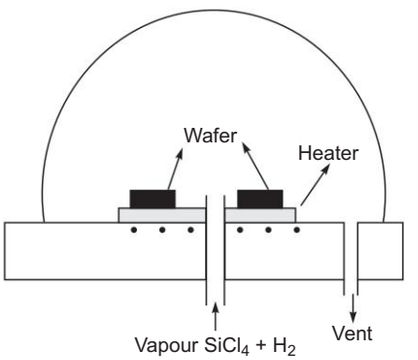


Fig. 11.18 Pancake-based VPE susceptor

$$B_2H_6(g) \rightarrow 3H_2\,(g) + B\,(s)$$
$$2AsH_3(g) \rightarrow 3H_2\,(g) + 2As\,(s)$$

11.8.2 Liquid-Phase Epitaxy

The growth of epitaxial layers on the substrate is possible by using direct precipitation from liquid phase. This process is called Liquid Phase Epitaxy (LPE). This technique is normally used for growing gallium arsenide and other III–V compounds. It is also useful in growing very thin epitaxial layers but growth rate is very small. The boat (made of pure graphite) containing reactant solution is put in wells in the furnace and graphite slider holding is moved to locate under wells at required temperature. Before completion of deposition, the temperature is lowered and the wafer is moved out from under solution. With this process, multilayer growth is also possible by moving the wafer to different wells.

11.8.3 Molecular Beam Epitaxial (MBE) Growth

MBE is an epitaxial process involving the reaction of one or more thermal beams of atoms or molecules with crystalline surface under ultra high vacuum condition (10^{-8} Pa). MBE can achieve precise control in both chemical compositions and doping profiles. Single-crystal multilayer structures with dimensions of the order of atomic layers can be made using MBE. It is a precise fabrication of semiconductor heterostructures having thin layers from a fraction of micron down to a nanolayer. Normally, MBE is used for growing a crystalline layer of III–IV group compounds such GaAsInP on the substrate. In general, MBE growth rates are quite low and for GaAs, a value of 1μm/h is typical. The system has film deposition control, cleanliness and *in- situ* chemical characterization. Figure 11.19 shows the schematic diagram of a conventional molecular beam epitaxy system chamber consisting of arrangement of the effusion ovens (source) and substrate.

The separate effusion ovens made up of pyrolytic boron nitride are housed for each constituent of the layer and dopants. The temperature of each oven is adjusted to give the desired evaporation rate.

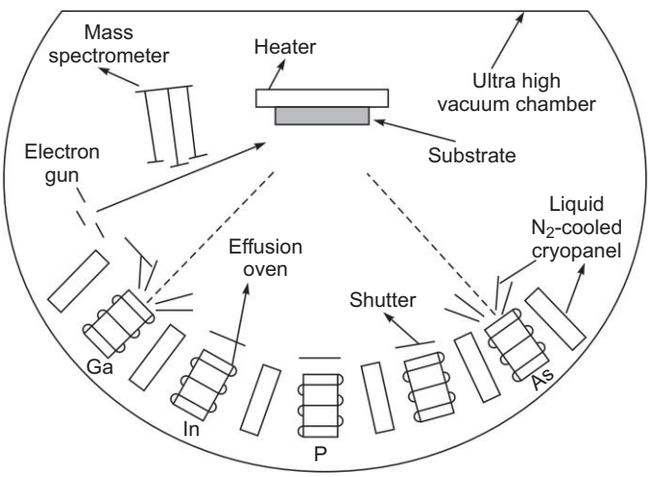


Fig. 11.19 MBE chamber with arrangement of the sources and substrate

Main Parts of MBE System

(a) Vaccum Creating System For MBE process, ultra high vacuum is created because system must be free from any type of contaminants and unwanted gas molecules to achieve high purity and precision. Since the operating ranges of different pumps differ to each other, here no single pump creates the degree of vacuum, which is needed. Therefore, different pumps are used in various combinations as given in Table 11.4.

(b) Effusion Cells Eight effusion cells constructed with tantalum and PBN material are provided—out of these, two cells have 14 cc capacity, three cells have 40 cc capacity and one cell has 125 cc capacity. The maximum outgassing temperature is 1600°C and maximum operating temperature is 1400°C.

Table 11.4 Different vacuum pumps used in MBE system

Types of Vacuum	Range	Pumps
Rough vacuum	760 Torr to 1×10^{-3} Torr	Rotatory vane oil sealed mechanical pumps and dry vacuum pumps
High vacuum	1×10^{-3} Torr to 1×10^{-8} Torr	Oil diffusion pumps, mechanical cryo-pumps, and turbo molecular pumps
Ultra high vacuum	Less than 1×10^{-8} Torr	Ion pumps, and titanium sublimation pumps

(c) Flux Monitor A node ionization gauge is provided to measure the flux of various atomic beams. The flux monitor is mounted on a bellow which enables pushing of the gauge right in front of the wafer during flux measurement.

(d) Residual Gas Analyzer A quardrupole mass spectrometer with 1–200 amu is provided for residual gas analysis. The minimum detectable partial pressure is 2×10^{-13} Torr. The spectrometer is controlled by a microcomputer which can control four such spectrometers.

(e) Residual Gas Analysis (Quadrupole Mass Spectrometer) QMS analyses residual gases by separating each of them, depending on their mass, to charge ratio and measure their relative abundance.

(f) Analysis Chamber This chamber is used for the analysis of the surface of grown layers or the substrates. This chamber consists of the followings—ion gun is provided to etch an area of 1 cm^2 to remove surface contaminant such as carbon and also to acquire the depth profile of the material. This system allows the elemental analysis of the grown crystal and observation of contamination level on surface of the substrate.

(g) Substrate Manipulator This consists of X, Y, Z and rotary motion. The manipulator can handle a wafer up to a 3-inch diameter. The major requirement of the substrate holder for the production of uniform layers is that the substrate manipulator is rotated by a stepper motor at a slow rate of

3–5 rpm. The general construction of such an assembly is of a refractory metal block (usually Mo), which is heated either resistively or by radiation. The substrate is usually held on the Mo block. A Mo holder holds the wafer with the help of a few Ta pins. The heating of GaAs substrate mounted on a holder must be done directly by thermal radiation from the heater. The GaAs wafer is suspended by a few thin Ta or Mo wires onto the Mo gasket like a holder, on which the rear surface of the wafer can receive the direct radiation from the heater. The Mo holder can be transferred and mounted on to the heater assembly just as the conventional Mo holder. Since the backside of the GaAs wafer mounted on the Mo block receives the radiation from the heater, the top portion of the back surface of GaAs can decompose particularly at high temperatures.

This can be minimized if the back surface of the GaAs facing the heater is surrounded by enclosure housing, since the excess pressure of As generated by the dissociation of the GaAs is maintained inside the housing and prevents the further decomposition of Ga. The thermocouple or a pyrometer usually measures the substrate temperature. To achieve high uniformity of film thickness and alloy composition, in most MBE systems, the substrate holder is equipped with a rotation mechanism. Effusion cells having an operating temperature range up to 1400°C makes possible short outgassing sequence at 1600°C. Although these temperatures are more than adequate for III–V materials, in practice, more cells are limited to operating temperatures ≤1200°C, which is only just within the range of that required for Ga, Al, In and P(dopant source) evaporation. This is because of outgassing problem caused principally by the large temperature difference between the heater and the actual temperature of the charge in the cell. A means of rapidly opening and closing line of sight between the beam sources and substrate must be available for each beam source. These shutters must then meet rigorous mechanical design criteria in order to satisfy the performance requirements necessary to fully exploit the precise layer thickness defined in MBE.

After the group V flux is set about 10–20 times greater than the group III flux, the thermal adsorption of the passivating oxide may commence. The growth is initiated by exposing the clean heated sample to the Group III and V beams via opening of the shutters. Dopants are incorporated similarly by the shutter actuation of the dopant source. The calibration of the dopant concentration as a function of the source temperature may be done empirically.

The presence of the charge on the crucible may cause the closing of an externally controlled shutter over the mouth of the crucible. Initially, the thermal equilibrium is obtained prior to the growth with the shutter closed. However, the initiation of the growth by opening the shutter introduces a temperature perturbation onto the charge because the effusion furnace radiates energy out of an open instead of a close end. Typically, the exposed surface of the charge retains thermal equilibrium within a minute. This shutter-induced thermal transient affects directly the amounts of the material flux from the charge. In the superlattice structures, where some layers require less than one minute for the deposition, the thermal transients are obstacles to rowing the constant composition of the Group III metals within single superlattice period.

The thermal transients have been reduced by increasing the separation between the mouth of the crucible and the shutter or varying the angle of the shutter with the plane of the crucible opening. Compositional control in MBE is a function of flux ratios, substrate temperature and chemical species, since all these factors affect the surface incorporation kinematics on a substrate. In many cases, the fluxes are controlled via the source temperature to grow to the desired composition. For that, the beam flux monitoring prior to the growth is required. In case of III–V alloy growth, the Group III

species have higher sticking coefficient at the substrate temperature below 600°C. The beam flux of the Group III species is proportional to the desired atomic composition (if the instrument is corrected for the instrument-induced error).

The stoichiometric control of III_x III_{1-x} VV alloys result in control of the band gap, refractive index, etc. As for an example, $In_xGa_{1-x}AsP$ of band gap from 1.28 eV to 1.43 eV is grown by choosing x from zero to one. In case of InP/GaAsInP waveguide, the InP layer acts as a substrate and GaAsInP acts as a guiding layer. The InP substrate is placed on the holder. The temperature of the substrate is kept at ~ 580°C. In Fig. 11.19, Ga, In, As, and, P are placed in the respective oven for the growth of GaAsInP.

11.9 ◻ LITHOGRAPHY

The word 'lithography' is a Greek word, which literally, means *stone writing*. Actually, lithography is the process of transferring the pattern using an optical image and photosensitive film on substrate. There are different types of lithography—electron-beam lithography, ion-beam lithography, X-ray lithography, and photolithography. Table 11.5 shows different chemicals used as positive and negative photoresist for different lithographic processes.

Table 11.5 Positive and negative photoresist for different lithographic processes

Lithgraphy process	Positive resist	Negative resist
E-beam	PBS (Poly-Butene-1-sulfone) PMMA(poly-methyle-metarylate)	COP GeSe
Optical or photolithography	AZ-1350J PR102 Kodak-1813	Kodak-747 Kodak KTFR
X-ray	PBS	DCOPA

11.9.1 Electron Beam Lithography

Electron beam lithography provides a means to reduce image resolution below 100 nm using accelerated electrons instead of photons. Like photons, electrons also have wave nature. Figure 11.20 (a) shows the schematic diagram of electron beam lithography. When electrons are accelerated through a potential V, wavelength is given by

$$\lambda = \frac{1.23}{V^{1/2}} \quad \text{where } V \text{ is in keV and } \lambda \text{ is in nm.}$$

When $V = 10$ keV, $\lambda = 0.0123$ nm which is much smaller than the wavelength of UV light. The electron-beam lithography pattern can be transferred directly on the wafer without mask. But there are drawbacks of e-beam lithographic systems. Mainly, it needs vacuum and slow exposure speed. Collision of electrons with the substrate causes random scattering and back-scattering electron generation and also secondary electrons. The scattered electrons create a proximity effect in which it exposes up to several micrometers from point of impact. It is specially used for the fabrication of the mask for

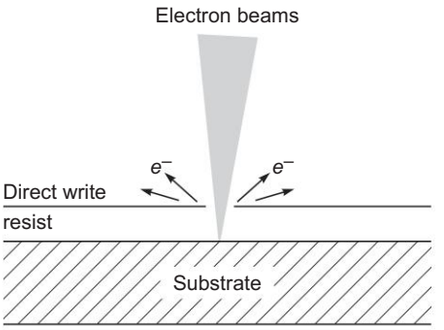


Fig. 11.20 (a) Electron-beam photolithography

high-resolution patterning of special devices (GaAs) SOI devices, etc. Moreover, this system is costly. Table 11.5 shows positive and negative photoresist for electron lithography.

11.9.2 X-ray Lithography

X-ray lithography is better than optical lithography because very large Depth Of Focus (DOF) can be achieved with shorter wavelength. Figure 11.20 (b) shows the X-ray lithography process. Since, the X-rays can pass through many solids including dust and skin flakes, a clean room is necessary for chip fabrication with specifications much less than that of UV lithography. This results in greater tolerances in dust particles. The wavelength of X-rays is order of 0.1–10 A°. The proximity mask is necessary to reduce the diffraction which increases the lifetime of mask pattern. The typical line width of 0.25– 0.15 μm can be resolved using X-ray lithography with a proximity mask. The main advantages of X-ray lithography are the fragility and dimensional instability of the mask and the complex alignment system. Table 11.5 shows positive and negative photoresist for X-ray lithography.

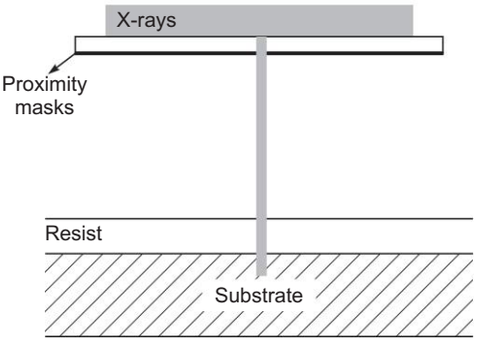


Fig. 11.20(b) X-ray lithography

11.9.3 Ion-beam Lithography

Figure 11.20(c) shows the ion-beam lithography process [6]. The ion beam consists of H^+, Ar^+ or He^+. These ion beams can be used to expose resists either through a mask or through broad beams by writing directly on the resist with a fine focussed beam. Since the ions are heavier than electrons, they

transfer their energies more efficiently to resist and scatter much less than 10 nm. Ion beam source also produces secondary electrons. This make a high resolution with ion-beam lithography without using proximity masks. The radiated beam size is 1–2 cm^2 broad. In an ion-beam, the diffraction effect is negligible. Resolution up to 100 nm can be achieved with it. There are two types of sources, which are used in ion-beam lithography:

1. Liquid metal source
2. Gas-field ionization source

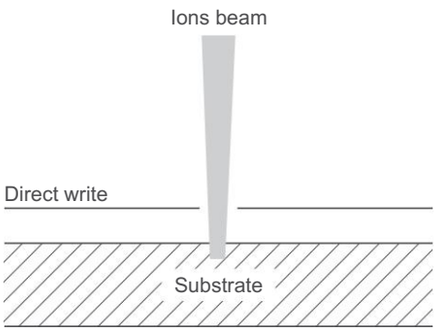


Fig. 11.20(c) Ion-beam lithography

11.9.4 Photolithography

Figure 11.20(d) shows a photolithography set-up consisting of a mask with a device pattern, UV light, etc. In photolithography, a film of the photoresist is first applied to the wafer. Table 11.5 shows common positive and negative photoresists used for photolithography. The radiation is shone through a transparent mask plate. The mask plate contains opaque or transparent regions according to the pattern of the devices. The resulting image is focussed onto the resist-coated wafer producing areas of light and shadow corresponding to the image on the mask plate. In those regions, where light was transmitted through the plate, the resist solubility is altered by photochemical reaction. In the shadowed regions, its solubility remains unaffected. This step, by analogy to photography, is termed exposure. Following exposure, the substrate is washed with a solvent that preferentially removes the resist areas of higher solubility. This step is called development.

Depending on the type of the resist, the washed-away areas may be either the illuminated or the shadowed regions of the coating. If the solubility of the resist is increased with the exposure, the resist is washed away in the areas corresponding to the transparent zones of the mask plate. The resist image is identical to the opaque image on the plate and the pattern is photographic positive. Therefore, resist is called positive resist. A resist that loses solubility when illuminated with radiation, forms negative images of the plates and is called negative resist. Before going to photolithography, the following steps are followed:

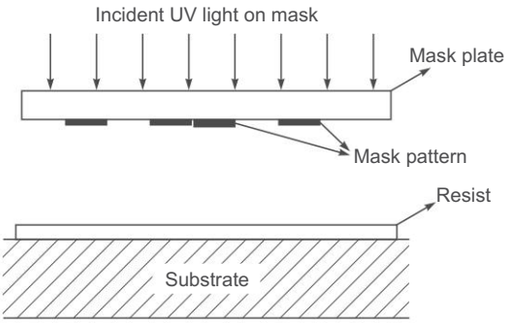


Fig. 11.20(d) Photolithography set-up

1. Masks Fabrication

The stencil used to generate a desired pattern in resist-coated wafers repeatedly is called mask. In use, a photomask, a nearly optically flat glass (transparent to near UV) or quartz plate (transparent to deep UV) with a metal (e.g., 800 Å thick chromium) absorber pattern is placed into direct contact with the photoresist coated surface, and the wafer is exposed to the ultraviolet radiation. The absorber pattern on the photomasks is opaque to UV radiation whereas glass/quartz is transparent.

A light field image or dark field image (shown in Fig. 11.21) is then transferred to the wafer surface. This procedure results in a 1:1 image of the entire mask on to the wafer. The described masks making direct physical contact to the substrate is called contact masks. Unfortunately, these masks degrade faster through wear than do non-contact, proximity masks (also referred as **soft contact**). The defects resulting from hard contact masks make it unsuitable for fabrication of device.

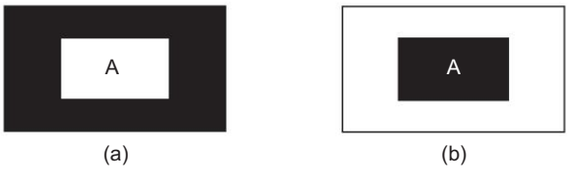


Fig. 11.21 Types of masks: (a) Dark field mask (b) Bright field mask

2. Spinning Resist and Soft Baking

A photosensitive material experiences a change in its physical properties when exposed to a radiation source. If we selectively expose a photosensitive material to radiation, the pattern on the material is transferred to the material exposed because the properties of the exposed and unexposed regions differ. The photoresist is dispensed from a viscous solution of polymer on to the wafer, laying a wafer plate in the resist spinner. The wafer is spun at a high speed between 1500 to 8000 rpm depending on viscosity and the required film thickness to make a uniform film. At these speeds, centrifrugal force causes the solution to flow to the edges where it builds up until expelled when the surface tension is exceeded.

The resulting polymer thickness (T) is a function of spin speed (w), polymer-solution concentration (C in grams/100 ml solution), and viscosity (η). The empirical expression for T is given by

$$T = \frac{K.C^{\beta}\eta^{\chi}}{w^{\alpha}}$$

where K = Overall calibration constant. The α, β, and χ are the constants of spinning process which are determined from known thickness. After spin coating, the resist still contains up to 15% of the solvent and may contain build in stresses. Figure 11.22(a) shows a spin-coating machine with operation. The

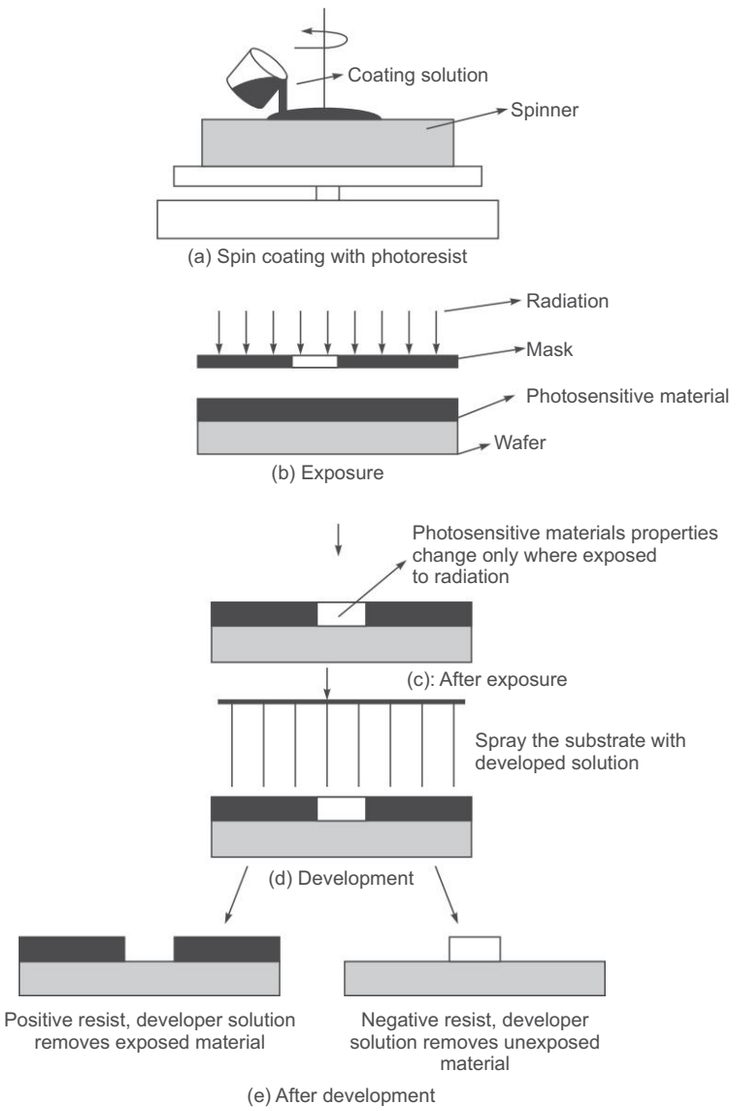


Fig. 11.22 Steps of photolithography

coated wafers are then soft baked at 95°C for 20–25 minuters to remove solvent and stresses, and promote adhesion of the resist layer to the wafer.

3. Mask Alignment

One of the important steps in photolithography process is mask-alignment. The mask alignment arrangement is shown in Fig. 11.23. A mask or photomask is a square glass plate with patterned emulsion of the metal film on one side. The mask is aligned with the wafer, so that the pattern can be transferred onto the wafer surface. Each mask after the first one must be aligned to the previous pattern.

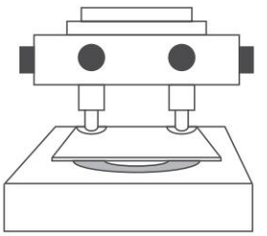


Fig. 11.23 Basic set-up for lithography process with alignment optics

4. Exposure

The exposure parameter is required to optimize in order to achieve accurate pattern transfer from the mask to the photosensitive layer. Figure 11.22(d) shows the arrangement of exposure of UV light on the wafer side containing a photoresist. It depends primarily on the wavelength of the radiation source. The dose of the radiation should be sufficient to get the desired change in the properties of the photoresist produced. Different photoresists exhibit different sensitivities to the different wavelengths. The dose required per unit volume of photoresist from good pattern transfer is somewhat constant. The exposure process also depends on the layer of wafer under the photoresist. For an example, a highly reflective layer under photoresist may need a higher dose than that for the lightly reflective underlying layer. The dose will also vary with the resist thickness. There are also higher-order effects such as interference patterns in thick resist films on a reflective wafer, which may affect the pattern quality and side-wall properties.

At the edges of the pattern, light is scattered and diffracted. If an image is overexposed, the dose received by photoresist at the edge that should not be exposed may become significant. If we are using a positive photo-resist, this will result in the photoresist image being eroded along the edges, resulting in a decrease in feature size and loss of sharpness or corners, as shown in Fig. 11.24. If we are using a negative resist, the photoresist image is dilated, causing the features to be larger than desired. Again, accompanied by a loss of sharpness of the corners. If an image is under-exposed, the pattern may not be transferred at all. Figure 11.24 shows the desired pattern, under exposed pattern, and overexposed transferred pattern on wafer.

The incident light intensity (in W/cm^2) multiplied by the exposure time (in seconds) gives the incident energy (J/cm^2) across the surface of the resist film. The exposure time depends upon the quality of the mask, atmospheric condition, wafers material, etc. The optimization of the exposure time is crucial

in a hole-fabrication process. The quality of lithography depends on the exposure time and it requires very good accuracy specially for fabrication of low-dimension waveguide core.

5. Development

Development transforms the latent resist image formed during exposure as a relief image, which serves as a mask for further subtractive and additive steps. During the development process, the resist dissolves selectively. Two main technologies are available for development—wet development and dry development. Wet development by solvents can be based on three different types of exposure induced changes—variation in molecular weight of the polymers (by cross linking or by chain session), reactivity change, and polarity change. Two main types of wet development set-ups are used—immersion and spray developers. During immersion developing, cassette-loaded wafers are batch-immersed for time period in a bath of developer and agitated at a specific temperature. During spray development [as shown in Fig. 11.22 (d)], fresh developing solution is directed across wafer surfaces by fan-type sprayers. After development, the wafer sample is shown in Fig. 11.22(e).

In wet development, the use of solvents leads to some swelling of the resist and loss of adhesion of the resist to the substrate. These problems can be removed by dry development, as it is based on vapor-phase process or plasma process. In case of plasma dry development oxygen, ion etching(O_2-RIE) is used to develop the latent image. The latent images formed during exposure exhibit a differential etch rate to O_2-RIE rather than differential solubility to a solvent.

Figure 11.24 shows the structure for desired exposed pattern, under-exposed pattern and overexposed pattern after photolithography.

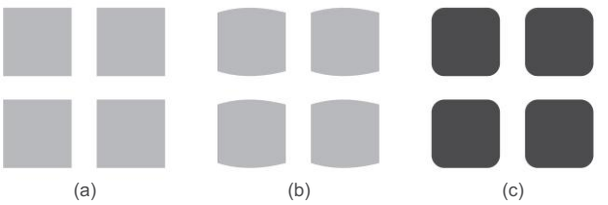


Fig. 11.24 (a) Desired pattern (b) Overexposed (c) Under-exposed pattern

6. Post-Baking

Before etching the substrate or adding material, the wafer needs post-baking. Post-baking or hard baking removes residual development solvents and anneals the thin film to promote interfacial adhesion of the resist weakened by the developer penetration. Hard baking also improves the hardness of the film. The post-bake frequently occurs at higher temperatures (105–120°C) and for longer times (30–40 min) than the pre-baking step.

7. Ashering

A mild oxygen plasma treatment, also known as **ashering**, removes unwanted resist left behind after development. Resists leave thin polymer films at the resist wafer interface. The time duration of

ashering depends on thickness of the resist after developing. Generally, 2–5 minutes is enough for ashering. It can remove resist thickness in the range of 100–500 Å, depending on RF power and time for flow of O_2.

11.10 □ METALLIZATION

In IC processing, metallization is one of the important steps in which proper connection of electrodes and protection of the layers is made. Aluminum (Al), nickel (Ni), chromium (Cr), and nichrome (Ni-Cr) are some of the popular metals for making ohmic contact to the devices and connecting these to the bonding pads as well as a thin layer for selective etching. The adhesive properties of aluminum and nichrome are very good with silicon and silicon dioxide, and they can be deposited easily (since they have low boiling point). Different metals are used for different function-related reasons. For example, nichrome is used to make the resistor in IC and also for protection layer to get selective etching. Some of the important properties of metallization process for better throughput of fabrication process are as follows:

1. Easy to form
2. Easy to etch for pattern generation
3. Stable in oxidizing ambient; oxidizable
4. Mechanically stable; good adherence and low stress
5. Surface smoothness
6. Stable throughout processing including high-temperture sintering, dry or wet oxidation etc
7. No reaction with the final material
8. Protection from contamination
9. Increase of lifetime
10. Low contact resistance, low electromigration, and minimal junction penetration.

Metal layers are deposited in vacuum onto wafers by one of the following methods [7]:

1. Filament evaporation
2. Flash evaporation
3. Electron-beam evaporation
4. Sputtering.

11.10.1 Evaporation

The evaporation process can be divided into three steps:

1. The solid metal must be changed into gaseous vapor (sublimation).
2. The gaseous metal must be transported to the substrate.
3. The gaseous metal must condense onto the substrate.

This process requires high temperature and low pressure. The chamber of the Denton vaccuum must be pumped down in order to achieve the low-pressure requirement. The pressure inside the chamber is determined by the equation below:

$$P = P_0 . e^{-(\text{S. T/V})} + Q/S \tag{11.17}$$

where P is the chamber pressure at time t, P_0 is the initial pressure, S is the pumping speed, Q is the rate of outgassing, and V is the volume of the chamber.

The concentration of gaseous metal molecules in the vaccuum chamber can be determined using

$$N = N_{av}/V = KT$$

The following quantities can be calculated:

1. The impingement rate of the gaseous metal hitting to the substrate
2. The time required for one mono-layer of metal to deposit on the wafer surface
3. An approximate final thickness of the metal on the substrate

The impingement rate is given by

$$\phi = \frac{P}{(2\pi mkT)^{1/2}} = 2.6 \times 10^{20}.P/(MT)^{1/2} \tag{11.18}$$

The time required for one monolayer of the metal to be deposited is given by

$$t = N_s/\phi$$

The final thickness of the metal is given by

$$T_h = \phi \Delta m/\rho A$$

where ρ = Density of the metal, A = Area of deposition and Δm = Mass of the metal to be deposited.

The evaporation process does not produce a uniform layer of Al across the substrate. The deposition rate changes as one moves radically from the center of the substrate. The radial dependence of the deposition rate is described by the following equation:

$$D = D_0/[1 + (R/H)^2]^{3/2} \tag{11.19}$$

1. Filament Evaporation

Filament evaporation is accomplished by gradually heating a filament of the metal to be evaporated. Figure 11.25(a) shows the schematic of filament evaporation method. This metal may come in one of the several different forms—pellets, wire, crystal, etc. Ag, pt, Al, Sn, Zn, Cd, Ni, and Ni are mostly used for filaments evaporation to be deposited on wafer. The metal is placed in a molybdenum filament basket. Electrodes are connected to either side of the basket to heat and high current is passed through it, causing the basket to heat. As the heat is increased, the metallic filament partially melts and then is eventually vaporized. In this way, atoms of the metal break free from the filament and deposit on to the wafers. It is the simplest way of metal deposition. The problem of this system is contamination during evaporation.

2. Flash Evaporation

Flash evaporation uses the principle of thermal-resistive heating to evaporate metals. Like filament evaporation, flash evaporation offers radiation-free coatings. This technique offers some benefits over filament evaporation—contamination-free coatings, speed or good throughput of wafers, and ability to coat the composite materials or layers. Flash evaporation is accomplished by passing a continuous supply of the material to be evaporated over ceramic structure that provides heat. The ceramic flash bar is heated between two positively and negatively charged posts and metal evaporates as the source material is fed on to the bar.

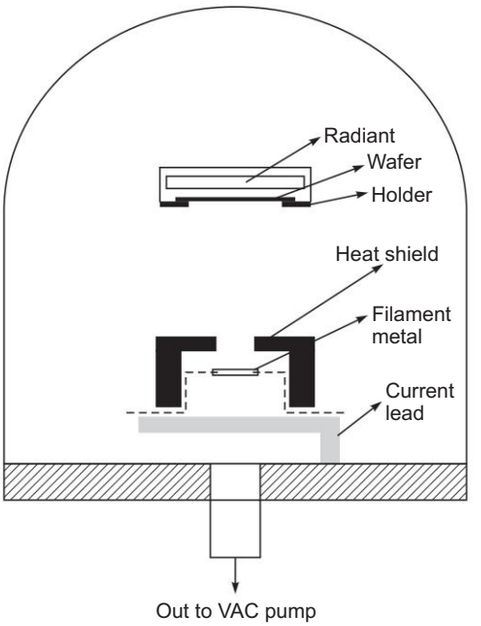


Fig. 11.25(a) Filament evaporation

3. Electron-Beam Evaporation

Figure 11.25 (b) shows the schematic of electron-beam evaporation. Electron-beam evaporation works by focusing an intense beam of electrons into a crucible or pocket, in the evaporator that contains Al, and Ni-Cr. As the beam is directed into the source area using electric and magnetic fields, the metal is heated to its melting point and eventually to evaporation temperature. The benefits of this technique are speed and low contamination, since the electron beam touches the source material.

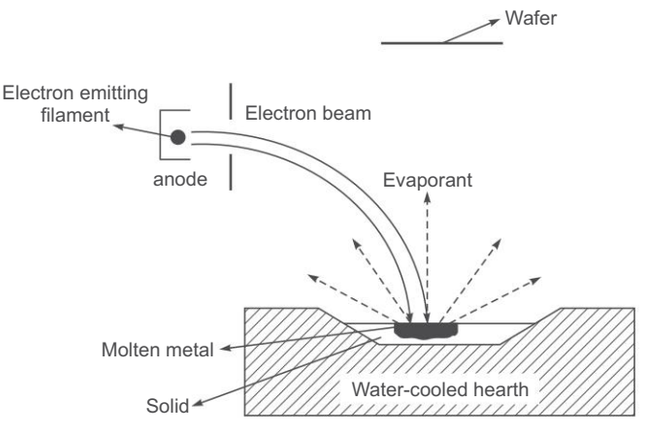


Fig. 11.25(b) Electron-beam deposition set-up

4. Sputtering

Sputtering is one of the commonly used metallization processes. It is popular because the adhesion of the deposited material is excellent. RF sputtering is the process of ionizing the inert gas particles such Ar + in an electric field (producing gas plasma). They are, then, accelerated towards the source or target through a potential gradient and the bombardment of these ions on target takes place. The energy of these gas particles physically dislodges via transfer of momentum by which atoms near the surface of the target material become volatile and sputters off the atoms of the source material. So sputtering is a versatile tool by which many materials can be deposited, using not only RF but also dc power. Figure 11.26 shows a sputtering set-up for deposition of metal. Electrons originated at the cathode are confined by the permanent magnets and collected by the anode. The magnetron operates at voltages in the order values less than those of beam sources. To obtain uniform film thickness, appropriate mechanical motion of the substrate can be employed to expose the substrate to the same average number of sputtered atoms. In this method, aluminum, silver, and gold are used for deposition. The system used for cathode sputtering is almost identical to that used for vacuum evaporation. The process, however, is much slower than evaporation, depositing a micron-thick film in minutes to hours compared to seconds to minutes for evaporation. However, sputtering is superior to vacuum evaporation regarding the quality of the film produced but sputtering equipment is more costly than most evaporation systems. The process of cathode sputtering is performed at a low pressure (about 10^{-12} Torr). The source material (material to be sputtered) is subjected to intense bombardment by the ions of a heavy inert gas such as argon.

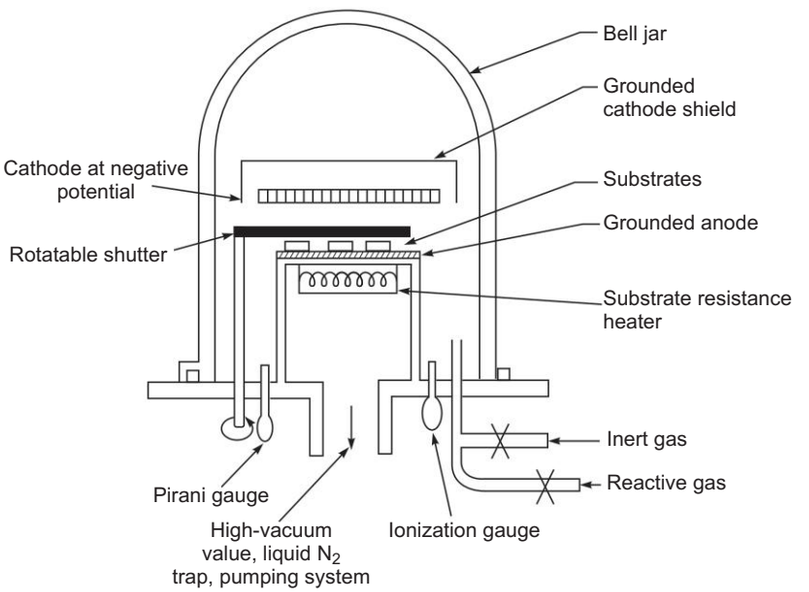


Fig. 11.26 Sputter system for deposition

These gas ions are usually accelerated by making the source material as the cathode of a dc glow discharge. As atoms are ejected from the surface of the cathode, they diffuse away from it through a low-pressure gas, depositing as a thin film on a nearby substrate. This sounds crude, but the high energy

of the particles landing on the substrate actually results in a very uniform film with good crystal structure and adhesion. A potential—typically 2 to 5 kV—is applied between the cathode (source material) and anode, and produces a glow discharge that fills the entire interelectrode space, except for a thin region close to the cathode.

11.10.2 Plating Technique

The two types of plating techniques used are (i) electroplating, and (ii) electroless plating. Electroplating uses the typical process of coating an object with one or more layers of different metals. Cathode and anode, in this case substrate and metal, are immersed in an electrolytic solution. When dc current is passed through the solution, the positive metal ions migrate from the anode and deposit on the cathode. This method is suitable for making conduction films of gold or copper.

In electroless deposition of thin films, a metal ion in solution is reduced to the free metal and deposited as a metallic coating without the use of an electric current. This process can be used to deposit metals on any substrate (such as glass, ceramic, and plastic) and films of considerable thickness can be deposited.

11.11 ◻ ETCHING

It is a process to remove material either selectively or non-selectively. Different methods are available for etching. The most commonly used etching methods are dry etching and wet etching/chemical etching. Wet etching is mainly used for cleaning, shaping, and polishing whereas dry etching is used for micromachining/precision machining. The advantages of dry etching are higher aspect ratio, high directionality, less corrosion problems for metal features in the structure, less undercutting, and broadening of photoresist features. The problem with dry etching is low etch rate and high sensitivity to operating parameters in comparison to wet etching. So, it consumes more time and is difficult to balance. Etching may be isotropic or anisotropic depending on the mechanism of etching. In case of isotropic etching, the etching rate in crystallographic directions is same whereas in the anisotropy case, the etching rate is different for different crystallographic directions.

Etch performance is being judged in terms of selectivity, uniformity, surface quality reproducibility, post-etch corrosion, and profile control. Good selectivity, uniformity, and profile control are achieved at lower etch rate. In case of silica waveguide formation, deep etching (1.5 to 10 μm) is required.

11.11.1 Dry Etching

Dry etching covers a family of methods by which a solid-state surface is etched in the gas phase, physically by ion bombardment, chemically by chemical reaction with reactive species at the surface or combined physical and chemical mechanisms. There are different types of dry etching—plasma etching, Reactive Ion Etching (RIE), Reactive Ion-Beam Etching (RIBE)[6], sputter etching[6], ion milling, barrel etching, and magnetically enhanced etching. Table 11.6 shows the comparison between different dry-etching processes.

Table 11.6 Comparison between etching processes

Process	RIE	Plasma etching	RIBE	Barrel etching	Ion beam Etching
Pressure in torr	~10^{-3}–10^{-1}	~10^{-1} –10^{1}	~10^{-4}	~10^{-1} –10^{0}	~10^{-4}
Etch mechanism	Chem./Phys.	Chem.	Chem./Phys.	Chem.	Phys.
Selectivity	Good	Good	Good	good	poor
Profile	Anis/iso	Anis/iso	Anis	Iso	Anis

Anis—anisotropic, iso—isotropic, Phys.—Physical, Chem.—chemical.

1. Plasma Etching

Figure 11.27 shows the schematic of plasma-etching process. The RF frequency is generally 13.56 MHz and RF power is kept in between 1–2 kW. In the plasma discharge tube, CF_4-O_2 plasmas are reacted with SiO_2 wafer. The SiO_2 are etched out through SiF_4 volatile and O_2. The pressure is kept at 10^{-1} to 1 torr. The selectivity of etching is good because it is isotropic in nature. The chemical reaction is as follows:

$$CF_4 + SiO_2 = Si\,F_4 + CO_2$$

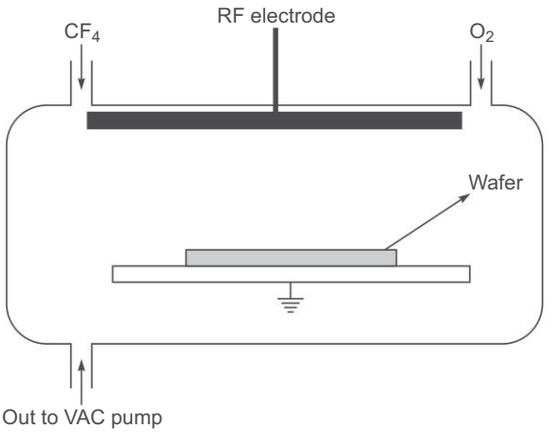


Fig. 11.27 Plasma etching

2. Reactive Ion-Beam Etching

Figure 11.28 shows the schematic of reactive ion-beam etching [6]. It is similar to reactive ion etching (RIE) (discussed later on) except that the wafers are separated from plasmas by a grid that accelerates ions (created in the plasma) towards the wafer. The ion energy is generally higher than RIE, perhaps over 1 keV, which accelerates the ion beam by using a focusing coil. Here, only physical etch mechanism occurs for etching. The pressure is kept at 10^{-4} torr.

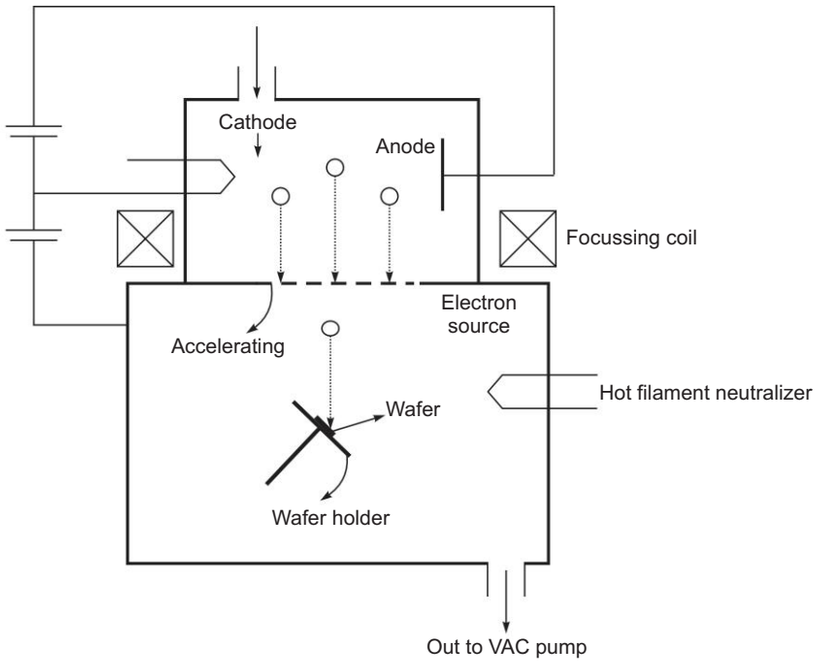


Fig. 11.28 Ion-beam etching apparatus

3. *Sputter Etching*

It is a purely mechanical phenomena. Figure 11.29 shows the schematic diagram for sputtering ion etching. In this process, energetic ions from plasma strike the substrate and physically blast the atom away from the wafer. In this process, there is no chemical reaction. Normally, the inert gases are used in this process. The inert gas ion is produced through ionization by plasma electrons. As for example, argon ions (Ar^{++}) are produced by simple ionization with electron impact:

$$Ar + e^- = Ar^{++} + 2e^-$$

It is also called **ion-beam etching**.

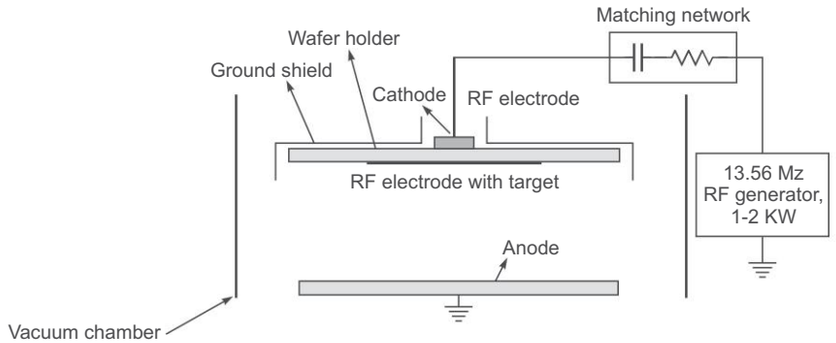


Fig. 11.29 Sputtering etching apparatus

4. Ion Milling

It is also a mechanical phenomenon. However, the energetic ions are used to create the surface by bombardment. It is much more like a particle accelerator. The ions are generated, roughly collimated and impinge on a selected area of wafer placed some distance away from RF circuitry. The ions are inert gas ions. Table 11.7 shows target materials and their etching rates.

Table 11.7 Target of ion mill and etching rates

Target material	Etchant abrev.	Etchant	Etching rate
Silicon	Ion mill	Argon ions at 500 V ~ 1 mA/cm^2, normal incidence	~38 nm/min
Aluminium	Ion mill	Argon ions at 500 V ~ 1 mA/cm^2, normal incidence	~73 nm/min
Chromium	Ion mill	Argon ions at 500 V ~ 1 mA/cm^2, normal incidence	~58 nm/min
Nicrome	Ion mill	Argon ions at 500 V ~ 1 mA/cm^2, normal incidence	~18 nm/min

Ion milling is also unique in that the angle of incidence of the ions is selectable by adjusting the angle between the wafer and ion beam. Adjusting the accelerated voltage controls ion energy, i.e, etch rate.

5. Reactive Ion-Etching

Reactive ion etching is most commonly used dry etching in which both chemical and physical erosion occurs. So, it is a chemical-physical process. It has gained a separate identity because of its own characteristics:

1. high directionality
2. kinetically assisted chemical etching

Figure 11.30 shows the schematic diagram for Reactive Ion Etching (RIE). The reactive ion species generated from plasma are used for etching. These ions have both a physical component (ion impact) and a chemical component (reactive etching). The impinging ions are bombarded on wafer and erode the surface by sputtering etch through the momentum transfer and chemical reaction. The chemical reaction takes place at low pressure of 10^{-4} to 10^{-1} torr. Table 11.8 presents different reactive ion etchants with etching targets and rates.

Here, we use RF power which is fed into the cathode (substrate) of the parallel planar electrodes to produce reactive gas plasma at low pressure. The cathode is self-biased at the negative potential, and the electric field yielded in the vicinity of the sample surface accelerates the positive ions in the plasma towards the sample. The chemical and physical mechanisms are therefore, incorporated in the etching. The RIE offers relatively large selectivity and anisotropy and is widely used for OIC fabrications. The reactant gases, $CHF_3 + O_2$ are used for SiON compound etching. Various reactive gases and additional H_2 and O_2 are used to improve selectivity and/or enhance anisotropy through suppression of the chemically reactive radicals.

The system is optimized during dummy wafer. The Ar plasmas are used for etching. Table 11.8 shows the process parameters for RIE. The flow rate of Ar and CHF_3 are 2.5 sccm and 5 sccm respectively. The vacuum is in the order of $\sim 5 \times 10^{-5}$ torr. The applied RF power of 13.56 MHz frequency is 215 W. The reflected RF power is 15 W with negative self-bias varied between 540–560 V.

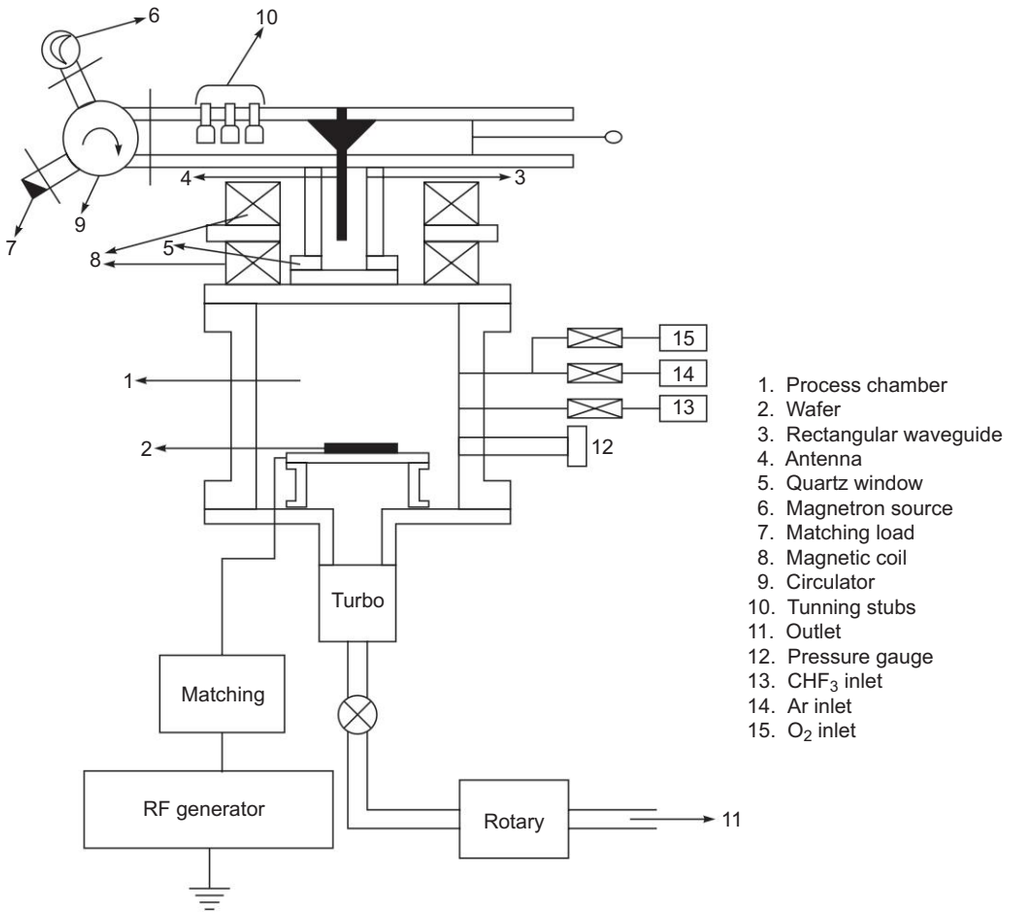


Fig. 11.30 Reactive ion-etching apparatus

Table 11.8 Reactive ion etchant and their target materials

Target material	Etchant Abrev.	Etchant	Etching rate
Silicon	DRIE HF Mech	Mechanical chuck, high frequency, typical recipe	~1500 nm/min
Silicon	DRIE HF ES	Electrostatic struck, high frequency, typical recipe	~1500 nm/min
Silicon	STS $SF_6 + O_2$	$SF_6 + O_2$ 100 W @13.56 MHz, 20 mtorr	~1500 nm/min
Silicon	STS $CF_4 + O_2$	$CF_4 + O_2$ 100 W @13.56 MHz, 60 mtorr	~95 nm/min
SiO_2	STS $SF_6 + O_2$	$SF_6 + O_2$ 100 W @13.56 MHz, 20 mtorr	~32 nm/min
SiO_2	STS $CF_4 + O_2$	$CF_4 + O_2$ 100 W @13.56 MHz, 60 mtorr	~43 nm/min
SiN	STS $SF_6 + O_2$	$SF_6 + O_2$ 100 W @13.56 MHz, 20 mtorr	~190 nm/min

Contd.

SiN	STS $CF_4 + O_2$	$CF_4 + O_2$ 100 W @13.56 MHz, 60 mtorr	~110 nm/min
Chromium	STS $SF_6 + O_2$	$SF_6 + O_2$ 100 W @13.56 MHz, 20 mtorr	<1 nm/min
Chromium	STS $CF_4 + O_2$	$CF_4 + O_2$ 100 W @13.56 MHz, 60 mtorr	<1.3 nm/min
Nicrome (80Ni + 20 Cr)	STS $SF_6 + O_2$	$SF_6 + O_2$ 100 W @13.56 MHz, 20 mtorr	~3.7 nm/min

As for example, in case of Si wafers, the gases used are $Cl_2/F_2/CFCl_3/ CF_4 + O_2$, in case of SiO_2, the gases are $CF_4/CF_4 + O_2/CF_4 + H_2/CHF_3$. The chemical reactions for $CF_4 + O_2$ etchant is given below:

For Si wafer with $CF_4 + O_2$,

$$CF_4 + Si + O_2 = SiF_4 + CO_2$$

For SiO_2 wafer with CF_4, the chemical reaction is

$$CF_4 + SiO_2 = SiF_4 + CO_2$$

11.11.2 Wet Etching

This process is called chemical etching. In case of fabrication of planar waveguide device, it is used for metal etching. Figure 11.31 shows typical wet etching process. It proceeds by reactant transport to the surface—(i) surface reaction (ii) reaction product transport away from the surface and (iii) stirring the solution for increase of (i) or (ii). The etching rate depends on temperature, etching material, and solution composition. Diffusion limited processes have lower activation energies (the order of kcal/mol) than reaction rate controlled processes and therefore, are relatively insensitive to temperature variations. In general, one prefers reaction rate limitation, as it is easier to reproduce a temperature setting than stirring rate. The etching apparatus needs to have a good temperature controller and reliable stirring facilities. Isotropic etchants and also polishing etchants, etch in all crystallographic directions at the same rate. They usually are acidic, such as $HF/HNO_3/CH_3COOH$ (HNA) and lead to rounded isotropic features in single crystalline, Si. Table 11.9 shows different wet etchants with etching targets and rates.

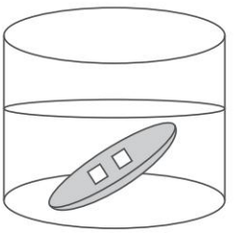


Fig. 11.31 Wet/chemical etching

Table 11.9 Different wet etchants and target materials

Target materiel	Etchant Abrev.	Etchant chemicals and etching temperature	Etch rate
Silicon	KOH	KOH (30% by weight), ~80°C	~1100 nm/min
SiO_2	10:1 HF	10:1 HF(10 H_2O: 1 49% HF), ~20°C	~ 470 nm/min

Contd.

Silicon nitride	Phosphoric	phosphoric acid (85% by weight), ~160°C	~20 nm/min
Aluminum	Al etch A	80% H_3PO_4 + 5%HNO_3 + 5%Hac + 10%H_20, ~50°C	>500 nm/min
Chromium	CR-7	9% $(NH_4)_2Ce(NO_3)_6$ + 6%$HClO_4$ + H_2O, ~20°C	~170 nm/min
Chromium	CR-14	22% $(NH_4)_2Ce(NO_3)_6$ + 8%HAc + H_2O, ~20°C	~93 nm/min
Tungsten	H_2O_2 ,50°C	30 wt % H_2O_2, 70wt%H_2O , ~50°C	~150 nm/min
Copper	Cu $FeCl_3$ 200	CE-200 (30% $FeCl_3$ + 3-4%HCl + H_2O), ~20°C	~3900 nm/min
Gold	Au-5	5%I_2 + 10%KI + 85% H_2O, ~20°C	~660 nm/min
NiCr	NiCr TFN	10-20% $(NH_4)_2Ce(NO_3)_6$ + 5-6%HNO_3 + H_2O,~20°C	~83 nm/min
Positive Photoresist	acetone	acetone, ~20°C	~120 µm/min
Negative Photoresist	acetone	acetone, ~20°C	~87 µm/min

In case of wet etching, the surface reaction takes place between the target material and etchant chemicals. Through diffusion, the products are removed from the surface. Some of the chemical reactions between target material and etchant are given below:

For SiO_2 target,

$$SiO_2 + 6\,HF = H_2\,Si\,F_6(aq) + 2H_2O$$

For Cr target,

$$Cr + 3(NH_4)_2Ce(IV)(NO_3)_6 = Cr(III)\,(NO_3)_3(aq) + 3Ce(III)(NH_4)_2(NO_3)_5\,(aq)$$

For Cu target,

$$Cu + 2FeCl_3 + 3HCl \rightarrow CuCl_3 + 2FeCl_2 + H_2$$

For Au target,

$$Au + I_2 \rightarrow AuI\,(aq)$$

11.12 □ ASSEMBLY AND PACKAGING

11.12.1 Assembly

Before assembly and packaging, front-end-of-line or back-end-of-line technique is made according to the specific requirements of a particular chip application. In the front end of line, steps such as oxidation, diffusion implantation, epitaxy, and lithography etching are almost same and masks used are also same as those for manufacturers whereas in back end of line, these steps may be different for those of manufacturers. These designs are divided in two parts:

- **Semi-Custom Design**: The masks for fabrications may not be determined by the designers.
- **Full-Custom Design**: The masks for fabrications may not be determined by the designers.

After fabrication of IC, proper assembly and packaging is required for easy using, thermal stability, and protection of IC in mass production point of view. Figure 11.32 shows steps used in assembly for plastic or ceramic packaging:

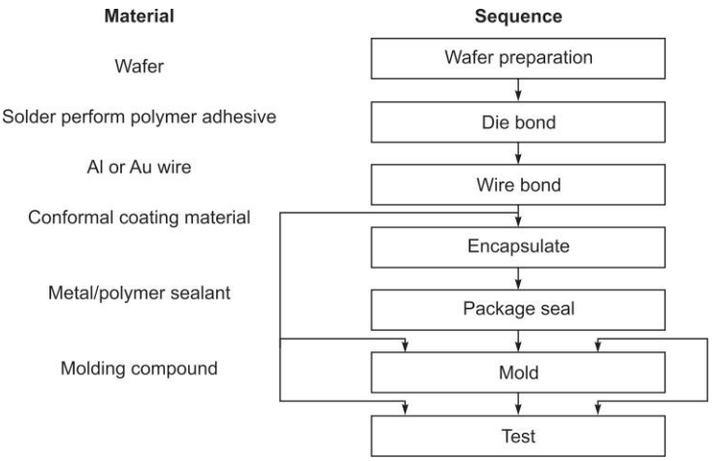


Fig. 11.32 Steps of assembly

(a) Wafer Preparation Wafer backgrinding, wafer slicing, and wafer handling are done in this step.

(b) Die Bonding Thermal stress management is made by using eutectic/hermetic ceramic and polymer or plastic technologies. In case of eutectic die bonding, the die is attached to a substrate material or to a ceramic substrate. Metallization is required on the back of the die to make it wettable by die bonding perform which is a thin sheet of the appropriate solder-bonding alloy. Solder die bonding to ceramic packages that are hermetically sealed is performed with gold.

(c) Wire Bonding Normally, wire bonding is made by using ball-wedge bonding which utilizes the symmetrical geometry of the capillary tip. The ball bond is made in the inner portion of the tip and then wedge bond is performed around the ball bond. Figure 11.33 shows a ball-wedge bonding cycle.

(d) Encapsulate Surface die coating is used for encapsulation to protect from atmospheric contaminants. Silicones are normally used for surface die coating.

(e) Package Seal Packaging seal provides the chip with physical support and environmental protection. Different types of packagings are discussed in the next section.

(f) Molding In molding, thermoset molding material is used for thermal stability. Epoxy resin is also used to mold ICs with condensing epichlorhydrin and bisphenol (combination of these material is called Epoxy-A).

(g) Test There are different kinds of testing made after assembly and packaging. Wire bond pull test and ball shear test are some of the tests which may be used.

11.12.2 Packaging

Each of the wafers processed contains several hundred chips, each being a complete circuit. So these chips must be separated and individually packaged. A common method called **scribing** and **cleaving** used for separation makes use of a diamond-tipped tool to cut lines into the surface of the wafer along the rectangular grid separating the individual chips. Then the wafer is fractured along the scribe lines

and the individual chips are physically separated. Each chip is then mounted on a ceramic wafer and attached to a suitable package. There are three different package configurations available:

1. Glass metal package
2. Ceramic flat package
3. Dual-in-line (ceramic or plastic type)

1. Glass Metal Package

Glass and metal can package is used for simpler chips with 5–12 leads connection. Since the leads/pins are to be isolated from each other, the leads are taken out through glass metal seals. The metal header is made up of an alloy of iron, nickel, and cobalt whose thermal expansion is matched with that of glass. Figure 11.33 shows an example of glass metal package.

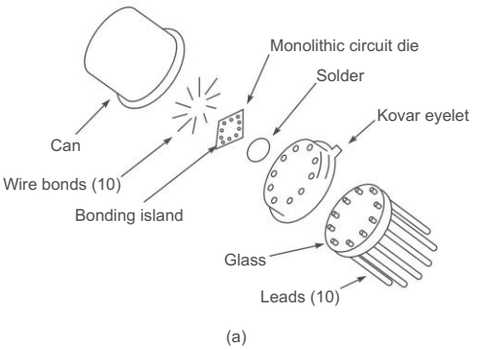


Fig. 11.33 Lead TO-5 glass metal package

2. Ceramic Flat Package

Figure 11.34 shows ceramic flat package in which pins/leads are isolated with ceramic seal material nd flat pins are used. It is available with 8–16 pins even with 24 or 36 or 42 pins. Because of use of ceramic material, this type of packaging is more costly than glass metal packaging.

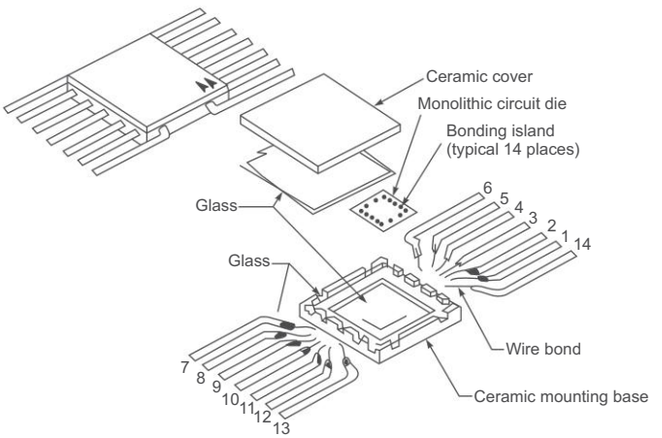


Fig. 11.34 14-lead ceramic flat package

3. Dual-in-line Package

Dual-in-line packaging (DIP) is made with ceramic and plastic molded materials. It consists of three component parts—top, lead fame, and base materials. The top is hermetically sealed whereas the lead frames are made with tin plating and fixed to the base. It is commonly available in 8, 14 or 16 leads, but even 24 or 36 or 42 leads are available for special circuits. Most general-purpose ICs are dual-in-line plastic packages due to economy.

11.13 □ FABRICATION OF A TYPICAL CIRCUIT

We consider the circuit of Fig. 11.35 for IC fabrication. In this section, the various steps are discussed for converting the circuit of Fig. 11.35.

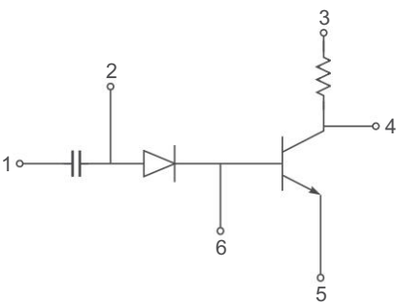


Fig. 11.35 A circuit having diode, BJT, capacitor, and resistor

Step I: Wafer Preparation

The starting material, called the substrate, is a *p*-type silicon wafer as shown in Fig. 11.36(a) (fabrication is discussed in Section 11.2 and 11.4). The wafers are usually of 100 mm diameter and 004 mm (= 400 μm) thickness. The resistivity is approximately 10 Ω-cm corresponding to concentration of acceptor atom, $N_A = 1.4 \times 10^{15}$ atoms/cm^2.

Step 2: Epitaxial Growth

An *n*-type epitaxial film is grown on the *p*-type substrate as shown in Fig. 11.36 (b). This ultimately becomes the collector region of the transistor, or an element of the diode and diffused capacitor associated with the circuit. Normally, all active and passive components are fabricated within this layer. The resistivity of *n*-epitaxial layer is of the order of 0.1–0.5 Ω-cm.

Step 3: Oxidation

Figure 11.36(c) shows an SiO_2 layer of thickness of the order of 0.02 to 2 μm grown on the *n*-epitaxial layer with oxidation (oxidation process is discussed in Section 11.3).

Step 4: Isolation Diffusion

As per the circuit of Figure 11.35, for fabrication of four components, we require four islands which are isolated. For this, SiO_2 is removed from five different places using photolithographic technique (discussed in Section-11.9.4). The wafer is next subjected to heavy *p*-type diffusion for a long time interval so that *p*-type impurities penetrate the *n*-type epitaxial layer and reach the *p*-type substrate. The areas under the SiO_2 are *n*-type islands that are completely surrounded by *p*-type materials as shown in Fig. 11.36(d).

The concentration of acceptor atoms ($N_A \sim 5 \text{ X } 10^{20}$ cm^{-3}) in the region between isolation islands is generally kept higher than *p*-type substrate for which $N_A = 1.4 \times 10^{15}$ atoms/cm^3. This ensures that the depletion region of the reverse-biased diode will not extend into *p*-region to the extent of electrically connecting the two isolation islands. There will, however, be a significant amount of barrier or transition capacitance present as a byproduct of the isolation diffusion.

Step 5: Base Diffusion

A new layer of SiO_2 is grown over the entire wafer and a new pattern of openings is formed using photolithographic technique. Now, *p*-type impurities, such as boron, are diffused through the openings into the islands of *n*-type epitaxial silicon. The depth of this diffusion must be controlled so that it does not penetrate through *n*-layer into the substrate. This diffusion makes the base region of the transistor, resistor, the anode or the diode, and junction capacitor as shown in Fig. 11.36(e).

Step 6: Emitter Diffusion

Like Step 5, a new layer of SiO_2 is again grown over the entire wafer by using PECVD/LPCVD method (discussed in Section 11.6), selectively etched to open a new set of windows by using reactive ion etching (discussed in Section 11.10), and *n*-type impurity (phosphorus) is diffused through them. This forms the transistor emitter and cathode region of diode. Windows are also etched into the *n*-region where contact is to be made to the *n*-type layer. Heavy concentration of phosphorus (n^+) is diffused into these regions simultaneously with the emitter diffusion as shown in Fig. 11.36(f). The reason for using heavily doped *n*-regions can be explained as follows—aluminum, normally used for making interconnections, is a *p*-type impurity in silicon, and can produce an unwanted rectifying contact with the lightly doped *a*-material. However, heavy concentration of phosphorous ($- 2 \times 10^{20}$ cm^{-3}) doping causes a high degree of damage to the Si-lattice at the surface, thus effectively making it semimetallic. This n^+ layer, thus, makes a good ohmic contact with the Al-layer.

Step 7: Aluminium Metallization

Now the IC chip is complete with all the active and passive devices and only interconnections between the various components have to be made in accordance with the circuit. The chip is further subjected to the process of the formation of a new SiO_2 layer and masked etching to open a new set of windows at the points where contacts have to be made.

Now, a thin even coating of aluminum is vacuum-deposited over the entire surface of the wafer (as discussed in Section 11.10). The interconnection pattern between the components is then formed by photoresist techniques. The undesired aluminum areas are etched away leaving a pattern of interconnections between transistor, resistor, diode, and capacitor as shown in Fig. 11.36(g).

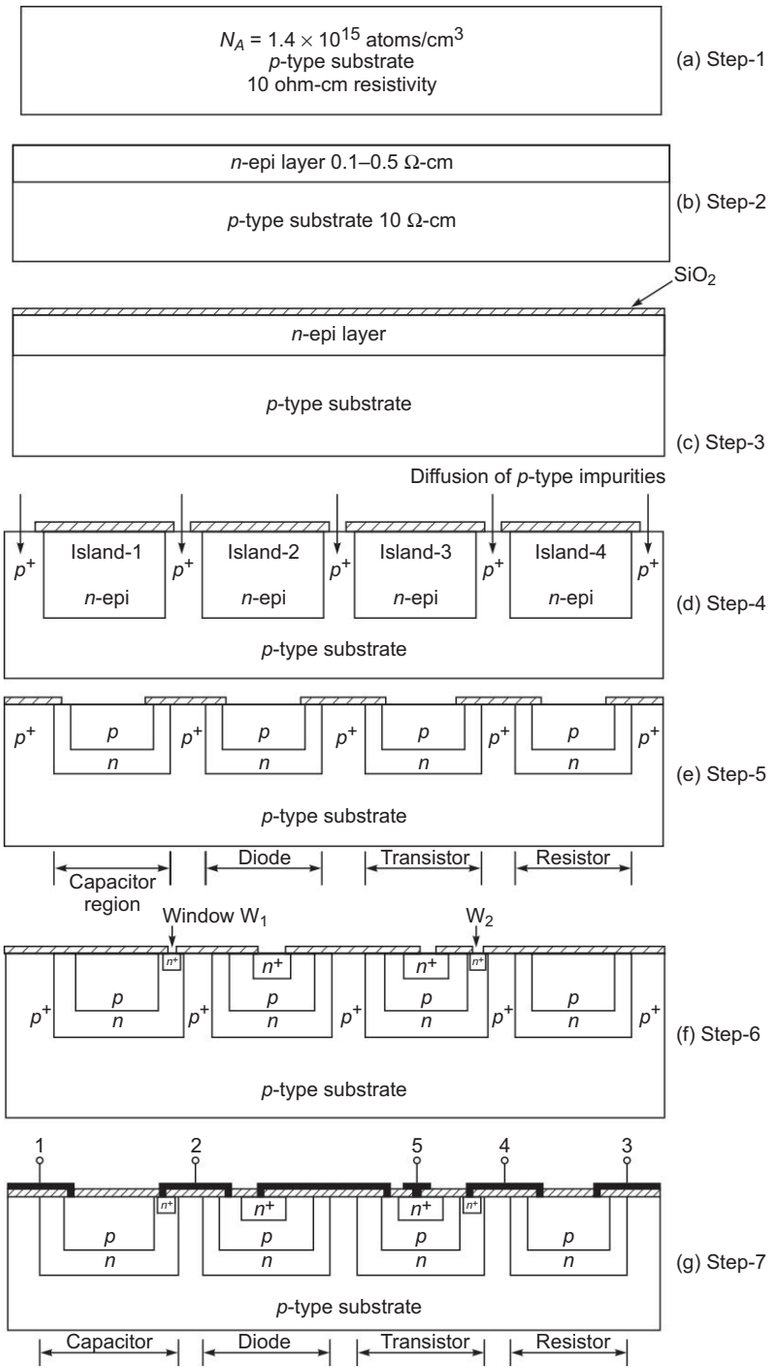


Fig. 11.36 Fabrication steps for processing of IC having capacitor, diode resistor, and BJT

REFERENCES

11.1 Marc, Madou, *Fundamentals of microfabrication*, CRC press, New York, 1997.

11.2 Joy, George, *Preparation of thin films*, Macel Dekker Inc. New York, 1992.

11.3 Kawachi, M., M., Yasu, and J. Edahiro, *Fabrication of SiO_2-TiO_2 glass planar optical waveguide by flame hydrolysis deposition*, IEE electronics letters, vol-19, pp 583–584, 1983.

11.4 Yun Se. Kim and Dong Wook Shin, *Compositional analysis of SiO_2 optical film fabricated by flame hydrolysis deposition*, Journal of Ceramic processing Research, vol-3(3), pp 186–191, 2002.

11.5 Kirti R. William, Kishan Gupta, and Mathew Wasilik, *Etch rates for micromachining processing—part II*, IEEE J. of micromechanical system, 12(6),761–778, 2003.

11.6 Kyungho Lee, Dae kyu Yu, Minchul Chung, Jongchan Kang, and Bumman Kim, *New collector underccut technique using a SiN sidewall for low base compact resistance in InP/InGaAs SHBTs*, IEEE trans. on electron devices, vol-44(6), pp 1089–1082, 2002.

11.7 Pandhumsoporn T. et.al., *High etch rate anistropic deep silicon plasma etching for fabrication of mcrosensors*, Micromaching and microfabrication process technology II, Austin, USA, pp 94–102, 1996.

11.8 Vasile, M.J., C. Biddick, and S. Schwalm, *Microfabrication by ion milling: the Lathe technique*, J. Vac. Cci. Technology, B12, pp 2388–2393, 1994.

11.9 Marxer, C., T. Christiaqn, M.A. Gretillat, N.F. De Rooij, D. Anthamatten, B. Walk, and P. Vogel, *Vertical mirror fabricated by deep reactive ion etching*, IEEE j. micromechanical systems, 16(3), pp 177-285, 1997.

11.10 Chem, K. S., A. Ayon, Xin Zhang, and S.M. Spearing, *Effect of process parameters on the surface morphology and mechanical of silicon structures after deep reactive ion etching* (*DRIE*), IEEE J. of micromechanical system, vol-11(3), pp 264–275, 2002.

EXERCISES

11.1 Describe different steps for wafer preparation including the cleaning step.

11.2 Explain the four-probe technique for measurement of resistance.

11.3 Discuss various kinds of IC design.

11.4 Explain the minimum doping necessary to cause a sample of *p*-type silicon to become degenerate and find the conditions.

11.5 A sample of Si at 300 K is doped with 4.7×10^6 /cm^3 donors and 2.5×10^{15} /cm^3 acceptors. Determine the location of the Fermi level in eV above or below the intrinsic Fermi level. Determine the density of electrons and density of holes and resistivity.

11.6 The empirical formula of plasma deposited silion oxinitride film is $Si\ O_y\ N_x$. Find the values of *x* and *y*, if oxygen content is 10 % and we have an Si/N ratio of 0.9.

11.7 In plasma-deposited $SiN_{0.9}H_{0.6}$, the hydrogen content is 10^{21} atoms/cm^3. The film density is 2g/cm^3. Find the atom density at % H.

11.8 Mention the problems of ion implantations.

11.9 What are the properties of metallization for IC?

11.10 If LPCVD SiO_2 contains 10^{21} atoms/cm^3, The film density is 2 gm/cm^3 find the atom density at % H and the empirical formula.

11.11 Compare the sputter deposition with evaporation deposition for a metal.

11.12 Calculate the depletion region width within *p*-type bulk silicon MOS capacitor with $N_D = 10^{17} cm^{-3}$, at the onset of inversion.

11.13 What do you mean by front-end-of-line and back-end-of-line processes? What is menat by full-custom and semi-custom IC design?

11.14 Convert the following circuit (Fig. P11.1) into IC form and describe steps used for the conversion.

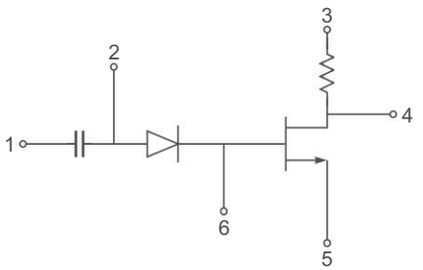


Fig. P11.1 A circuit having diode, BJT, capacitor, and resistor

11.15 Convert the following IC into a circuit.

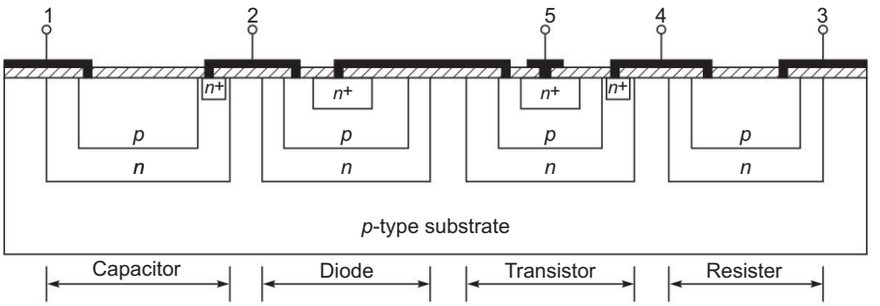


Fig. P11.2 IC diagram

11.16 Convert the following circuit having six terminals into IC form and describe steps used for the conversion.

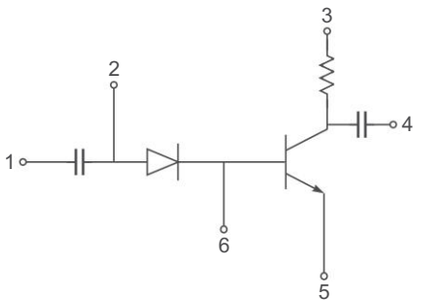


Fig. P11.3 A circuit having diode, BJT, capacitor, and resistor

Index

A

Access 57, 241, 433
Active load, 121, 139, 167
Adaptive biasing, 164, 209
ADC. See Analog-to-digital converter, 168, 179
Adder, 2, 69
Address register, 362, 370, 394
Algorithm 96, 250, 378
  assignment, 269
  BEAVER, 288, 289
  Constrained via minimization
  constructive force-directed, 292, 293
  greedy channel router, 287
  greedy switchbox, 299
  integer linear programming, 281, 283
  iterative force-directed, 303
  iterative-improvement
  Genetic bi-partitioning
  Genetic multi-partitioning
  layout-WRST, 281
  linear placement, 269, 299, 302
  one-step approach
  ratio-cut, 281, 284, 285
  resistive network, 269, 299, 302
  simulated annealing, 264, 300
Aliasing. See Analog-to-digital converter
ALU, 11, 371, 381
Amplifier, 94, 195, 215
  adaptive biasing, 164, 209
  biasing, 118
  cascode, 150, 210
  class A, 126, 152
  class AB, 127, 152
  common gate, 123, 124
  current source load, 121, 124, 137
  distortion, 152
  efficiency, 200
  frequency response, 153
  noise, 109, 140, 143
  push-pull, 125, 210
  single-stage, 121, 123
  slew-rate, 127, 161, 165
  source-follower, 127, 148
  transimpedance, 125
  Op-amp 127
  OTA 144
Analog-to-digital converter, 168
  architectures, 180, 184, 190, 213
  binary search, 188, 189
  charge redistribution, 189, 190, 210
  flash, 108, 180
  least significant bit (LSB), 180,
  oversampling, 211, 215
  parallel, feed-forward, 183
  pipeline, 180, 184
  quantization error, 214
  quantization levels, 188
  residue, 183, 184, 215
  specifications, 140, 221
  successive approximation, 180, 188, 190
  successive approximation register, 188
  transfer curve, 193, 215
  two-step, 183, 209, 215
Annealing 264, 299, 300, 436, 442
  Rapid thermal annealing 436
  Furnace annealing 435, 437
AOI logic, 110
Architecture, 110, 180, 190, 213
Array types, 329, 330
Ashering 453, 454
Assembly and packaging 464
Assignment algorithm, 303
  cost function, 252, 264, 275, 284
  removing overlaps, 273
Asynchronous reset, 344, 368
Attributes 328, 330

B

Batteries, 127, 171
BCD, 105, 109
  corner router, 288
  line sweep router, 288
  thread router 288
BiCMOS, 216, 220, 223
  full-swing, 223
  inverter, 60, 63, 71
  NAND gate, 17, 45, 59
  NOR gate, 10, 47, 62

Biquad, 146
Bit 31, 36
BJT 42, 44, 216
Blocks, 21, 73, 114
Body effect, 17, 19, 20
Bonding pad, 10, 454
Branch, 100, 193
Branch instructions, 371
Breakpoints, 408
Breakdown 21, 29, 95
oxide, 18, 22, 29
voltage, 20, 30, 52
hand calculations, 3, 4, 109, 161
oxide encroachment, 153
subthreshold current, 21, 29
Buffers, 90, 105, 150
Bus 35, 59, 100

C

Capacitance, 9, 17, 24, 30
bottom, 49, 173
depletion, 5, 7, 9
depletion, 5, 7, 9
layout, 12, 30, 34
undercutting, 458
CASE statement, 324, 384
Channel, 6, 14, 40
Channel routing, 285, 296
Channel length modulation, 15, 21, 115
Charge injection, 172, 210, 172
Charge pump, 198
Cleaning 415, 419, 439
Clock 42, 76, 81
Clock feed through, 172
Clock generation, 85
Clock recovery circuit, 196
self-correcting, 199
Clocked logic, 85
CMOS, 5, 9, 30, 170, 220
scaling, 24, 48, 190
CMRR, 140, 143, 161
design rules, 5, 30, 36, 38
oxide capacitance, 18, 25, 91
Common gate amplifier, 123
Common-mode feedback (CMFB), 150
Communication system, 169, 196
block diagram, 90, 140, 169
characterizing, 140, 427
clocked, 77, 85, 96
decision circuit, 166
dynamic, 23, 56, 85
inverter based, 34, 76
operation, 76, 80, 85
output buffer, 59, 90
preamp, 166
propagation delay, 29, 82
self-biased, 209
sensitivity, 3, 120, 129
Compensation, 140, 144
Composite, 103, 329
Contact resistance, 28, 454
Crosstalk, 297
Comparator, 69, 144, 164
Component declaration, 354, 355
Component instantiation, 337, 358
Composite type, 329
Constraint 21, 100
Configurations, 57, 179, 466
Concurrent signal assignment, 322, 326
Concurrent signal assignment statement, 327, 332
Cost function 252, 264
Counter 21, 41, 79
CPLD 307, 340
CPU 370 ,375
RTL 42, 241, 375
Shift 31, 35, 44
testbenches, 410
top-level design, 370, 400
Crystal growth 415, 417
Current differential amplifier, 136, 139
Current source/sink, 117, 158
design, 1, 30, 38
dynamic, 3, 23
matching, 140, 141, 154
mirror, 117, 136
output resistance, 115, 120
self-biasing, 168
sensitivity, 3, 120, 129
wide swing, 139, 212
Current source load, 121, 124, 137
CVD 7, 42, 416

D

Digital-to-Analog Converter, 180, 190
Data conversion: 140
Decoder, 66
tree, 72, 192, 253
Data types 328, 329
array types, 329, 330
composite types, 329, 330

physical types, 329, 330
scalar types, 329, 330
subtypes, 336
Delay-locked loop, 197, 198
Delay time, 53, 72, 111
Delay, 3, 25, 29
Depletion, 5, 7, 9
Depletion capacitance, 91, 216
Design rules, 5, 30, 38
Design verification, 38
Detail routing 252, 275
Differential amplifier, 136, 140, 164
cascode, 136, 138
class AB, 127, 152
current, 3, 11, 13, 17
current source load, 121, 124, 137
gain, 7, 11, 49, 116
low-voltage, 139, 152
matching, 140, 154, 183
noise, 3, 23, 28, 48
output resistance, 115, 117, 119
p-channel, 18, 53, 54, 60
self-biased, 209, 212, 461
slew-rate, 127, 161
source cross-coupled pair, 136
transconductance, 3, 16, 22, 60
wide-swing, 139, 212
Differential output op-amp, 149
simulating, 53, 140, 324
Differential to single-ended
Digital-to-analog converter, 180, 190
accuracy, 28, 183, 190
charge scaling, 190, 194, 214
current steering, 190, 213
cyclic, 195, 311
decoder, 66, 67, 90
dynamic range, 152, 173
input combinations, 228, 236
integral nonlinearity (INL), 185, 193
least significant bit (LSB), 180
most significant bit (MSB), 180
offset, 18, 119, 142
pipeline, 180, 184
R-2R ladder, 190, 193
reference voltage, 120, 121, 181
resistor string, 181, 183, 191
resolution, 28, 42, 180
settling time, 153
specifications, 3, 30, 96
transfer curve, 55, 60, 63
block diagram, 90, 99, 140, 169
charge-pump, 198
loop-filter, 198
tri-state output, 59
Digital signal:
Diode, 154, 216, 467
breakdown voltage, 29, 95
storage capacitance, 85, 177, 216
Distortion, 152,
Diffusion 7, 10, 30, 46, 116
Interchange 3, 423
Interstitial 421, 425
Substitution 54, 421, 423
Drain-induced barrier lowering, 21
Drive current, 3, 23, 221
Drivers, 325, 326,
amplifiers, 90, 114, 121
charge injection, 172, 210
comparator, 69, 144, 164
current mirrors, 119, 154, 171, 178
logic, 28, 34, 53
storage capacitors, 173
switch, 11, 17, 26, 29
switched-capacitor integrator, 175, 212
switched-capacitor resistor, 174
Dynamic CMOS latch, 178
Dynamic logic, 85
domino logic, 87, 111
PE logic, 86, 112,

E

Effective switching resistance, 93
Efficiency, 30, 200
protection, 454, 464, 465
Epi layer, 469
ELSE 211, 321, 332
ELSIF 377, 400,
End-of-file , 403
Entity, 234, 319, 320, 461

F

Feedback, 56, 91, 119
amplifier, 90, 114, 121
closed-loop parameters, 156
open-loop parameters, 156
output sampling, 129
phase margin, 148
series-series, 134
series-shunt, 131, 156
series mixing, 130

series sampling, 130
shunt-series, 135, 157
shunt-shunt, 132, 157
shunt mixing, 130, 135
shunt sampling, 130
stability, 140, 148, 150
Field Programmable Gate Array (FPGA), 297, 307, 365
Filament evaporation 454, 455
File, 224, 308, 328, 458
Filter, 145, 152
switched capacitor, 109, 140, 152
Flip-flops, 77, 80, 93
clocked, 77, 96
edge triggered, 77, 112
JK, 79, 203
timing, 112, 188
Floating current source, 163
Flame Hydrolysis Deposition (FHD)
dual graph, 261, 293
extended dual graph, 262
hierarchical, 251, 260, 267
nonsliceable, 260
Polish expression, 265, 301
rectangular, 251
sliceable, 260, 265
chip area, 121, 192, 241
Genetic algorithm, 258, 273, 297
global routing, 252, 275, 280
hierarchical approach, 263, 281
Floorplan sizing, 261, 267
Vertical 42, 103, 193
Flash evaporation 454
Flip-flop 77, 82
D type 243, 283
FOR loop, 255, 348
FPGA, 296, 300, 305
Full testbench, 401, 405, 410
Functions, 66, 72

G

Global routing, 4, 252, 275
bottom_up_routing, 275
delay minimization, 251, 294
integer linear programming, 281, 283,
integer programming
Gray code, 282
Greedy channel routing 286
Grid 275, 298, 302

H

Hierarchy, 263, 282, 287
High-impedance node, 85, 144
Hysteresis, 92

I

IF statement, 321, 324
Implantation 415, 431, 443
IN, 401, 402, 405, 406, 410
Inverter, 178
BiCMOS, 216, 217, 218
dynamic power dissipation, 56,
layout, 251, 252
noise margins, 55, 109, 224
SPICE simulation, 154
switching characteristics, 218
transfer characteristics, 54, 55, 92
tri-state output, 59
Iterative improvement, 254
INOUT, 374, 405
Instruction, 99, 100
Instruction register, 370, 371
Ion implantation, 415, 431

J

Jitter, 198, 199

K

KP, 408, 438

L

Lambda, 38, 39, 42
Latch, 57, 77, 78
Latch-up, 23, 57
Layout, 85, 91
Layout methodology, 38
detailed routing, 252, 285, 294
floorplanning, 300
global routing, 297, 298
hierarchical approach, 263, 281, 282
layout verification, 252
partitioning, 294, 296
Layout rules, 252
contact, 428, 450
Leakage current, 226, 227, 228
Lee's algorithm, 275
'LEFT, 330
Left-edge algorithm, 286, 304, 305
Level shifting, 168, 171
Libraries, 354, 408

Linear region, 16
Line searching approach
  line probes, 277
  sink probes, 277
  source probes, 277
  track graph, 277, 278, 304
Line sweep connection
  dogleg, 288
  horseshoe, 288
  single bend, 288
  straight-line wire, 288
  Line sweep router, 288
Lithography, 415, 447, 448
  Electron beam lithography, 447
  Ion beam lithography, 448, 449
  X-ray lithography, 447, 448
Load, 3, 453, 462
Load instructions, 378
LoadI, 81
LOOP statement, 324, 348
‘LOW, 330, 359, 360

M

Matching, 140, 154, 183, 214, 349
  differential amplifier, 136, 137, 138, 139
Maze running algorithm, 276, 277, 285
Memory, 314, 328, 351
  cells, 438, 445, 446
  DRAM, 90, 91, 92
  DRAM array, 91
  EEPROM, 89, 338, 339
  EPROM, 89, 90
  flash, 90, 108, 180, 181
  PROM, 338, 339
  SRAM, 90, 91, 93
  Mixed signal, 149, 164
MOSFET, 164, 165, 166
  threshold voltage, 170, 171, 172
  capacitances, 175, 216
  channel length modulation, 15, 16, 21
  depletion, 21, 23, 24
  IV characteristics, 290
  output resistance, 115, 116
  pinch-off, 21, 22, 30
  multiplying quad, 169, 170, 171
  simulating, 140, 324
  squaring circuits, 209
  Multivibrator circuits, 92, 93
  source coupled, 136
  phase, 148, 149
  thermal, 217, 221
  Noise margins, 55, 57, 109, 224
MUX/DEMUX, 84
Multilayer routing, 252, 277
  sink terminal, 275, 276
  source terminal, 275

N

N-select, 458
NAND gate, 17, 45, 59
  layout, 85, 91
  switching characteristics, 218
  switching point voltage, 92, 93, 110
NMOS, 3, 5
NMOS inverter, 33
  Noise, 3, 23, 28
  amplifier, 114, 121
  analysis, 122, 152
  differential amplifier, 164, 165
NOR gate, 205, 220, 233
  switching characteristics, 218

O

Offset voltage, 142, 143
Operational amplifiers, 152, 153
  block diagram, 169, 183, 185
  cascode, 150, 151, 210
  characterizing, 140, 427
  CMRR, 140, 143
  compensation, 143, 144, 150
  compensation capacitor, 173
  differential output, 149
  Output voltage swing, 54, 150, 161
  Phase margin, 148
  PSRR, 140, 143
  slew-rate, 127, 161, 165
  wide-swing, 139, 212, 218
Oscillator, 196, 208
  astable, 92, 93
  ring, 96, 99
  Schmitt Trigger, 92, 113
  voltage-controlled, 196, 208
OTA, 213, 215, 225
  compensation, 140, 141, 143
  DC stability, 148
  folded-cascode, 150
  fully differential, 151, 153, 172
  wide-swing, 139, 212, 218
Output buffer, 140, 141, 144

Output offset storage (OOS), 178
Oxide layer, 13, 23, 421, 440
Oxide breakdown, 95
Oxide capacitance, 18, 25, 91, 115

P

P-base, 196, 197
Package declaration, 354, 355
Packages, 355, 356, 403
Partitioning, 4, 5, 241, 251
  Bipartitioning, 253, 254, 256
Pass transistor, 36, 53, 314
Path selector, 84
Patterning, 9, 11, 42, 108, 448
PE logic, 86, 87, 112
Phase-locked loop (PLL), 195, 196
Phase detector, 196, 197, 198
  characteristics, 11, 13, 16, 54
  gain, 116, 117, 121, 122
  Phase margin, 148
Photolithography 447, 449, 450, 451
Pinch-off, 21, 22, 30
Pipelining, 361, 362, 363
process, 452, 453, 454, 455
Placement 5, 102, 173, 252
  Constructive 269, 270, 271, 298
  Iterative improvement 254
  Simulated annealing 264, 265, 266, 269
  Genetic algorithm 258, 259, 273, 275
Plasma etching 42, 458, 459, 470
Plasma enhanced CVD 437
PMOS, 3, 9, 10, 11
Polish expression 265, 266, 301
Poly, 7, 8, 9, 10
PORT 4, 17, 18, 22
PSPICE, 152
  PSRR, 140, 143, 161
  Pumps, 95, 432, 445
  Push-pull amplifier, 125

R

Reactive ion etching (RIE) 458, 459, 461
  voltage and current, 13
  Regularity, 100
  Reliability,1, 3, 227, 249
Resistance: 7, 14, 25, 28
  looking into the drain, 119, 125
  looking into the source, 124, 125
  sheet, 14, 46, 51, 114, 425
  Resistor, 93, 113, 120, 121
  delay through, 53, 54, 57, 85
  n+, 5, 6, 7, 10
  n-well, 9, 10, 11, 42
  sheet resistance, 14, 46, 51, 114
  switched-capacitor, 109, 140, 152, 174
  TC, See Temperature Coefficient
  Ring oscillator, 56, 76, 109
Ringing, 225, 287, 291, 428
Routing, 3, 4, 5, 252, 260
  Global routing 4, 252, 275, 280
  Detail routing 471, 252, 275, 285

S

Sample-and-hold, 171
hold mode, 173, 174
linearity, 170, 183, 185, 187
offset, 18, 119, 142, 143
sample mode, 173, 174
Schmitt trigger, 92, 93,113
  applications, 2, 3, 4, 71
Self-biasing, 168
Sense amplifier, 90, 91
Sequential statements, 321, 322, 323, 324
  CASE statement, 324, 384, 386, 394
  IF statement, 321, 324
  LOOP statements, 324, 348
Series connection of MOSFETs, 49
  delay, 3, 5, 25, 29
  Settling time, 153
Sheet resistance, 14, 46, 51, 114
Shift register, 31, 36
Short channel, 48
  first-order, 198
  higher order, 435, 452
  noise shaping, 211
Signal assignment, 322, 324, 325, 326
Signal assignment statement, 322, 326, 332, 412
Signals, 5, 36, 53, 68
Silicon, 1, 3, 7, 8
Silicon dioxide 30, 419, 421, 440
Silicon nitride 418, 441, 442, 464
Silicon wafer preparation 415
Slew rate, 126, 127, 140, 142
Source coupled VCO, 197
Source follower, 127, 148, 154, 171
SPICE, 21, 109, 152, 154
  distortion, 152
  transient, 446
Sputtering 454, 457, 460, 461

Sputter etching 458, 460
Squaring circuits, 209
Stability, 3, 4, 140, 148
Standard-Cell, 236
Static logic, 110
Stepper, 38, 445
Stick diagram, 30, 31, 32, 33, 34
Storage node, 85
functions, 66, 72, 73
procedures, 225, 358
Substrate, 415, 420, 421
  current source/sink, 117, 158
  output resistance, 160, 178
  transconductance, 178, 221
  Switched capacitor: 109, 140
  filter, 145, 146
  integrator, 175, 176, 185
  resistor, 186, 190, 191
  Synthesis, 196, 294

T

Testbenches, 410
Threshold voltage, 3, 7, 9
Thermal voltage, 217, 221
Transconductance, 16, 22, 23
Transconductance parameter, 23, 60, 178
Transimpedance amplifier, 125
Transmission gate, 83, 84, 100
Tri-state buffer, 110, 358

V

Vaccum creating system, 445
'VALUE, 371, 378
Vapour phase epitaxy, 443
Variable assignment, 329
Variables, 102, 104, 396, 403
VLSI, 108, 112, 118
Voltage controlled oscillator (VCO), 196, 208
Voltage dividers, 120
Voltage follower, 159, 165, 212
Voltage generators, 92, 94
  higher voltages, 95

W

Wafer, 415, 417, 419
  Cleaning, 419, 420
  Polishing, 420
Waveforms, 172, 329
Wet etching, 458, 463
Weak inversion, 21
Wire-length minimization, 3, 5, 251
WRST, 278, 279, 280

X

XOR gate, 65, 74
  characteristics, 92, 109
  gain, 116, 117, 121

Y

Yet another channel router, 286, 299
  branch, 371, 373
  trunk, 286
  YACR, 286, 299
Yield, 330, 331

Z

  Zipper logic, 88