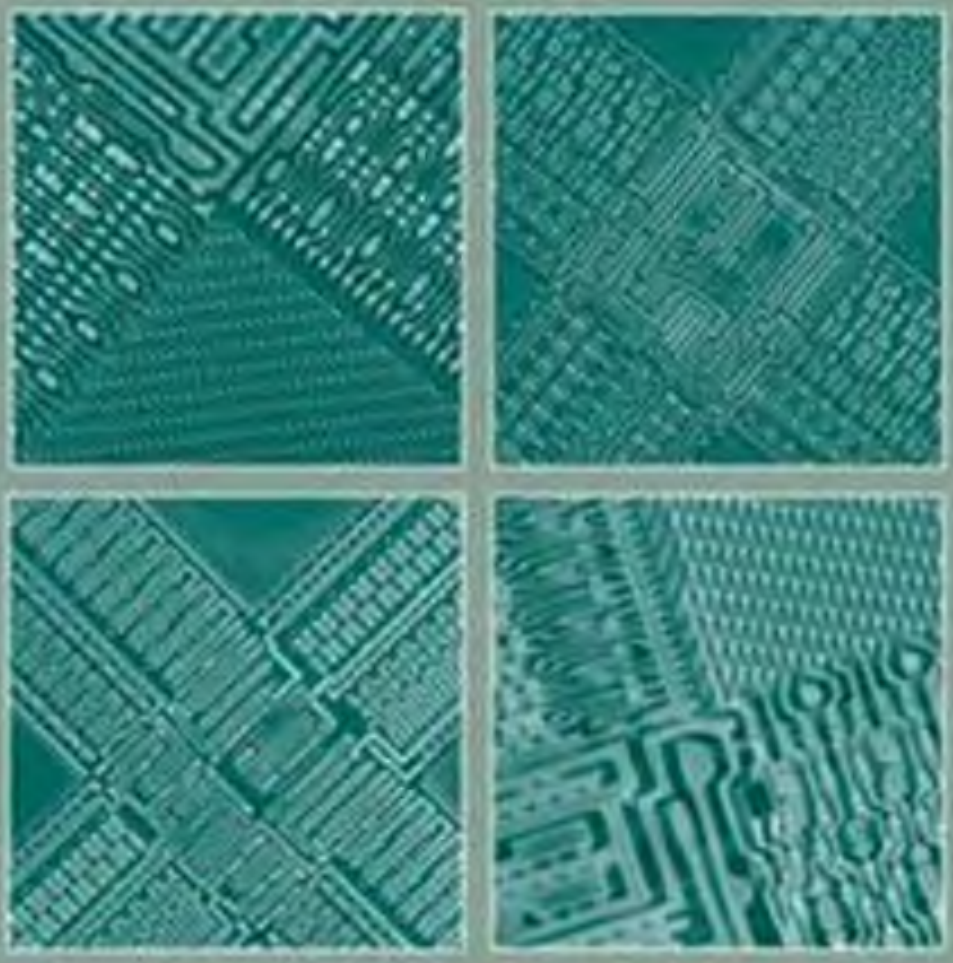
DRAM CIRCUIT DESIGN

Fundamental and High-Speed Topics



Brent Keeth • R. Jacob Baker
Brian Johnson • Feng Lin

DRAM Circuit Design

IEEE Press
445 Hoes Lane
Piscataway, NJ 08854

IEEE Press Editorial Board
Mohamed E. El-Hawary, *Editor in Chief*

R. Abari	T. G. Croda	S. V. Kartalopoulos
S. Basu	S. Farshchi	M. S. Newman
A. Chatterjee	B. M. Hammerli	
T. Chen	R. J. Herrick	

Kenneth Moore, *Director of IEEE Book and Information Services (BIS)*
Catherine Faduska, *Senior Acquisitions Editor*
Jeanne Audino, *Project Editor*

IEEE Solid-State Circuits Society, Sponsor
IEEE SSCS Liaison to the IEEE Press, Stuart K. Tewksbury

Technical Editing and Illustrations
Mary J. Miller, Micron Technology, Inc.

Technical Reviewers
Jim Frenzel, University of Idaho
Zhao Zhang, Iowa State University

DRAM Circuit Design

Fundamental and High-Speed Topics

Brent Keeth
R. Jacob Baker
Brian Johnson
Feng Lin

IEEE Press Series on Microelectronic Systems
Stuart K. Tewksbury and Joe E. Brewer, *Series Editors*

IEEE Solid-State Circuits Society, *Sponsor*



IEEE PRESS



WILEY-INTERSCIENCE
A John Wiley & Sons, Inc., Publication

Copyright © 2008 by the Institute of Electrical and Electronic Engineers, Inc. All rights reserved.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4470, or on the web at www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at http://www.wiley.com/go/permission.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services or for technical support, please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic format. For information about Wiley products, visit our web site at www.wiley.com.

Wiley Bicentennial Logo: Richard J. Pacifico

Library of Congress Cataloging-in-Publication Data is available.

ISBN 978-0-470-18475-2

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

To Susi, John, Katie, Gracie, Dory, and Faith (B.K.)
To Julie, Kyri, and Josh (J.B.)
To Cassandra, Parker, Spencer, and Dolpha (B.J.)
To Hui, Luan, and Conrad (F.L.)
To Leah and Nicholas (M.M.)

Contents

Preface xi

Chapter 1 An Introduction to DRAM

1.1 DRAM Types and Operation 1
1.1.1 The 1k DRAM (First Generation) 1
1.1.2 The 4k–64 Meg DRAM (Second Generation) 7
1.1.3 Synchronous DRAM (Third Generation) 15
1.2 DRAM Basics 22
1.2.1 Access and Sense Operations 24
1.2.2 Write Operation 28
1.2.3 Opening a Row (Summary) 29
1.2.4 Open/Folded DRAM Array Architectures 31

Chapter 2 The DRAM Array

2.1 The Mbit Cell 33
2.2 The Sense Amp 44
2.2.1 Equilibration and Bias Circuits 44
2.2.2 Isolation Devices 46
2.2.3 Input/Output Transistors 46
2.2.4 Nsense and Psense Amplifiers 47
2.2.5 Rate of Activation 49
2.2.6 Configurations 49
2.2.7 Operation 52
2.3 Row Decoder Elements 54
2.3.1 Bootstrap Wordline Driver 55
2.3.2 NOR Driver 57
2.3.3 CMOS Driver 58

2.3.4 Address Decode Tree 58
2.3.5 Static Tree 59
2.3.6 P&E Tree 59
2.3.7 Predecoding 60
2.3.8 Pass Transistor Tree 61
2.4 Discussion 61

Chapter 3 Array Architectures

3.1 Array Architectures 65
3.1.1 Open Digitline Array Architecture 65
3.1.2 Folded Array Architecture 75
3.2 Design Examples: Advanced Bilevel DRAM Architecture 83
3.2.1 Array Architecture Objectives 84
3.2.2 Bilevel Digitline Construction 85
3.2.3 Bilevel Digitline Array Architecture 88
3.2.4 Architectural Comparison 93

Chapter 4 The Peripheral Circuitry

4.1 Column Decoder Elements 99
4.2 Column and Row Redundancy 102
4.2.1 Row Redundancy 104
4.2.2 Column Redundancy 107

Chapter 5 Global Circuitry and Considerations

5.1 Data Path Elements 111
5.1.1 Data Input Buffer 111
5.1.2 Data Write Muxes 115
5.1.3 Write Driver Circuit 116
5.1.4 Data Read Path 118
5.1.5 DC Sense Amplifier (DCSA) 119
5.1.6 Helper Flip-Flop (HFF) 121
5.1.7 Data Read Muxes 122
5.1.8 Output Buffer Circuit 124
5.1.9 Test Modes 125
5.2 Address Path Elements 126
5.2.1 Row Address Path 126
5.2.2 Row Address Buffer 127
5.2.3 CBR Counter 127
5.2.4 Predecode Logic 128
5.2.5 Refresh Rate 128
5.2.6 Array Buffers 130
5.2.7 Phase Drivers 131

5.2.8 Column Address Path 131
5.2.9 Address Transition Detection 132
5.3 Synchronization in DRAMs 135
5.3.1 The Phase Detector 137
5.3.2 The Basic Delay Element 137
5.3.3 Control of the Shift Register 138
5.3.4 Phase Detector Operation 139
5.3.5 Experimental Results 140
5.3.6 Discussion 142

Chapter 6 Voltage Converters
6.1 Internal Voltage Regulators 147
6.1.1 Voltage Converters 147
6.1.2 Voltage References 148
6.1.3 Bandgap Reference 153
6.1.4 The Power Stage 154
6.2 Pumps and Generators 158
6.2.1 Pumps 158
6.2.2 DVC2 Generator 165
6.3 Discussion 165

Chapter 7 An Introduction to High-Speed DRAM
7.1 The Performance Paradigm 167
7.2 Performance for DRAM Memory Devices 169
7.3 Underlying Technology Improvements 171

Chapter 8 High-Speed Die Architectures
8.1 Introduction: Optimizing DRAM Architecture for High Performance 173
8.2 Architectural Features: Bandwidth, Latency, and Cycle Time . 175
8.2.1 Architectural Limiters: The Array Data Path 176
8.2.2 Architectural Limiters: The Read Data Path 183
8.2.3 Architectural Limiters: Latency 186
8.3 Conclusion: Designing for High Performance 190

Chapter 9 Input Circuit Paths
9.1 Introduction 193
9.2 Input Receivers 196
9.3 Matched Routing 200
9.4 Capture Latches 203
9.5 Input Timing Adjustments 206
9.6 Current Mode Logic (CML) 212

Chapter 10 Output Circuit Paths

10.1 Transmission Line, Impedance, and Termination 220
10.2 Impedance Control 224
10.3 Simultaneous Switching Noise (SSN) 230
10.4 Signal Return Path Shift (SRPS) 238
10.5 Electrostatic Discharge (ESD) 242
10.6 Parallel-to-Serial Conversion 244
10.7 Emerging Memory *I/O* Features 248

Chapter 11 Timing Circuits

11.1 Introduction 251
11.2 All-Digital Clock Synchronization Design 254
11.2.1 Timing Analysis 255
11.2.2 Digital Delay Line 259
11.2.3 Phase Detector (PD) 267
11.2.4 Test and Debug 273
11.2.5 Dual-Loop Architecture 274
11.3 Mixed-Mode Clock Synchronization Design 275
11.3.1 Analog Delay Line 276
11.3.2 Charge-Pump Phase Detector (CPPD) 283
11.3.3 Dual-Loop Analog DLL 286
11.3.4 Mixed-Mode DLL and Its Applications 289
11.4 What's Next for Timing 291

Chapter 12 Control Logic Design

12.1 Introduction 295
12.2 DRAM Logic Styles 298
12.2.1 Process Limitations 298
12.2.2 Array Operation 301
12.2.3 Performance Requirements 304
12.2.4 Delay-Chain Logic Style 306
12.2.5 Domino Logic 309
12.2.6 Testability 314
12.3 Command and Address Control 317
12.3.1 Command Decoder 318
12.3.2 Read and Write Data Address Registers 324
12.3.3 Column Access Control 328
12.4 Write Data Latency Timing and Data Demultiplexing 330
12.4.1 Write Latency Timing 332
12.4.2 Write Data Demultiplexing 338
12.5 Read Data Latency Timing and Data Multiplexing 346
12.5.1 Read Data FIFO 350

12.5.2 Read Latency (CL) Tracking 359
12.6 Comments on Future Direction for DRAM Logic Design . . . 367

Chapter 13 Power Delivery
13.1 Power Delivery Network Design. 373
13.2 Device/Package Co-design 376
13.3 Full-Chip Simulations 380

Chapter 14 Future Work in High-Performance Memory 385

Appendix. 391

Glossary 407

Index 413

Preface

From the core memory that rocketed into space during the Apollo moon missions to the solid-state memories used in today's commonplace computer, memory technology has played an important, albeit quiet, role during the prior and current centuries. It has been quiet in the sense that memory, although necessary, is not glamorous and sexy, and is instead being relegated to the role of a commodity. Yet, it is important because memory technology, specifically, CMOS DRAM technology, has been one of the greatest driving forces in the advancement of solid-state technology. It remains a driving force today, despite the segmentation in its market space.

The very nature of the commodity memory market, with high product volumes and low pricing, is what ultimately drives the technology. To survive, let alone remain viable over the long term, memory manufacturers work aggressively to drive down their manufacturing costs while maintaining, if not increasing, their share of the market. One of the best tools to achieve this goal remains the ability of manufacturers to shrink their technology, essentially in getting more memory chips per wafer through process scaling. Unfortunately, with all memory manufacturers pursuing the same goals, it is literally a race to see who can get there first. As a result, there is tremendous pressure to advance the state of the art—more so than in other related technologies due to the commodity status of memory.

While the memory industry continues to drive forward, most people can relax and enjoy the benefits—except for those of you who need to join in the fray. For you, the only way out is straight ahead, and it is for you that we have written the second edition of this book.

The goal of *DRAM Circuit Design: Fundamental and High-Speed Topics* is to bridge the gap between the introduction to memory design available in most CMOS circuit texts and the advanced articles on DRAM design that

are available in technical journals and symposium digests. The book introduces the reader to DRAM theory, history, and circuits in a systematic, tutorial fashion. The level of detail varies, depending on the topic. In most cases, however, our aim is merely to introduce the reader to a functional element and illustrate it with one or more circuits. After gaining familiarity with the purpose and basic operation of a given circuit, the reader should be able to tackle more detailed papers on the subject.

The second half of the book is completely devoted to advanced concepts pertaining to state-of-the-art high-speed and high-performance DRAM memory. The two halves of the book are worlds apart in content. This is intentional and serves to rapidly advance the reader from novice to expert through the turning of a page.

The book begins in Chapter 1 with a brief history of DRAM device evolution from the first 1Kbit device to the 64Mbit synchronous devices. This chapter introduces the reader to basic DRAM operation in order to lay a foundation for more detailed discussion later. Chapter 2 investigates the DRAM memory array in detail, including fundamental array circuits needed to access the array. The discussion moves into array architecture issues in Chapter 3, including a design example comparing known architecture types to a novel, stacked digitline architecture. This design example should prove useful, for it delves into important architectural trade-offs and exposes underlying issues in memory design. Chapter 4 then explores peripheral circuits that support the memory array, including column decoders and redundancy. The reader should find Chapter 5 very interesting due to the breadth of circuit types discussed. This includes data path elements, address path elements, and synchronization circuits. Chapter 6 follows with a discussion of voltage converters commonly found on DRAM designs. The list of converters includes voltage regulators, voltage references, $V_{DD}/2$ generators, and voltage pumps.

Chapter 7 introduces the concept of high-performance memory and underlying market forces. Chapter 8 examines high-speed DRAM memory architectures including a discussion of performance-cost trade-offs. Chapter 9 takes a look at the input circuit path of high-performance DRAM while Chapter 10 takes a complementary look at the output circuit path. Chapter 11 dives into the complicated world of high-performance timing circuits including delay-lock-loops and phase-lock-loops. Chapter 12 ties it all together by tackling the difficult subject of control logic design. This is an especially important topic in high-performance memory chips. Chapter 13 looks at power delivery and examines methods to improve performance through careful analysis and design of the power delivery network. Finally, Chapter 14 discusses future work in high-performance memory. We wrap

up the book with the Appendix, which directs the reader to a detailed list of papers from major conferences and journals.

Brent Keeth
R. Jacob Baker
Brian Johnson
Feng Lin

Chapter
1

An Introduction to DRAM

Dynamic random access memory (DRAM) integrated circuits (ICs) have existed for more than thirty years. DRAMs evolved from the earliest kilobit (Kb) generation to the gigabit (Gb) generation through advances in both semiconductor process and circuit design technology. Tremendous advances in process technology have dramatically reduced feature size, permitting ever higher levels of integration. These increases in integration have been accompanied by major improvements in component yield to ensure that overall process solutions remain cost-effective and competitive. Technology improvements, however, are not limited to semiconductor processing. Many of the advances in process technology have been accompanied or enabled by advances in circuit design technology. In most cases, advances in one have enabled advances in the other. In this chapter, we introduce some fundamentals of the DRAM IC, assuming that the reader has a basic background in complementary metal-oxide semiconductor (CMOS) circuit design, layout, and simulation [1].

1.1 DRAM TYPES AND OPERATION

To gain insight into how modern DRAM chips are designed, it is useful to look into the evolution of DRAM. In this section, we offer an overview of DRAM types and modes of operation.

1.1.1 The 1k DRAM (First Generation)

We begin our discussion by looking at the 1,024-bit DRAM (1,024 x 1 bit). Functional diagrams and pin connections appear in Figure 1.1 and Figure 1.2, respectively. Note that there are 10 address inputs with pin labels R_1–R_5 and C_1–C_5. Each address input is connected to an on-chip

address input buffer. The input buffers that drive the *row (R)* and *column (C)* decoders in the block diagram have two purposes: to provide a known *input capacitance* (C_{IN}) on the address input pins and to detect the input address signal at a known level so as to reduce timing errors. The level V_{TRIP}, an idealized trip point around which the input buffers slice the input signals, is important due to the finite transition times on the chip inputs (Figure 1.3). Ideally, to avoid distorting the duration of the logic zeros and ones, V_{TRIP} should be positioned at a known level relative to the maximum and minimum input signal amplitudes. In other words, the reference level should change with changes in temperature, process conditions, *input maximum amplitude* (V_{IH}), and *input minimum amplitude* (V_{IL}). Having said this, we note that the input buffers used in first-generation DRAMs were simply inverters.

Continuing our discussion of the block diagram shown in Figure 1.1, we see that five address inputs are connected through a decoder to the 1,024-bit memory array in both the row and column directions. The total number of addresses in each direction, resulting from decoding the 5-bit word, is 32. The single memory array is made up of 1,024 memory elements laid out in a square of 32 rows and 32 columns. Figure 1.4 illustrates the conceptual layout of this memory array. A memory element is located at the intersection of a row and a column.

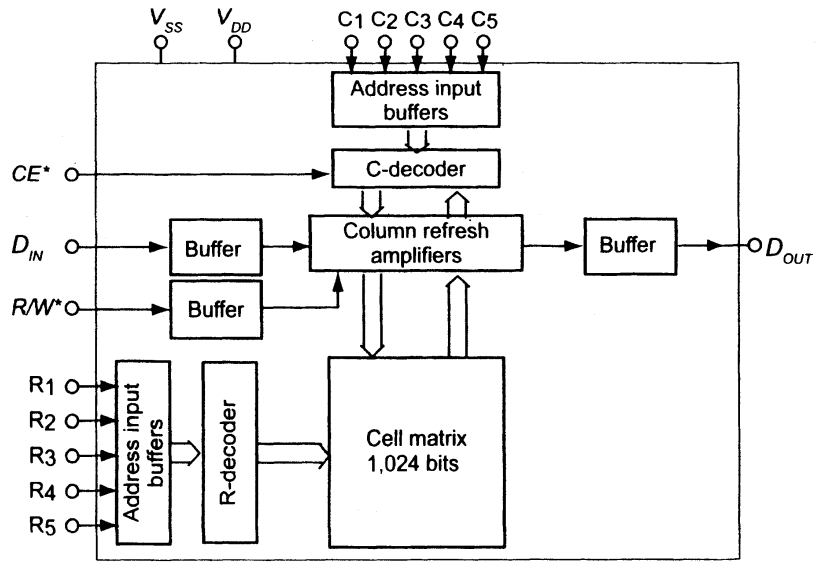


Figure 1.1 1,024-bit DRAM functional diagram.

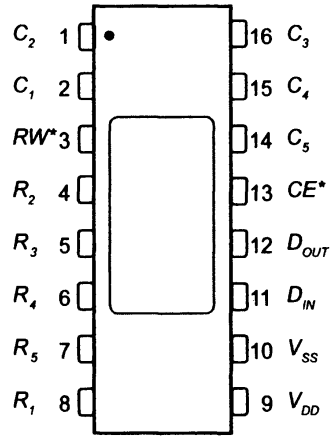


Figure 1.2 1,024-bit DRAM pin connections.

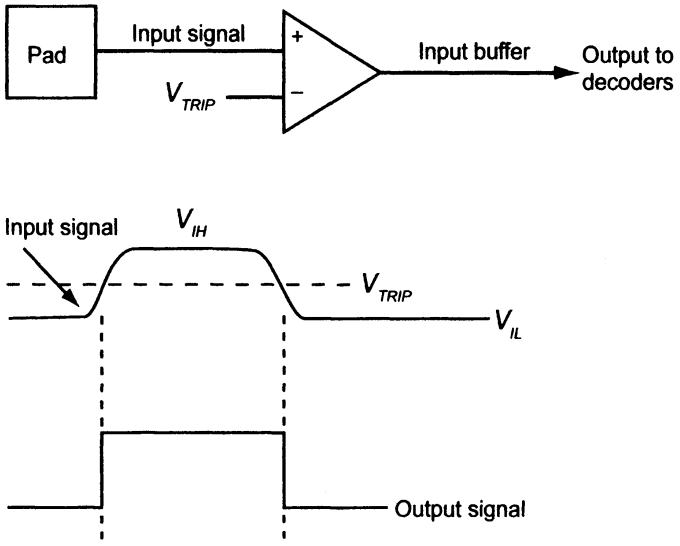


Figure 1.3 Ideal address input buffer.

By applying an address of all zeros to the 10 address input pins, the memory data located at the intersection of row0, *RA*0, and column 0, *CA*0, is accessed. (It is either written to or read out, depending on the state of the *R/W** input and assuming that the *CE** pin is LOW so that the chip is enabled.)

It is important to realize that a single bit of memory is accessed by using both a row and a column address. Modern DRAM chips reduce the number of external pins required for the memory address by using the same pins for both the row and column address inputs *(address multiplexing).* A clock signal *row address strobe (RAS*)* strobes in a row address and then, on the same set of address pins, a clock signal *column address strobe (CAS*)* strobes in a column address at a different time.

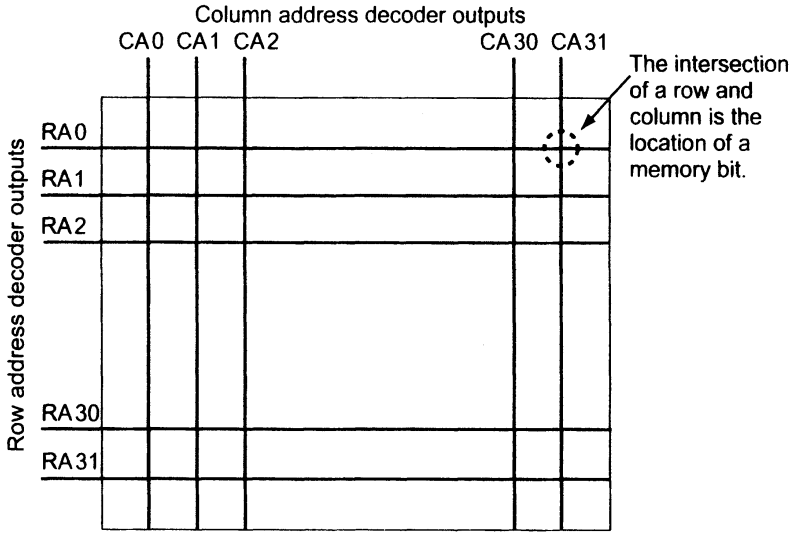


Figure 1.4 Layout of a 1,024-bit memory array.

Also note how a first-generation memory array is organized as a logical square of memory elements. (At this point, we don't know what or how the memory elements are made. We just know that there is a circuit at the intersection of a row and column that stores a single bit of data.) In a modern DRAM chip, many smaller memory arrays are organized to achieve a larger memory size. For example, 1,024 smaller memory arrays, each composed of 256 kbits, may constitute a 256-Meg (256 million bits) DRAM.

1.1.1.1 Reading Data Out of the 1k DRAM. Data can be read out of the DRAM by first putting the chip in the Read mode by pulling the *R/W** pin HIGH and then placing the chip enable pin *CE** in the LOW state. Figure 1.5 illustrates the timing relationships between changes in the address inputs and data appearing on the D_{OUT} pin. Important timing specifications present in this figure are *Read cycle time* (t_{RC}) and *Access time* (t_{AC}). The term t_{RC} specifies how fast the memory can be read. If t_{RC} is 500 ns, then the DRAM can supply 1-bit words at a rate of 2 MHz. The term t_{AC} specifies the maximum length of time after the input address is changed before the output data (D_{OUT}) is valid.

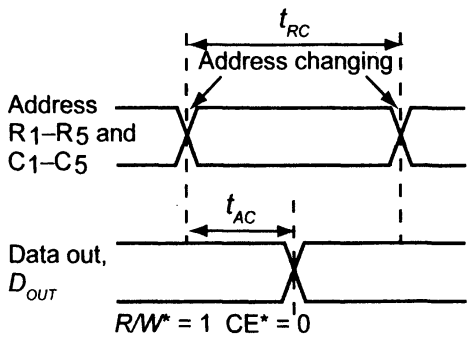


Figure 1.5 **1k** DRAM Read cycle.

1.1.1.2 Writing to the 1k DRAM. Writing data to the DRAM is accomplished by bringing the *R/W** input LOW with valid data present on the D_{IN} pin. Figure 1.6 shows the timing diagram for a Write cycle. The term *Write cycle time* (t_{WC}) is related to the maximum frequency at which we can write data into the DRAM. The term *Address to Write delay time* (t_{AW}) specifies the time between the address changing and the *R/W** input going LOW. Finally, *Write pulse width* (t_{WP}) specifies how long the input data must be present before the *R/W** input can go back HIGH in preparation for another Read or Write to the DRAM. When writing to the DRAM, we can think of the *R/W** input as a clock signal.

1.1.1.3 Refreshing the 1k DRAM. The dynamic nature of DRAM requires that the memory be refreshed periodically so as not to lose the contents of the memory cells. Later we will discuss the mechanisms that lead to the dynamic operation of the memory cell. At this point, we discuss how memory Refresh is accomplished for the 1k DRAM.

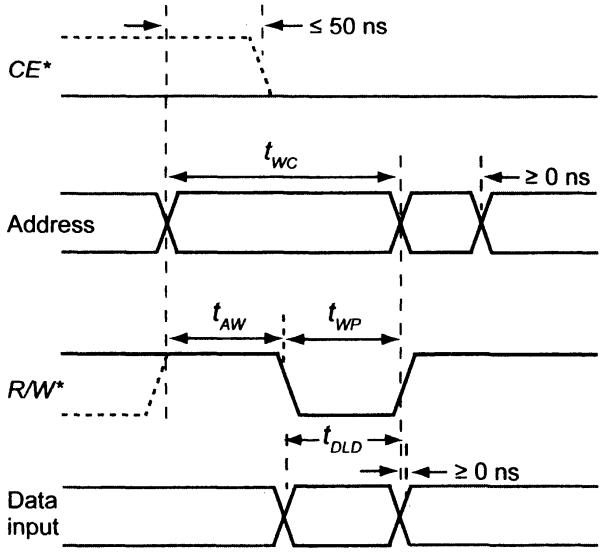


Figure 1.6 1k DRAM Write cycle.

Refreshing a DRAM is accomplished internally: external data to the DRAM need not be applied. To refresh the DRAM, we periodically access the memory with every possible row address combination. A timing diagram for a Refresh cycle is shown in Figure 1.7. With the *CE** input pulled HIGH, the address is changed, while the *R/W** input is used as a strobe or clock signal. Internally, the data is read out and then written back into the same location at full voltage; thus, logic levels are restored (or refreshed).

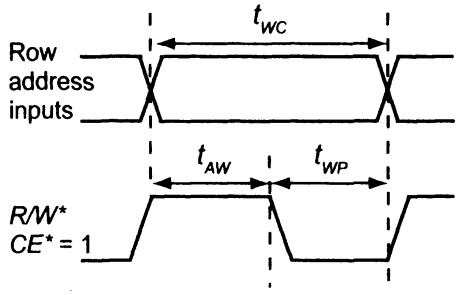


Figure 1.7 1k DRAM Refresh cycle.

1.1.1.4 A Note on the Power Supplies. The voltage levels used in the 1k DRAM are unusual by modern-day standards. In reviewing Figure 1.2, we see that the 1k DRAM chip uses two power supplies: V_{DD} and V_{SS}. To begin, V_{SS} is a greater voltage than V_{DD}: V_{SS} is nominally 5 V, while V_{DD} is −12 V. The value of V_{SS} was set by the need to interface to logic circuits that were implemented using transistor-transistor logic (TTL) logic. The 17 V difference between V_{DD} and V_{SS} was necessary to maintain a large signal-to-noise ratio in the DRAM array. We discuss these topics in greater detail later in the book. The V_{SS} power supply used in modern DRAM designs, at the time of this writing, is generally zero; the V_{DD} is in the neighborhood of 1.5 V.

1.1.1.5 The 3-Transistor DRAM Cell. One of the interesting circuits used in the 1k DRAM (and a few of the 4k and 16k DRAMs) is the 3-transistor DRAM memory cell shown in Figure 1.8. The column- and rowlines shown in the block diagram of Figure 1.1 are split into Write and Read line pairs. When the Write rowline is HIGH, M1 turns ON. At this point, the data present on the Write columnline is passed to the gate of M2, and the information voltage charges or discharges the input capacitance of M2. The next, and final, step in writing to the mbit cell is to turn OFF the Write rowline by driving it LOW. At this point, we should be able to see why the memory is called dynamic. The charge stored on the input capacitance of M2 will leak off over time.

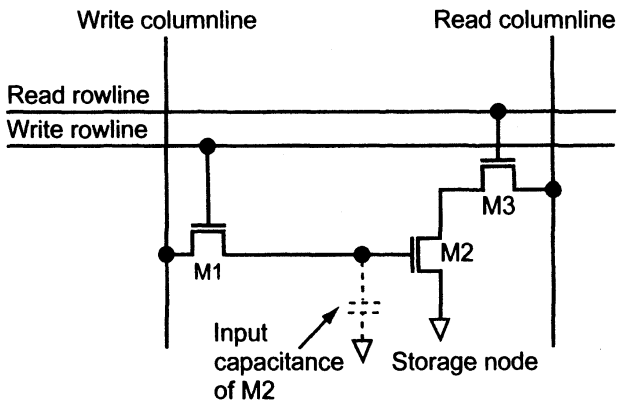


Figure 1.8 3-transistor DRAM cell.

If we want to read out the contents of the cell, we begin by first precharging the Read columnline to a known voltage and then driving the Read rowline HIGH. Driving the Read rowline HIGH turns M3 ON and allows M2 either to pull the Read columnline LOW or to not change the precharged voltage of the Read columnline. (If M2's gate is a logic LOW, then M2 will be OFF, having no effect on the state of the Read columnline.) The main drawback of using the 3-transistor DRAM cell, and the reason it is no longer used, is that it requires two pairs of column and rowlines and a large layout area. Modern 1-transistor, 1-capacitor DRAM cells use a single rowline, a single columnline, and considerably less area.

1.1.2 The 4k–64 Meg DRAM (Second Generation)

We distinguish second-generation DRAMs from first-generation DRAMs by the introduction of multiplexed address inputs, multiple memory arrays, and the 1-transistor/1-capacitor memory cell. Furthermore, second-generation DRAMs offer more modes of operation for greater flexibility or higher speed operation. Examples are *page mode, nibble mode, static column mode, fast page mode* (FPM), and *extended data out* (EDO). Second-generation DRAMs range in size from 4k (4,096 x 1 bit, i.e., 4,096 address locations with 1-bit input/output word size) up to 64 Meg (67,108,864 bits) in memory sizes of 16 Meg x 4 organized as 16,777,216 address locations with 4-bit input/output word size, 8 Meg x 8, or 4 Meg x 16.

Two other major changes occurred in second-generation DRAMs: (1) the power supply transitioned to a single 5 V and (2) the technology advanced from NMOS to CMOS. The change to a single 5 V supply occurred at the 64kbit density. It simplified system design to a single power supply for the memory, processor, and any TTL (transistor-transistor logic) used in the system. As a result, rowlines had to be driven to a voltage

greater than 5 V to turn the NMOS access devices fully ON (more on this later), and the substrate held at a potential less than zero. For voltages outside the supply range, charge pumps are used (see Chapter 6). The move from NMOS to CMOS, at the 1Mb density level, occurred because of concerns over speed, power, and layout size. At the cost of process complexity, complementary devices improved the design.

1.1.2.1 Multiplexed Addressing. Figure 1.9 shows a 4k DRAM block diagram, while Figure 1.10 shows the pin connections for a 4k chip. Note that compared to the block diagram of the 1k DRAM shown in Figure 1.1, the number of address input pins has decreased from 10 to 6, even though the memory size has quadrupled. This is the result of using multiplexed addressing in which the same address input pins are used for both the row and column addresses. The row address strobe *(RAS*)* input clocks the address present on the DRAM address pins A_0 to A_5 into the row address latches on the falling edge. The column address strobe *(CAS*)* input clocks the input address into the column address latches on its falling edge.

Figure 1.11 shows the timing relationships between *RAS**, *CAS**, and the address inputs. Note that t_{RC} is still (as indicated in the last section) the random cycle time for the DRAM, indicating the maximum rate we can write to or read from a DRAM. Note too how the row (or column) address must be present on the address inputs when *RAS** (or *CAS**) goes LOW. The parameters t_{RAS} and t_{CAS} indicate how long *RAS** or *CAS** must remain LOW after clocking in a column or row address. The parameters t_{ASR}, t_{RAH}, t_{ASC}, and t_{CAH} indicate the setup and hold times for the row and column addresses, respectively.

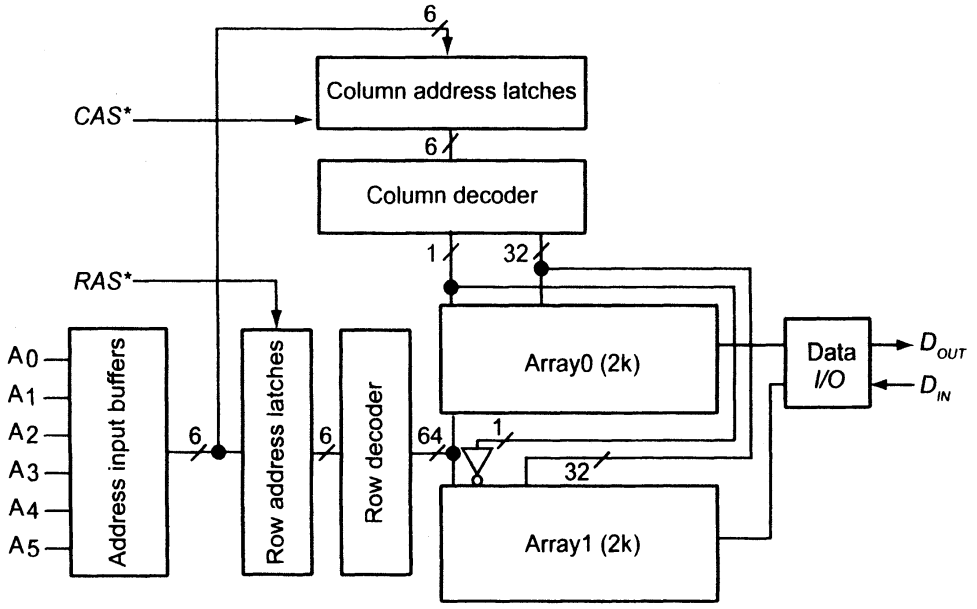


Figure 1.9 Block diagram of a 4k DRAM.

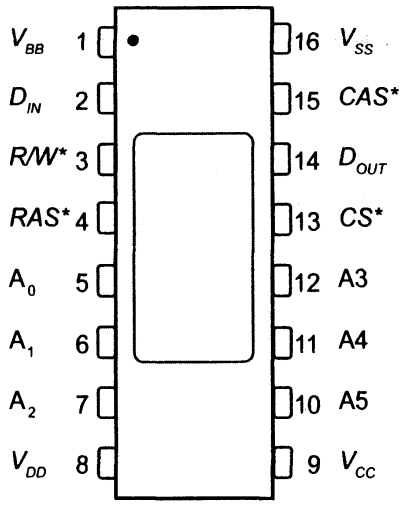


Figure 1.10 4,096-bit DRAM pin connections.

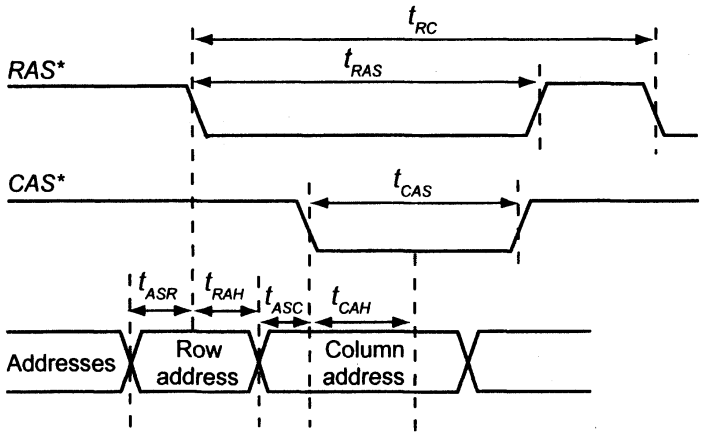


Figure 1.11 Address timing.

1.1.2.2 Multiple Memory Arrays. As mentioned earlier, second-generation DRAMs began to use multiple or segmented memory arrays. The main reason for splitting up the memory into more than one array at the cost of a larger layout area can be understood by considering the parasitics present in the dynamic memory circuit element. To understand the origins of these parasitics, consider the modern DRAM memory cell comprising one MOSFET and one capacitor, as shown in Figure 1.12.

In the next section, we cover the operation of this cell in detail. Here we introduce the operation of the cell. Data is written to the cell by driving the rowline (a.k.a., wordline) HIGH, turning ON the MOSFET, and allowing the columnline (a.k.a., digitline or bitline) to charge or discharge the storage capacitor. After looking at this circuit for a moment, we can make the following observation.

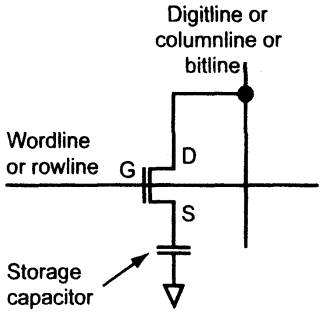


Figure 1.12 1-transistor, 1-capacitor (1T1C) memory cell.

1. The wordline (rowline) may be fabricated using polysilicon (poly). This allows the MOSFET to be formed by crossing the poly wordline over an n+ active area.
2. To write a full V_{CC} logic voltage (where V_{CC} is the maximum positive power supply voltage) to the storage capacitor, the rowline must be driven to a voltage greater than V_{CC} + the n-channel MOSFET threshold voltage (with body effect). This voltage, $> V_{CC} + V_{TH}$, is often labeled V_{CC} *pumped* (V_{CCP}).
3. The bitline (columnline) may be made using metal or polysilicon. The main concern, as we'll show in a moment, is to reduce the parasitic capacitance associated with the bitline.

Consider the row of N dynamic memory elements shown in Figure 1.13. Typically, in a modern DRAM, N is 512, which is also the number of bitlines. When a row address is strobed into the DRAM, via the address input pins using the falling edge of *RAS**, the address is decoded to drive a wordline (rowline) to V_{CCP}. *This turns ON an entire row in a DRAM memory array.* Turning ON an entire row in a DRAM memory array allows the information stored on the capacitors to be sensed (for a Read) via the bitlines or allows the charging or discharging, via the bitlines, of the storage capacitors (for a Write). Opening a row of data by driving a wordline HIGH is a ***very important*** concept for understanding the modes of DRAM operation. For Refresh, we only need to supply row addresses during a Refresh operation. For page Reads—when a row is open—a large amount of data, which is set by the number of columns in the DRAM array, can be accessed by simply changing the column address.

We're now in a position to answer the question: "Why are we limited to increasing the number of columnlines (or bitlines) used in a memory array?" or "Why do we need to break up the memory into smaller memory arrays?" The answer to these questions comes from the realization that the

more bitlines we use in an array, the longer the delay through the wordline (Figure 1.13).

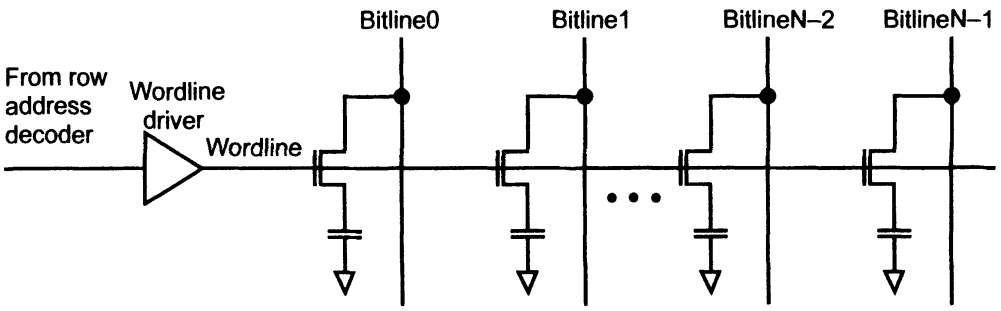


Figure 1.13 Row of N dynamic memory elements.

If we drive the wordline on the left side of Figure 1.13 HIGH, the signal will take a finite time to reach the end of the wordline (the wordline on the right side of Figure 1.13). This is due to the distributed resistance/capacitance structure formed by the resistance of the polysilicon wordline and the capacitance of the MOSFET gates. The delay limits the speed of DRAM operation. To be precise, it limits how quickly a row can be opened and closed. To reduce this *RC* time, a polycide wordline is formed by adding a silicide, for example, a mixture of a refractory metal such as tungsten with polysilicon, on top of polysilicon. Using a polycide wordline will have the effect of reducing the wordline resistance. Also, additional drivers can be placed at different locations along the wordline, or the wordline can be *stitched* at various locations with metal.

The limitations on the additional number of wordlines can be understood by realizing that by adding more wordlines to the array, more parasitic capacitance is added to the bitlines. This parasitic capacitance becomes important when sensing the value of data charge stored in the memory element. We'll discuss this in more detail in the next section.

1.1.2.3 Memory Array Size. A comment is in order about memory array size and how addressing can be used for setting word and page size. (We'll explain what this means in a moment.) If we review the block diagram of the 4k DRAM shown in Figure 1.9, we see that two 2k-DRAM memory arrays are used. Each 2k memory is composed of 64 wordlines and 32 bitlines for 2,048 memory elements/address locations per array. In the block diagram, notice that a single bit, coming from the column decoder, can be used to select data, via the bitlines, from Array0 or Array1.

From our discussion earlier, we can open a row in Array0 while at the same time opening a row in Array1 by simply applying a row address to the input address pins and driving *RAS** LOW. Once the rows are open, it is a simple matter of changing the column address to select different data asso-

ciated with the same open row from either array. If our word size is 1 bit, we could define a page as being 64 bits in length (32 bits from each array). We could also define our page size as 32 bits with a 2-bit word for input/output. We would then say that the DRAM is a 4k DRAM organized as 2k x 2. Of course, in the 4k DRAM, in which the number of bits is small, the concepts of page reads or size aren't too useful. We present them here simply to illustrate the concepts. Let's consider a more practical and modern configuration.

Suppose we have a 64-Meg DRAM organized as 16 Meg x 4 (4 bits input/output) using 4k row address locations and 4k column address locations (12 bits or pins are needed for each 4k of addressing). If our (sub) memory array size is 256kbits, then we have a total of 256 memory arrays on our DRAM chip. We'll assume that there are 512 wordlines and 512 bitlines (digitlines), so that the memory array is logically square. (However, physically, as we shall see, the array is not square.) Internal to the chip, in the address decoders, we can divide the row and column addresses into two parts: the lower 9 bits for addressing the wordlines/bitlines in a 256k memory array and the upper 3 bits for addressing one of the 64 "group-of-four" memory arrays (6 bits total coming from the upper 3 bits of the row and column addresses).

Our 4-bit word comes from the group-of-four memory arrays (one bit from each memory array). We can define a page of data in the DRAM by realizing that when we open a row in each of the four memory arrays, we are accessing 2k of data (512 bits/array x 4 arrays). By simply changing the column address without changing the row address and thus opening another group-of-four wordlines, we can access the 2k "page" of data. With a little imagination, we can see different possibilities for the addressing. For example, we could open 8 group-of-four memory arrays with a row address and thus increase the page size to 16k, or we could use more than one bit at a time from an array to increase word size.

1.1.2.4 Refreshing the DRAM. Refreshing the DRAM is accomplished by sequentially opening each row in the DRAM. (We'll discuss how the DRAM cell is refreshed in greater detail later in the book.) If we use the 64-Meg example in the last section, we need to supply 4k row addresses to the DRAM by changing the external address inputs from 000000000000 to 111111111111 while clocking the addresses into the DRAM using the falling edge of *RAS**. In some DRAMs, an internal row address counter is present to make the DRAM easier to refresh. The general specification for 64-Meg DRAM Refresh is that all rows must be refreshed at least every 64 ms, which is an average of 15.7 μs per row. This means, that if the Read cycle time t_{RC} is 100 ns (see Figure 1.11), it will take 4,096 • 100 ns or

410 μs to refresh a DRAM with 4k of row addresses. The percentage of time the DRAM is unavailable due to Refresh can be calculated as 410 μs/64 ms or 0.64% of the time. Note that the Refresh can be a burst, taking 410 μs as just described, or distributed, where a row is refreshed every 15.7 μs.

1.1.2.5 Modes of Operation. From the last section, we know that we can open a row in one or more DRAM arrays concurrently, allowing a page of data to be written to or read from the DRAM. In this section, we look at the different modes of operation possible for accessing this data via the column address decoder. Our goal in this section is not to present all possible modes of DRAM operation but rather to discuss the modes that have been used in second-generation DRAMs. These modes are page mode, nibble mode, static column mode, fast page mode, and extended data out.

Figure 1.14 shows the timing diagram for a age mode Read, Write, and Read-Modify-Write. We can understand this timing diagram by first noticing that when *RAS** goes LOW, we clock in a row address, decode the row address, and then drive a wordline in one or more memory arrays to V_{CCP}. The result is an open row(s) of data sitting on the digitlines columnlines). *Only one row can be opened in any single array at a time.* Prior to opening a row, the bitlines are precharged to a known voltage. (Precharging to $V_{CC}/2$ is typically performed using internal circuitry.) Also notice at this time that data out, D_{OUT}, is in a Hi-Z state; that is, the DRAM is not driving the bus line connected to the D_{OUT} pin.

The next significant timing event occurs when *CAS** goes LOW and the column address is clocked into the DRAM (Figure 1.14). At this time, the column address is decoded, and, assuming that the data from the open row is sitting on the digitlines, it is steered using the column address decoder to D_{OUT}. We may have an open row of 512 bits, but we are steering only one bit to D_{OUT}. Notice that when *CAS** goes HIGH, D_{OUT} goes back to the Hi-Z state.

By strobing in another column address with the same open row, we can select another bit of data (again via the column address decoder) to steer to the D_{OUT} pin. In this case, however, we have changed the DRAM to the Write mode (Figure 1.14). This allows us to write, with the same row open via the D_{IN} pin in Figure 1.10, to any column address on the open row. *Later, second-generation DRAMs used the same pins for both data input and output to reduce pin count. These bidirectional pins are labeled DQ.*

The final set of timing signals in Figure 1.14 (the right side) read data out of the DRAM with *R/W** HIGH, change *R/W** to a LOW, and then write to the same location. Again, when *CAS** goes HIGH, D_{OUT} goes back to the Hi-Z state.

The remaining modes of operation are simple modifications of page mode. As seen in Figure 1.15, FPM allows the column address to propagate into the column circuits while *CAS** is HIGH. The speed of the DRAM thus improves by reducing the delay between *CAS** going LOW and valid data present, or accessed, on D_{OUT} (t_{CAC}). EDO is simply an FPM DRAM that doesn't force D_{OUT} to a Hi-Z state immediately when *CAS** goes HIGH. The data out of the DRAM is thus available for a longer period of time, allowing for faster system operation. In general, opening the row is the operation that takes the longest amount of time. Once a row is open, the data sitting on the columnlines can be steered to D_{OUT} at a fast rate. Interestingly, using column access modes has been the primary method of boosting DRAM performance over the years, especially in double-data rate (DDR) DRAMs (see Chapter 8).

The other popular modes of operation in second-generation DRAMs were the static column and nibble modes. Static column mode DRAMs used flow-through latches in the column address path. When a column address was changed externally, with *CAS** LOW, the column address fed directly to the column address decoder. (The address wasn't clocked on the falling edge of *CAS**.) This increased the speed of the DRAM by preventing the outputs from going into the Hi-Z state with changes in the column address.

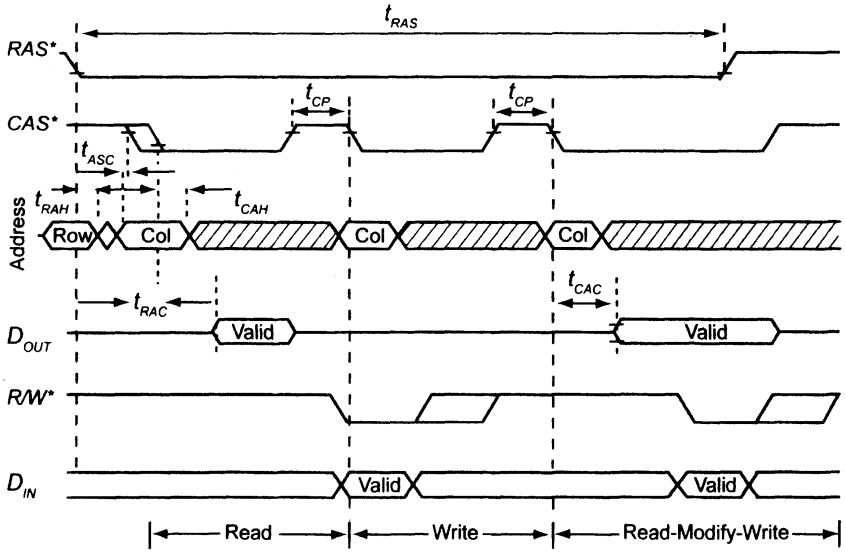


Figure 1.14 Page mode.

Nibble mode DRAMs used an internal presettable address counter so that by strobing *CAS**, the column address would change internally. Figure 1.16 illustrates the timing operation for a nibble mode DRAM. The first time *CAS** transitions LOW (first being defined as the first transition after *RAS** goes LOW), the column address is loaded into the counter. If *RAS** is

held LOW and *CAS** is toggled, the internal address counter is incremented, and the sequential data appears on the output of the DRAM. The term *nibble mode* comes from limiting the number of *CAS** cycles to four (a nibble).

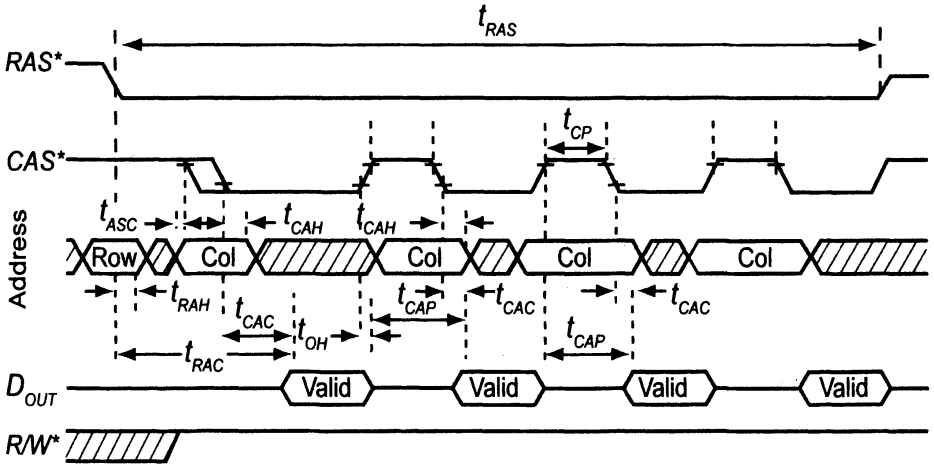


Figure 1.15 Fast page mode.

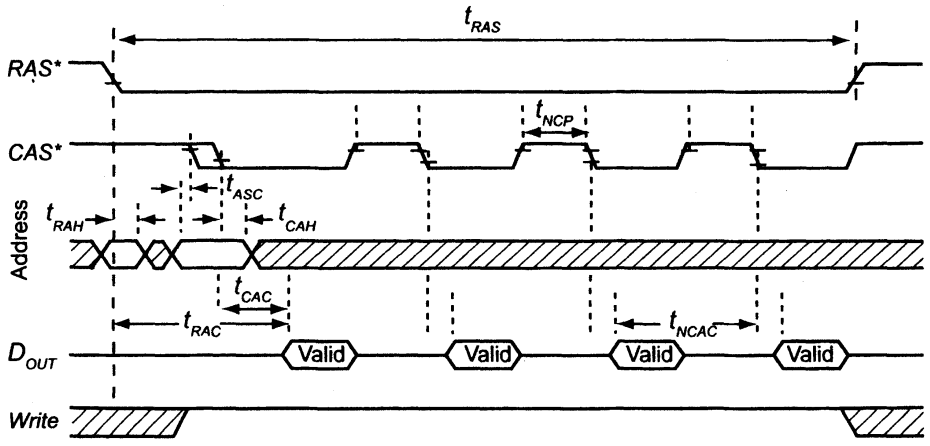


Figure 1.16 Nibble mode.

1.1.3 Synchronous DRAM (Third Generation)

Synchronous DRAMs (SDRAMs) are made by adding a synchronous interface between the basic core DRAM operation/circuitry of second-generation DRAMs and the control coming from off-chip to make the DRAM operation faster. All commands and operations to and from the DRAM are executed on the rising edge of a master or command clock signal that is common to all SDRAMs and labeled *CLK*. See Figure 1.17 for the pin connections of a 64Mb SDRAM with 16-bit input/output (*I/O*).

SDRAMs operate with a maximum *CLK* frequency in the range of 100–167 MHz, clocking data on one edge of the clock. This means that if a 64Mb SDRAM is organized as a x16 part (that is, the input/output word size is 16 bits), the maximum rate at which the words can be written to the part is 200–334 MB/s.

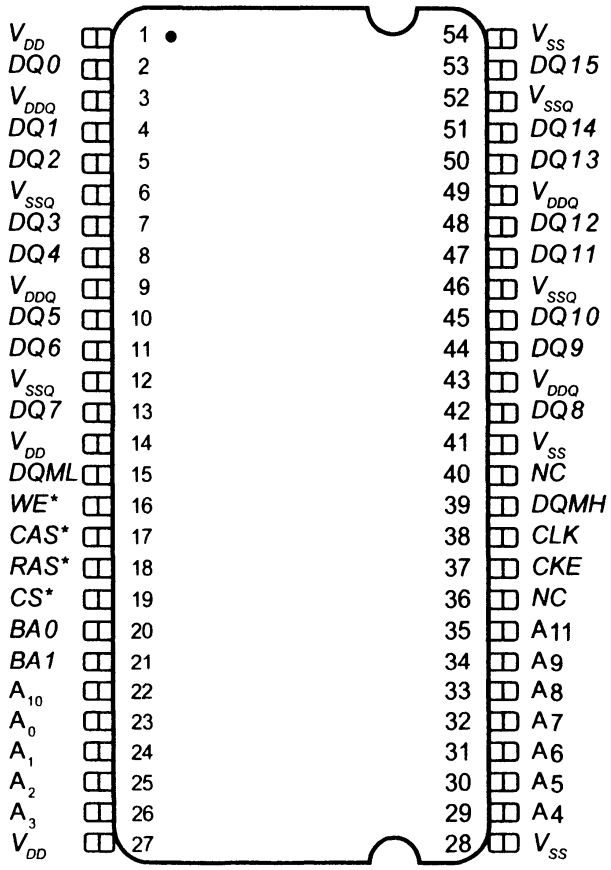


Figure 1.17 Pin connections of a 64Mb SDRAM with 16-bit *I/O*.

A variation of the SDRAM is the *double-data-rate SDRAM* (DDR SDRAM, or simply DDR DRAM). The DDR parts register commands and operations on the rising edge of the clock signal while allowing data to be transferred on both the rising and falling edges. A differential input clock signal is used in the DDR DRAM with the labeling of, not surprisingly, *CLK* and *CLK**. In addition, the DDR DRAM provides an output data strobe, labeled DQS, synchronized with the output data and the input *CLK*. DQS is used at the controller to strobe in data from a DRAM. The big benefit of using a DDR part is that the data transfer rate can be twice the clock frequency because data can be transferred on both the rising and falling edges of *CLK*. This means that when using a 133 MHz clock, the data written to and read from the DRAM can be transferred at 266M words/s. Using

the numbers from the previous paragraph, this means that a 64Mb DDR SDRAM with an input/output word size of 16 bits will transfer data to and from the memory controller at 400–572 MB/s.

Figure 1.18 shows the block diagram of a 64Mb SDRAM with 16-bit *I/O*. Note that although *CLK* is now used for transferring data, we still have the second-generation control signals *CS**, *WE**, *CAS**, and, *RAS** present on the part. (*CKE* is a clock enable signal which, unless otherwise indicated, is assumed HIGH.) Let's discuss how these control signals are used in an SDRAM by recalling that in a second-generation DRAM, a Write was executed by first driving *WE** and *CS** LOW. Next a row was opened by applying a row address to the part and then driving *RAS** LOW. (The row address is latched on the falling edge of *RAS**.) Finally, a column address was applied and latched on the falling edge of *CAS**. A short time later, the data applied to the part would be written to the accessed memory location.

For the SDRAM Write, we change the syntax of the descriptions of what's happening in the part. However, the fundamental operation of the DRAM circuitry is the same as that of the second-generation DRAMs. We can list these syntax changes as follows:

1. The memory is segmented into banks. For the 64Mb memory of Figure 1.17 and Figure 1.18, each bank has a size of 16Mbs (organized as 4,096 row addresses [12 bits] x 256 column addresses [8 bits] x 16 bits [16 *DQ I/O* pins]). As discussed earlier, this is nothing more than a simple logic design of the address decoder (the banks can be laid out so that they are physically in the same area). The bank selected is determined by the addresses *BA*0 and *BA*1.
2. In second-generation DRAMs, we said, "We open a row," as discussed earlier. In SDRAM, we now say, "We activate a row in a bank." We do this by issuing an active command to the part. Issuing an active command is accomplished on the rising edge of *CLK* with a row/bank address applied to the part with *CS** and *RAS** LOW, while *CAS** and *WE** are held HIGH.
3. In second-generation DRAMs, we said, "We write to a location given by a column address," by driving *CAS** LOW with the column address applied to the part and then applying data to the part. In an SDRAM, we write to the part by issuing the Write command to the part. Issuing a Write command is accomplished on the rising edge of *CLK* with a column/bank address applied to the part: *CS**, *CAS**, and *WE** are held LOW, and *RAS** is held HIGH.

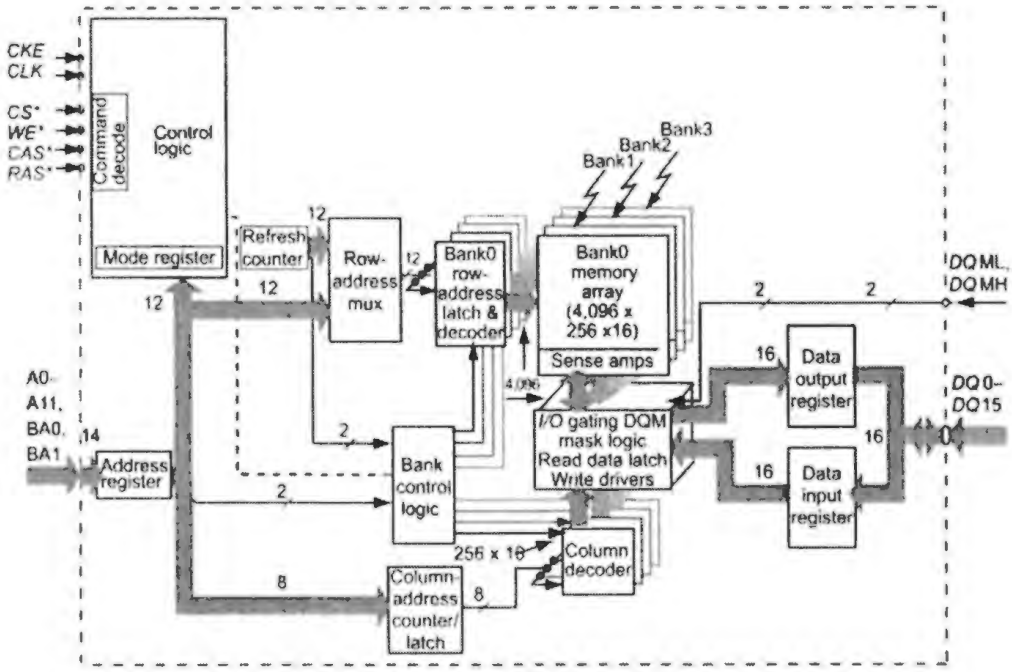


Figure 1.18 Block diagram of a 64Mb SDRAM with 16-bit *I/O*.

Table 1.1 shows the commands used in an SDRAM. In addition, this table shows how inputs/outputs *(DQs)* can be masked using the *DQ* mask (DQM) inputs. This feature is useful when the DRAM is used in graphics applications.

SDRAMs often employ pipelining in the address and data paths to increase operating speed. Pipelining is an effective tool in SDRAM design because it helps disconnect operating frequency and access latency. Without pipelining, a DRAM can only process one access instruction at a time. Essentially, the address is held valid internally until data is fetched from the array and presented to the output buffers. This single instruction mode of operation ties operating frequency and access time (or latency) together. However, with pipelining, additional access instructions can be fed into the SDRAM before prior access instructions have completed, which permits access instructions to be entered at a higher rate than would otherwise be allowed. Hence, pipelining increases operating speed.

Table 1.1 SDRAM commands. (Notes: 1)

Name	*CS**	*RAS**	*CAS**	*WE**	DQM	ADDR	*DQ*s	Notes
Command inhibit (NOP)	H	X	X	X	X	X	X	—
No operation (NOP)	L	H	H	H	X	X	X	—
Active (select bank and activate row)	L	L	H	H	X	Bank/ row	X	3
Read (select bank and column, and start Read burst)	L	H	L	H	L/H[8]	Bank/ col	X	4
Write (select bank and column, and start Write burst)	L	H	L	L	L/H[8]	Bank/ col	Valid	4
Burst terminate	L	H	H	L	X	X	Activ e	—
PRECHARGE (deactive row in bank or banks)	L	L	H	L	X	Code	X	5
Auto-Refresh ors self-refresh (enter self-refresh mode)	L	L	L	H	X	X	X	6, 7
Load mode register	L	L	L	L	X	Op-code	X	2
Write ENABLE/ output ENABLE	—	—	—	—	L	—	Activ e	8
Write inhibit/ output Hi-Z	—	—	—	—	H	—	Hi-Z	8

Notes

1. *CKE* is HIGH for all commands shown except for self-refresh.
2. *A*0–*A*11 define the op-code written to the mode register.
3. *A*0–*A*11 provide row address, and *BA*0, *BA*1 determine which bank is made active.
4. *A*0–*A*9 (x4), *A*0–*A*8 (x8), or *A*0–*A*7 (x16) provide column address; *A*10 HIGH enables the auto *PRECHARGE* feature (nonpersistent), while *A*10 LOW disables the auto *PRECHARGE* feature; *BA*0, *BA*1 determine which bank is being read from or written to.
5. *A*10 LOW: *BA*0, *BA*1 determine the bank being precharged. *A*10 HIGH: all banks precharged and *BA*0, *BA*1 are "don't care."
6. This command is Auto-Refresh if *CKE* is HIGH and Self-Refresh if *CKE* is LOW.

7. Internal Refresh counter controls row addressing; all inputs and *I/Os* are "don't care" except for *CKE*.
8. Activates or deactivates the *DQs* during Writes (zero-clock delay) and Reads (two-clock delay).

Pipeline stages in the data path can also be helpful when synchronizing output data to the system clock. *CAS* latency refers to a parameter used by the SDRAM to synchronize the output data from a Read request with a particular edge of the system clock. A typical Read for an SDRAM with *CAS* latency set to three is shown in Figure 1.19. SDRAMs must be capable of reliably functioning over a range of operating frequencies while maintaining a specified *CAS* latency. This is often accomplished by configuring the pipeline stage to register the output data to a specific clock edge, as determined by the *CAS* latency parameter.

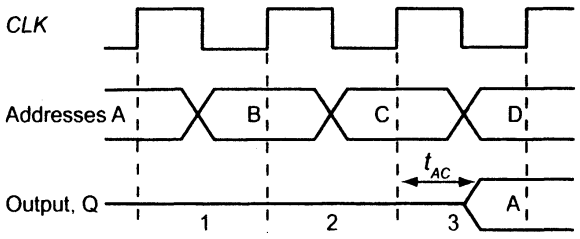


Figure 1.19 SDRAM with a latency of three.

At this point, we should understand the basics of SDRAM operation, but we may be asking, "Why are SDRAMs potentially faster than second-generation DRAMs such as EDO or FPM?" The answer to this question comes from the realization that it's possible to activate a row in one bank and then, while the row is opening, perform an operation in some other bank (such as reading or writing). In addition, one of the banks can be in a *PRECHARGE* mode (the bitlines are driven to $V_{CC}/2$) while accessing one of the other banks and, thus, in effect hiding *PRECHARGE* and allowing data to be continuously written to or read from the SDRAM. (Of course, this depends on which application and memory address locations are used.) We use a mode register, as shown in Figure 1.20, to put the SDRAM into specific modes of operation for programmable operation, including pipelining and burst Reads/Writes of data [2].

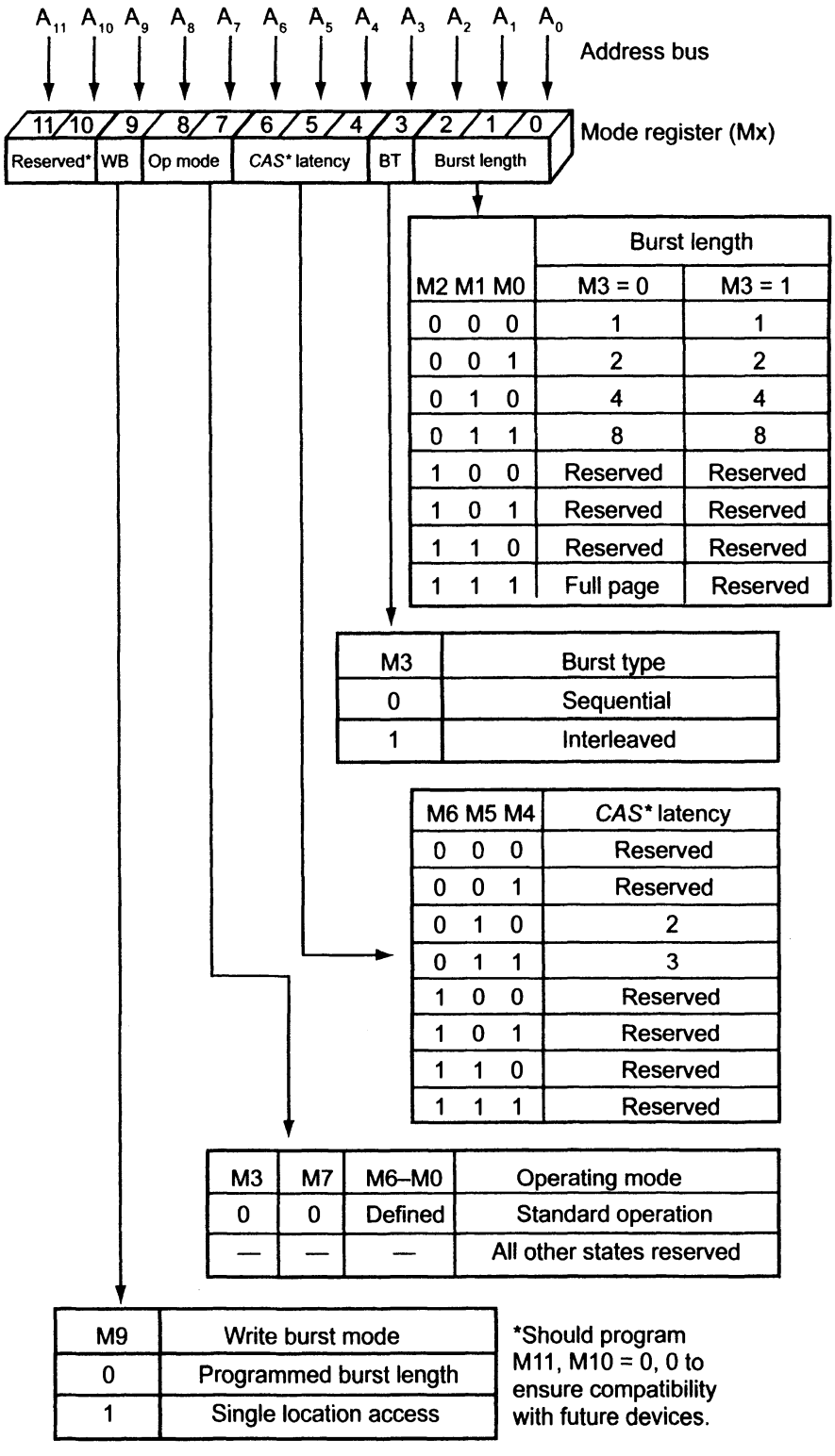


Figure 1.20 Mode register.

1.2 DRAM BASICS

A modern DRAM memory cell or memory bit (mbit), as shown in Figure 1.21, is formed with one transistor and one capacitor, accordingly referred to as a 1T1C cell. The mbit is capable of holding binary information in the form of stored charge on the capacitor. The mbit transistor operates as a switch interposed between the mbit capacitor and the digitline. Assume that the capacitor's common node is biased at $V_{CC}/2$, which we will later show as a reasonable assumption. Storing a logic one in the cell requires a capacitor with a voltage of $+V_{CC}/2$ across it. Therefore, the charge stored in the mbit capacitor is

$$Q = \frac{V_{CC}}{2} \cdot C \tag{1.1}$$

where C is the capacitance value in farads. Conversely, storing a logic zero in the cell requires a capacitor with a voltage of $-V_{CC}/2$ across it. Note that the stored charge on the mbit capacitor for a logic zero is

$$Q = \frac{-V_{CC}}{2} \cdot C \tag{1.2}$$

The charge is negative with respect to the $V_{CC}/2$ common node voltage in this state. Various leakage paths cause the stored capacitor charge to slowly deplete. To return the stored charge and thereby maintain the stored data state, the cell must be refreshed. The required refreshing operation is what makes DRAM memory dynamic rather than static.

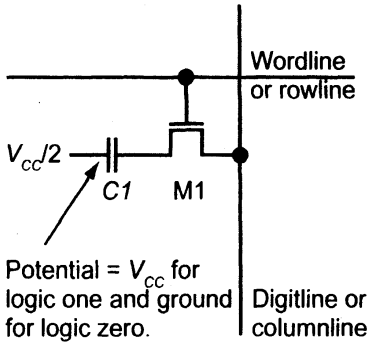


Figure 1.21 1T1C DRAM memory cell.
(Note the rotation of the rowline and columnline.)

The digitline referred to earlier consists of a conductive line connected to a multitude of mbit transistors. The conductive line is generally con-

structed from either metal or silicide/polycide polysilicon. Because of the quantity of mbits connected to the digitline and its physical length and proximity to other features, the digitline is highly capacitive. For instance, a typical value for digitline capacitance on a 50 nm process might be 120 fF. Digitline capacitance is an important parameter because it dictates many other aspects of the design. We discuss this further in Section 2.1. For now, we continue describing basic DRAM operation.

The mbit transistor gate terminal is connected to a wordline rowline). The wordline, which is connected to a multitude of mbits, is actually formed of the same polysilicon as that of the transistor gate. The wordline is physically orthogonal to the digitline. A memory array is formed by tiling a selected quantity of mbits together such that mbits along a given digitline do not share a common wordline and mbits along a common wordline do not share a common digitline. Examples of this are shown in Figures 1.22 and 1.23. In these layouts, mbits are paired to share a common contact to the digitline, which reduces the array size by eliminating duplication.

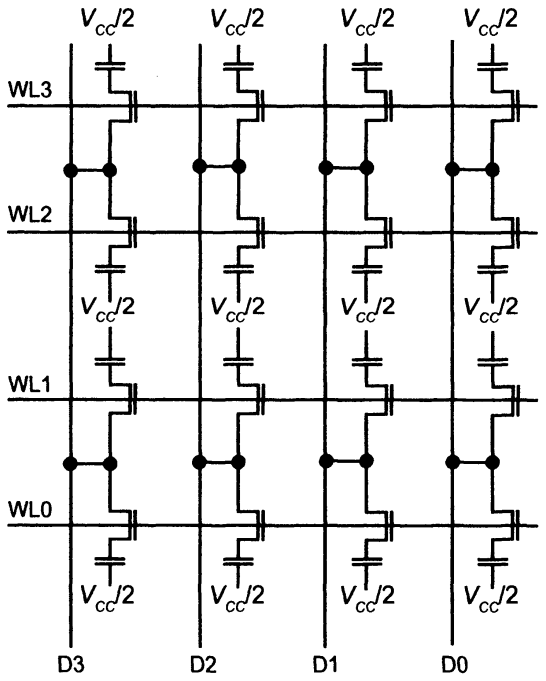


Figure 1.22 Open digitline memory array schematic.

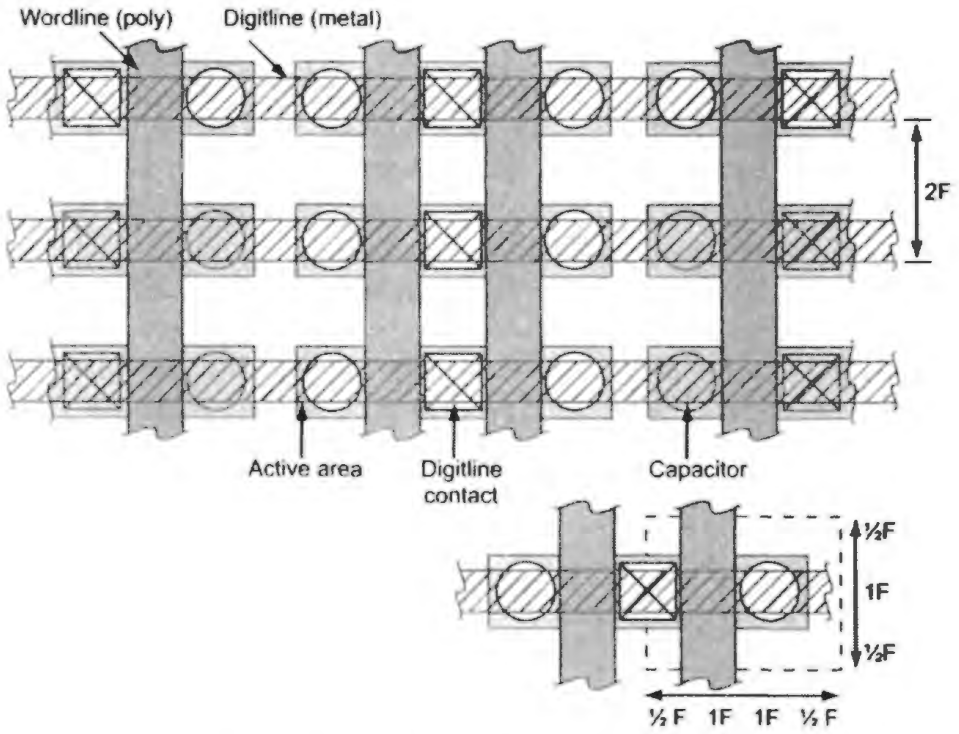


Figure 1.23 Open digitline memory array layout.

1.2.1 Access and Sense Operations

Next, we examine the access and sense operations. We begin by assuming that the cells connected to D1, in Figure 1.24, have logic one levels ($+V_{CC}/2$) stored on them and that the cells connected to D0 have logic zero levels ($-V_{CC}/2$) stored on them. Next, we form a digitline pair by considering two digitlines from adjacent arrays. The digitline pairs, labeled D0/D0* and D1/D1*, are initially equilibrated to $V_{CC}/2$ V. All wordlines are initially at 0 V, ensuring that the mbit transistors are OFF. Prior to a wordline firing, the digitlines are electrically disconnected from the $V_{CC}/2$ bias voltage and allowed to float. They remain at the $V_{CC}/2$ *PRECHARGE* voltage due to their capacitance.

To read mbit1, wordline WL0 changes to a voltage that is at least one transistor V_{TH} above V_{CC}. This voltage level is referred to as V_{CCP} or V_{PP}. To ensure that a full logic one value can be written back into the mbit capacitor, V_{CCP} must remain greater than one V_{TH} above V_{CC}. The mbit capacitor begins to discharge onto the digitline at two different voltage levels depending on the logic level stored in the cell. For a logic one, the capacitor begins to discharge when the wordline voltage exceeds the digitline *PRECHARGE* voltage by V_{TH}. For a logic zero, the capacitor begins to discharge when the wordline voltage exceeds V_{TH}. Because of the finite rise time of the word-

line voltage, this difference in turn-on voltage translates into a significant delay when reading ones, as seen in Figure 1.25.

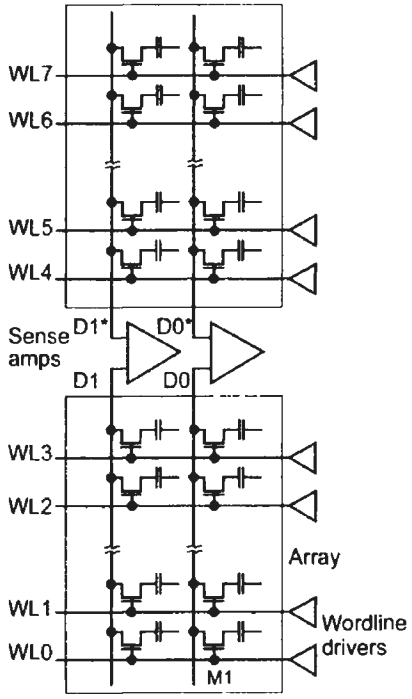


Figure 1.24 Simple array schematic (an open DRAM array).

Accessing a DRAM cell results in charge sharing between the mbit capacitor and the digitline capacitance. This charge sharing causes the digitline voltage either to increase for a stored logic one or to decrease for a stored logic zero. Ideally, only the digitline connected to the accessed mbit will change. In reality, the other digitline voltage also changes slightly, due to parasitic coupling between digitlines and between the firing wordline and the other digitline. (This is especially true for the folded bitline architecture discussed later.) Nevertheless, a differential voltage develops between the two digitlines. The magnitude of this voltage difference, or signal, is a function of the *mbit capacitance* (C_{mbit}), *digitline capacitance* (C_{digit}), and voltage stored on the cell prior to access (V_{cell}). See Figure 1.26. Accordingly,

$$V_{signal} = V_{cell} \cdot \frac{C_{mbit}}{C_{digit} + C_{mbit}} \tag{1.3}$$

A V_{signal} of 200 mV is yielded from a design in which V_{cell} = 1.00, C_{mbit} = 20 fF, and C_{digit} = 200 fF.

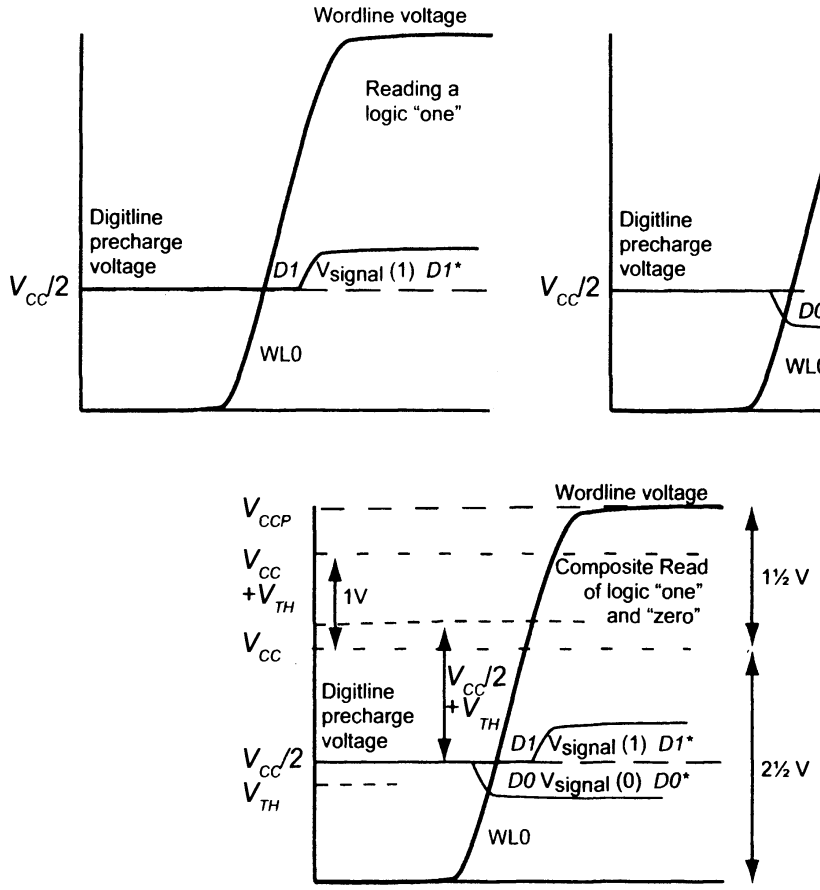


Figure 1.25 Cell access waveforms.

After the cell has been accessed, sensing occurs. Sensing is essentially the amplification of the digitline signal or the differential voltage between the digitlines. Sensing is necessary to properly read the cell data and refresh the mbit cells. (The reason for forming a digitline pair now becomes apparent.) presents a schematic diagram for a simplified sense amplifier circuit: a cross-coupled NMOS pair and a cross-coupled PMOS pair. The sense amplifiers also appear like a pair of cross-coupled inverters in which *ACT* and *NLAT** provide power and ground. The NMOS pair or Nsense-amp has a common node labeled *NLAT** (for *Nsense-amp latch*).

Similarly, the Psense-amp has a common node labeled *ACT* (for *Active pull-up*). Initially, *NLAT** is biased to $V_{CC}/2$, and *ACT* is biased to V_{SS} or signal ground. Because the digitline pair D1 and D1* are both initially at $V_{CC}/2$, the Nsense-amp transistors are both OFF. Similarly, both Psense-amp transistors are OFF. Again, when the mbit is accessed, a signal develops across the digitline pair. While one digitline contains charge from the cell access, the other digitline does not but serves as a reference for the Sensing operation. The sense amplifiers are generally fired sequentially: the Nsense-amp first, then the Psense-amp. Although designs vary at this point,

the higher drive of NMOS transistors and better V_{TH} matching offer better sensing characteristics by Nsense-amps and thus lower error probability compared to Psense-amps.

Waveforms for the Sensing operation are shown in Figure 1.28. The Nsense-amp is fired by bringing *NLAT** (Nsense-amp latch) toward ground. As the voltage difference between *NLAT** and the digitlines (D1 and D1* in) approaches V_{TH}, the NMOS transistor, whose gate is connected to the higher voltage digitline, begins to conduct. This conduction occurs first in the subthreshold and then in the saturation region as the gate-to-source voltage exceeds V_{TH} and causes the low-voltage digitline to discharge toward the *NLAT** voltage. Ultimately, *NLAT** will reach ground and the digitline will be brought to ground potential. Note that the other NMOS transistor will not conduct: its gate voltage is derived from the low-voltage digitline, which is being discharged toward ground. In reality, parasitic coupling between digitlines and limited subthreshold conduction by the second transistor result in a temporary voltage drop on the high digitline, as seen in Figure 1.28.

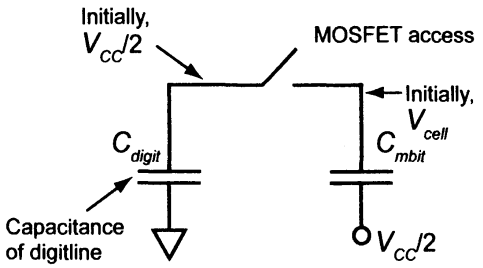


Figure 1.26 DRAM charge sharing..

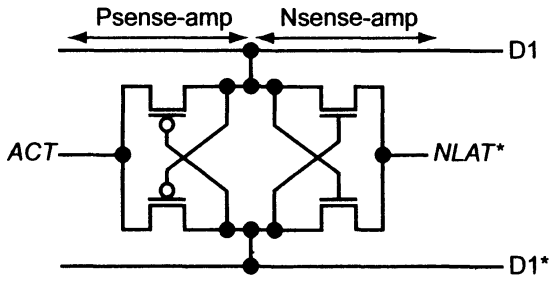


Figure 1.27 Sense amplifier schematic.

Sometime after the Nsense-amp fires, *ACT* will be brought toward V_{CC} to activate the Psense-amp, which operates in a complementary fashion to the Nsense-amp. With the low-voltage digitline approaching ground, there is a strong signal to drive the appropriate PMOS transistor into conduction. This conduction, again moving from subthreshold to saturation, charges the

high-voltage digitline toward *ACT*, ultimately reaching V_{CC}. Because the mbit transistor remains ON, the mbit capacitor is refreshed during the Sensing operation. The voltage, and hence charge, which the mbit capacitor held prior to accessing, is restored to a full level: V_{CC} for a logic one and ground for a logic zero. It should be apparent now why the minimum wordline voltage is a V_{TH} above V_{CC}. If V_{CCP} were anything less, a full V_{CC} level could not be written back into the mbit capacitor. The mbit transistor source voltage V_{source} cannot be greater than $V_{gate} - V_{TH}$ because this would turn OFF the transistor.

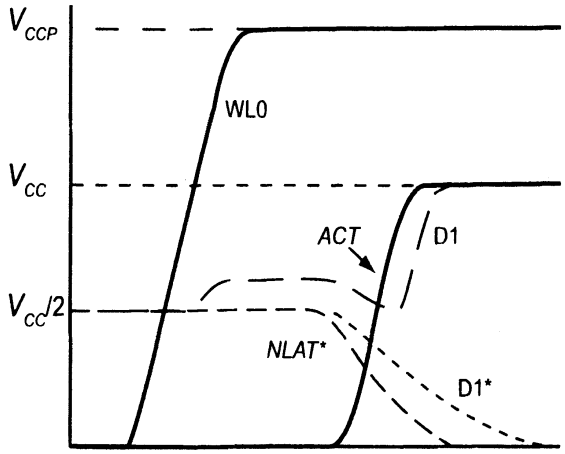


Figure 1.28 Sensing operation waveforms.

1.2.2 Write Operation

A Write operation is similar to a Sensing and Restore operation except that a separate Write driver circuit determines the data that is placed into the cell. The Write driver circuit is generally a tristate inverter connected to the digitlines through a second pair of pass transistors, as shown in Figure 1.29. These pass transistors are referred to as *I/O* transistors. The gate terminals of the *I/O* transistors are connected to a common *column select* (*CSEL*) signal. The *CSEL* signal is decoded from the column address to select which pair (or multiple pairs) of digitlines is routed to the output pad or, in this case, the Write driver.

In most current DRAM designs, the Write driver simply overdrives the sense amplifiers, which remain ON during the Write operation. After the new data is written into the sense amplifiers, the amplifiers finish the Write cycle by restoring the digitlines to full rail-to-rail voltages. An example is shown in Figure 1.30 in which D1 is initially HIGH after the Sensing operation and LOW after the writing operation. A Write operation usually involves only 2–4 mbits within an array of mbits because a single *CSEL* line is generally connected to only four pairs of *I/O* transistors. The remain-

ing digitlines are accessed through additional *CSEL* lines that correspond to different column address locations.

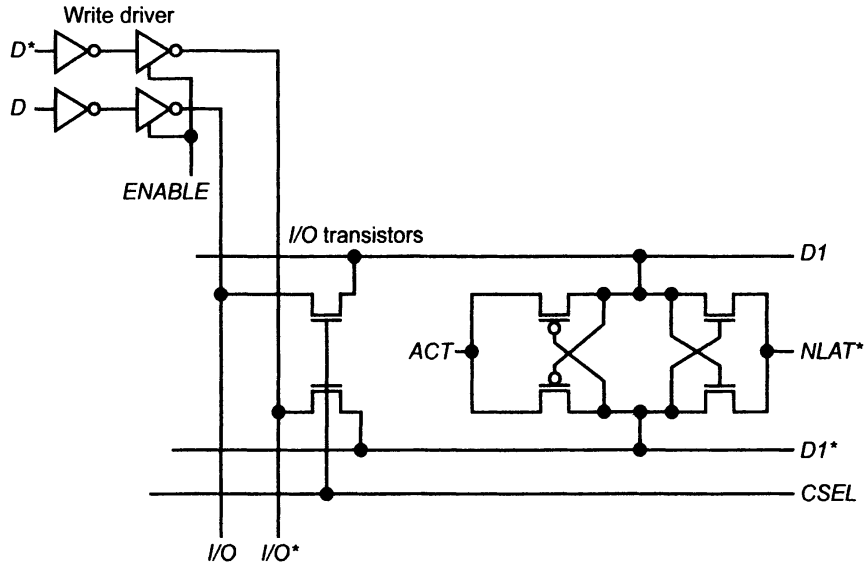


Figure 1.29 Sense amplifier schematic with *I/O* devices.

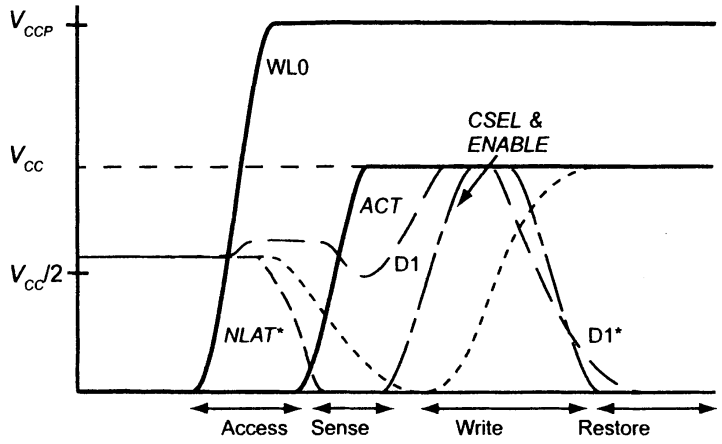


Figure 1.30 Write operation waveforms.

1.2.3 Opening a Row (Summary)

Opening a row of mbits in a DRAM array is a fundamental operation for both reading and writing to the DRAM array. Sometimes the chain of events from a circuit designer's point of view, which lead to an open row, is called the *RAS** timing chain. We summarize the *RAS** timing chain of events below, assuming that for a second-generation DRAM both *RAS** and *CAS** are HIGH. (It's easy to extend our discussion to third-generation DRAMs where *RAS** and *CAS** are effectively generated from the control logic.)

1. Initially, both *RAS** and *CAS** are HIGH. All bitlines in the DRAM are driven to $V_{CC}/2$, while all wordlines are at 0 V. This ensures that all of the mbit's access transistors in the DRAM are OFF.
2. A valid row address is applied to the DRAM and *RAS** goes LOW. While the row address is being latched, on the falling edge of *RAS**, and decoded, the bitlines are disconnected from the $V_{CC}/2$ bias and allowed to float. The bitlines at this point are charged to $V_{CC}/2$, and they can be thought of as capacitors.
3. The row address is decoded and applied to the wordline drivers. This forces only one rowline in at least one memory array to V_{CCP}. Driving the wordline to V_{CCP} turns ON the mbits attached to this rowline and causes charge sharing between the mbit capacitance and the capacitance of the corresponding bitline. The result is a small perturbation (upwards for a logic one and downwards for a logic zero) in the bitline voltages.
4. The next operation is Sensing, which has two purposes: a) to determine if a logic one or zero was written to the cell and b) to refresh the contents of the cell by restoring a full logic zero (0 V) or one (V_{CC}) to the capacitor. Following the wordlines going HIGH, the Nsense-amp is fired by driving, via an n-channel MOSFET, *NLAT** to ground. The inputs to the sense amplifier are two bitlines: the bitline we are sensing and the bitline that is not active (a bitline that is still charged to $V_{CC}/2$—an inactive bitline). Pulling *NLAT** to ground results in one of the bitlines going to ground. Next, the *ACT* signal is pulled up to V_{CC}, driving the other bitline to V_{CC}. Some important notes:

   (a) It doesn't matter if a logic one or logic zero was sensed because the inactive and active bitlines are pulled in opposite directions.

   (b) The contents of the active cell, after opening a row, are restored to full voltage levels (either 0 V or V_{CC}). The entire DRAM can be refreshed by opening each row.

Now that the row is open, we can write to or read from the DRAM. In either case, it is a simple matter of steering data to or from the active array(s) using the column decoder. When writing to the array, buffers set the new logic voltage levels on the bitlines. The row is still open because the wordline remains HIGH. (The row stays open as long as *RAS** is LOW.)

When reading data out of the DRAM, the values sitting on the bitlines are transmitted to the output buffers via the *I/O* MOSFETs. To increase the speed of the reading operation, this data, in most situations, is transmitted to

the output buffer (sometimes called a *DQ* buffer) either through a helper flip-flop or another sense amplifier.

A note is in order here regarding the word size stored in or read out of the memory array. We may have 512 active bitlines when a single rowline in an array goes HIGH (keeping in mind once again that only one wordline in an array can go HIGH at any given time). This literally means that we could have a word size of 512 bits from the active array. By adjusting the word size, or the number of these 512 bits we send to the chip's output, we can change the speed on power performance of the chips. For example, if our chip's *I/O* is x4 (by 4, that is, 4-bit input/output), then sending eight bits at a time (out of these 512) provides data for two clock edges. Sending 32 bits provides data for eight clock edges. The cost of transmitting larger word sizes is larger bus widths (more layout area). However, the benefit is that more time is available to send the data once it is in the pipeline. The output word size is the basic difference between the DDR, DDR2, and DDR3 DRAM topologies.

1.2.4 Open/Folded DRAM Array Architectures

Throughout the book, we make a distinction between the open array architecture as shown in Figures 1.22 and 1.24 and the folded DRAM array used in many modern DRAMs and seen in Figure 1.31. At the cost of increased layout area, folded arrays increase noise immunity by moving sense amp inputs next to each other. These sense amp inputs come directly from the DRAM array. The term *folded* comes from taking the DRAM arrays seen in Figure 1.24 and folding them together to form the topology seen in Figure 1.31.

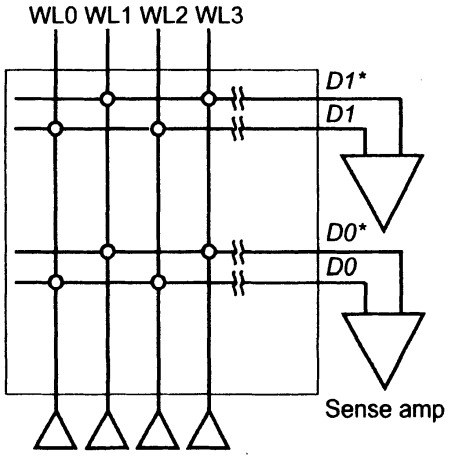


Figure 1.31 A folded DRAM array.

REFERENCES

[1] R. J. Baker, *CMOS: Circuit Design, Layout, and Simulation*, 2nd ed. Hoboken, NJ: John Wiley & Sons, Inc., and IEEE Press, 2005.

[2] Micron Technology, Inc., Synchronous DRAM datasheet, 1999.

Chapter
2

The DRAM Array

This chapter begins a more detailed examination of standard DRAM array elements. This examination is necessary for a clear understanding of fundamental DRAM elements and how they are used in memory block construction. A common point of reference is required before considering the analysis of competing array architectures. Included in this chapter is a detailed discussion of mbits, array configurations, sense amplifier elements, and row decoder elements.

2.1 THE MBIT CELL

The primary advantage of DRAM over other types of memory technology is low cost. This advantage arises from the simplicity and scaling characteristics of its 1T1C memory cell [1]. Although the DRAM mbit is simple conceptually, its actual design and implementation are highly complex. Therefore, successful, cost-effective DRAM designs require a tremendous amount of process technology.

Figure 2.1 presents the layout of a modern buried capacitor DRAM mbit pair. (*Buried* means that the capacitor is below the digitline.) This type of mbit is also referred to as a *bitline over capacitor* (BOC) cell. Because sharing a contact significantly reduces overall cell size, DRAM mbits are constructed in pairs. In this way, a digitline contact can be shared. The mbits comprise an active area rectangle (in this case, an n+ active area), a pair of polysilicon wordlines, a single digitline contact, a metal or polysilicon digitline, and a pair of cell capacitors formed with an oxide-nitride-oxide dielectric between two layers of polysilicon. For most processes, the wordline polysilicon is silicided to reduce sheet resistance, permitting longer wordline segments without reducing speed. The mbit layout, as shown in

Figure 2.1, is essentially under the control of process engineers, for every aspect of the mbit must meet stringent performance and yield criteria.

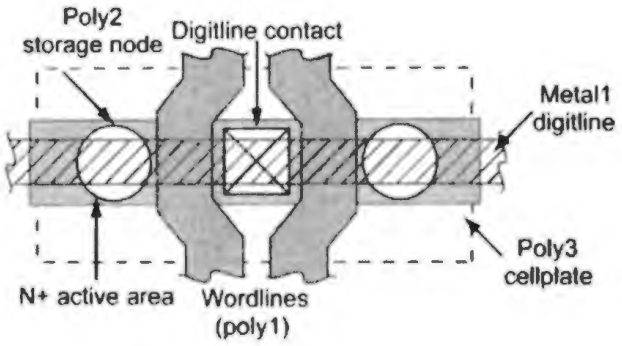


Figure 2.1 Mbit pair layout.

A small array of mbits appears in Figure 2.2. This figure is useful to illustrate several features of the mbit. First, note that the digitline pitch (width plus space) dictates the active area pitch and the capacitor pitch. Process engineers adjust the active area width and the field oxide width to maximize transistor drive and minimize transistor-to-transistor leakage. Field oxide technology greatly impacts this balance. A thicker field oxide or a shallower junction depth affords a wider transistor active area. Second, the wordline pitch (width plus space) dictates the space available for the digitline contact, transistor length, active area, field poly width, and capacitor length. Optimization of each of these features by process engineers is necessary to maximize capacitance, minimize leakage, and maximize yield. Contact technology, subthreshold transistor characteristics, photolithography, and etch and film technology dictate the overall design.

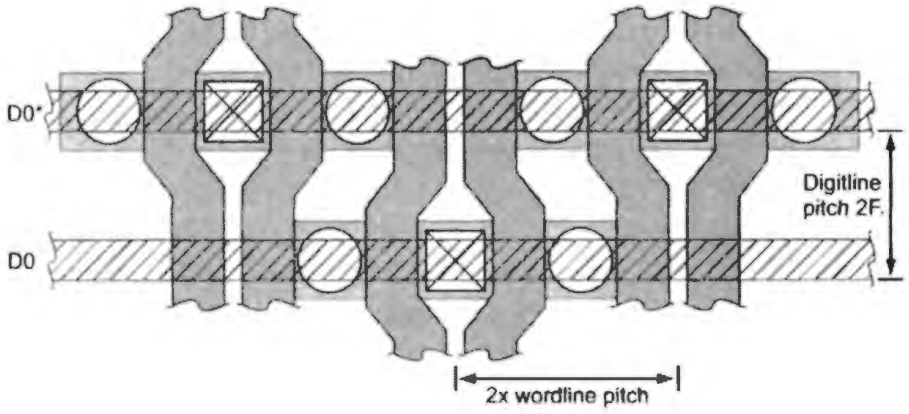


Figure 2.2 Layout to show array pitch.

At this point in the discussion, it is appropriate to introduce the concept of feature size and how it relates to cell size. The mbit shown in Figures 2.1

and 2.2 is by definition an eight-square feature cell ($8F^2$) [2][3]. The intended definition of feature (F) in this case is minimum realizable process dimension but in fact equates to a dimension that is one-half the wordline (row) or digitline (column) pitch. A 45 nm process having wordline and digitline pitches of 100 nm yields an mbit size that is

$$(8) \cdot 50nm = 0.02\mu m^2 \tag{2.1}$$

It is easier to explain the $8F^2$ designation with the aid of Figure 2.3. An imaginary box drawn around the mbit defines the cell's outer boundary. Along the x-axis, this box includes one-half digitline contact feature, one wordline feature, one capacitor feature, one field poly feature, and one-half poly space feature for a total of four features. Along the y-axis, this box contains two one-half field oxide features and one active area feature for a total of two features. The area of the mbit is therefore

$$4F \cdot 2F = 8F^2 \tag{2.2}$$

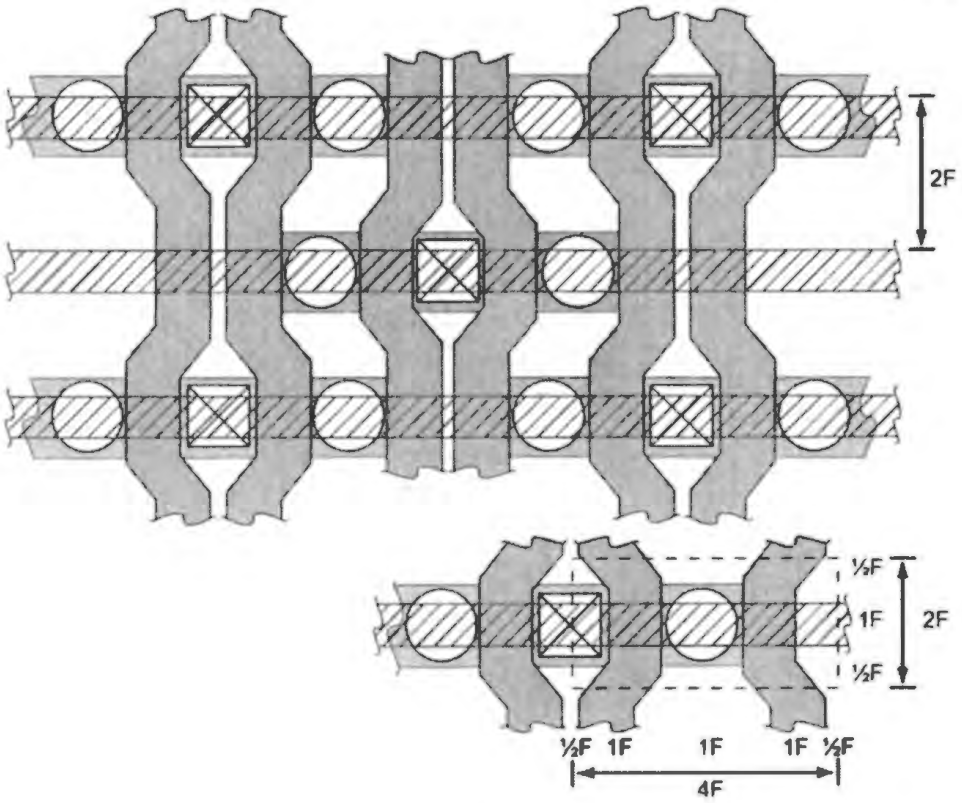


Figure 2.3 Layout to show $8F^2$ derivation.

The folded array architecture, as shown in Figure 2.2, always produces an $8F^2$ mbit. This results from the fact that each wordline connects or forms a crosspoint with an mbit transistor on every other digitline and must pass around mbit transistors as field poly on the remaining digitlines. The field poly in each mbit cell adds two square features to what would have been a $6F^2$ cell. Although the folded array yields a cell that is 25% larger than other array architectures, it also produces superior signal-to-noise performance, especially when combined with some form of digitline twisting [4]. Superior low-noise performance made folded array architecture the architecture of choice since the 64kbit generation [5].

A folded array is schematically depicted in Figure 2.4. Sense amplifier circuits placed at the edge of each array connect to both true and complement digitlines *(D* and *D*)* coming from a single array. Optional digitline pair twisting in one or more positions reduces and balances the coupling to adjacent digitline pairs and improves overall signal-to-noise characteristics [4]. Figure 2.5 shows the variety of twisting schemes used throughout the DRAM industry [6].

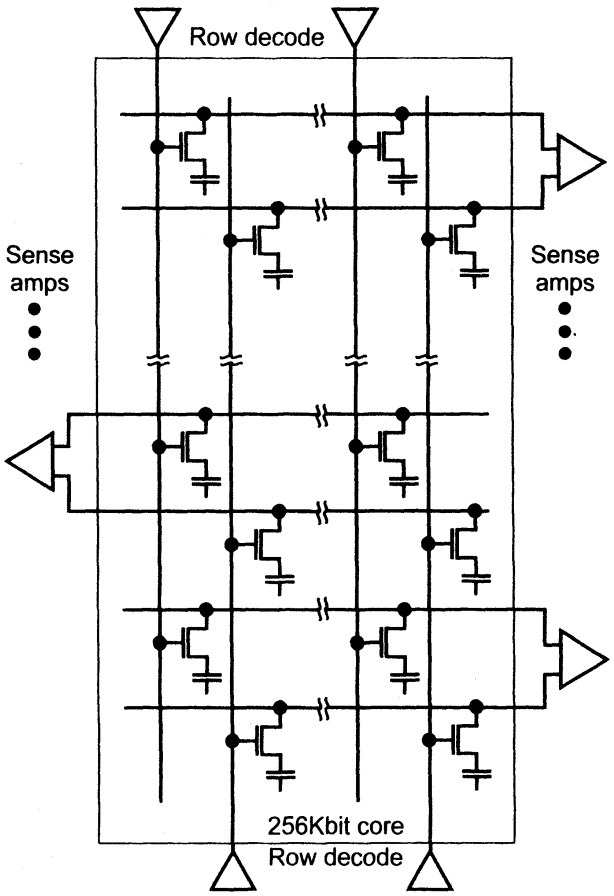


Figure 2.4 Folded digitline array schematic.

Ideally, a twisting scheme equalizes the coupling terms from each digitline to all other digitlines, both true and complement. If implemented properly, the noise terms cancel or at least produce only common-mode noise to which the differential sense amplifier is more immune.

Each digitline twist region consumes valuable silicon area. Thus, design engineers resort to the simplest and most efficient twisting scheme to get the job done. Because the coupling between adjacent metal lines is inversely proportional to the line spacing, the signal-to-noise problem gets increasingly worse as DRAMs scale to smaller and smaller dimensions. Hence, the industry trend is toward use of more complex twisting schemes on succeeding generations [6][7].

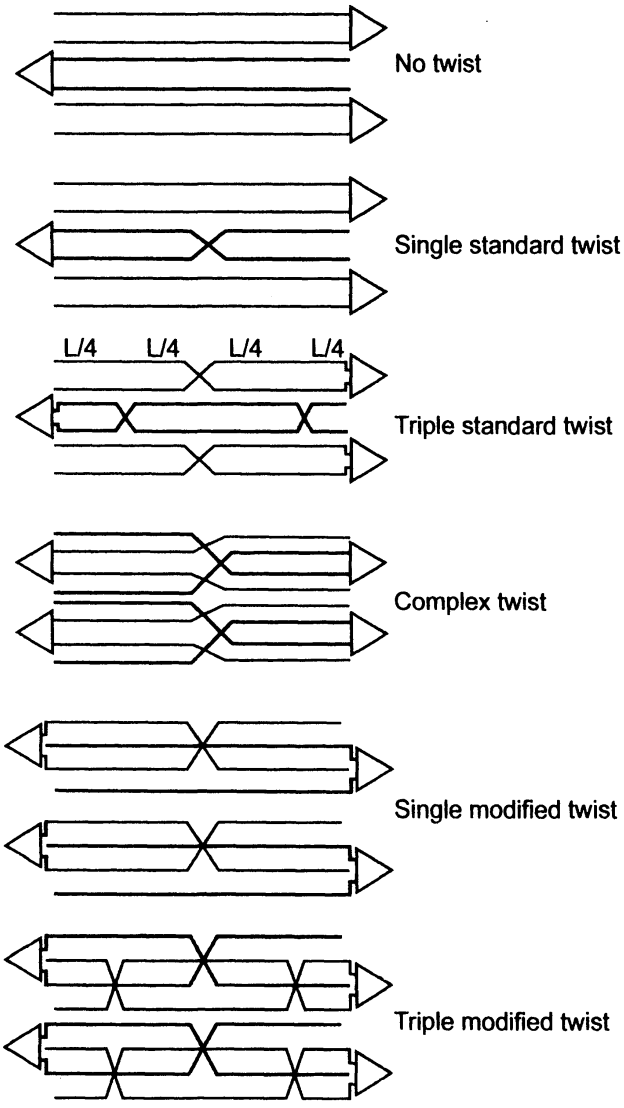


Figure 2.5 Digitline twisting scheme.

An alternative to the folded array architecture, popular prior to the 64kbit generation [1], was the open digitline architecture. Seen schematically in Figure 2.6, this architecture also features the sense amplifier circuits between two sets of arrays [8]. Unlike the folded array, however, true and complement digitlines *(D* and *D*)* connected to each sense amplifier pair come from separate arrays [9]. This arrangement precludes using digitline twisting to improve signal-to-noise performance, which is the prevalent reason why the industry switched to folded arrays. Note that unlike the folded array architecture, each wordline in an open digitline architecture connects to mbit transistors on every digitline, creating crosspoint-style arrays.

This feature permits a 25% reduction in mbit size to only $6F^2$ because the wordlines do not have to pass alternate mbits as field poly. The layout for an array of standard $6F^2$ mbit pairs is shown in Figure 2.7 [2]. A box is drawn around one of the mbits to show the $6F^2$ cell boundary. Again, two mbits share a common digitline contact to improve layout efficiency. While folded array architectures were used extensively in second-generation DRAMs [8] and the open array architectures, as just mentioned were used extensively in first-generation DRAMs, the third-generation designs are seeing a return to the open architecture. As the device/array size continue to shrink with scaling, the effects of coupled noise on the arrays is reduced, which makes the open array architecture both viable and preferable (due to the reduced size).

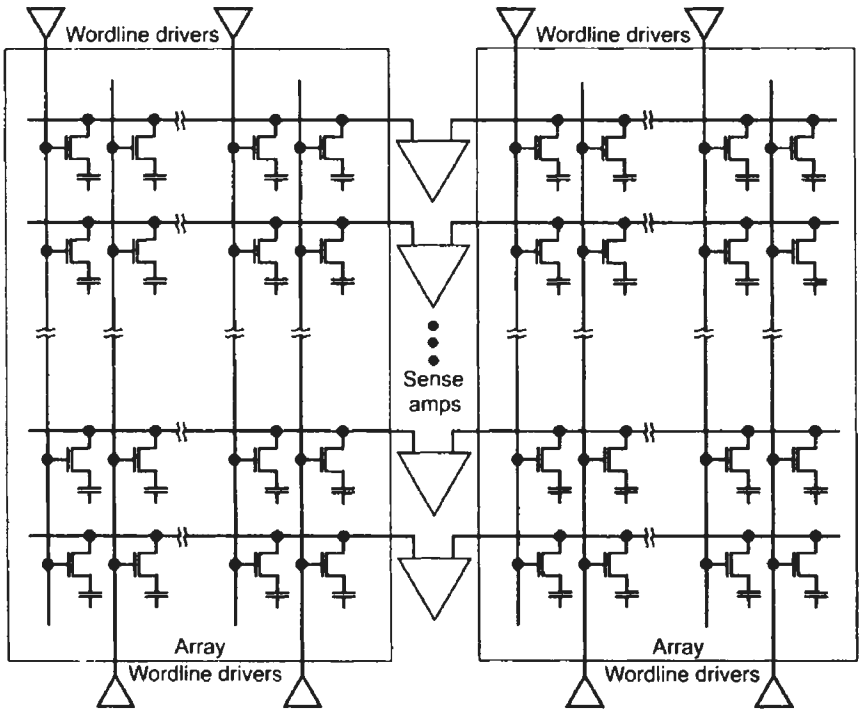


Figure 2.6 Open digitline array schematic.

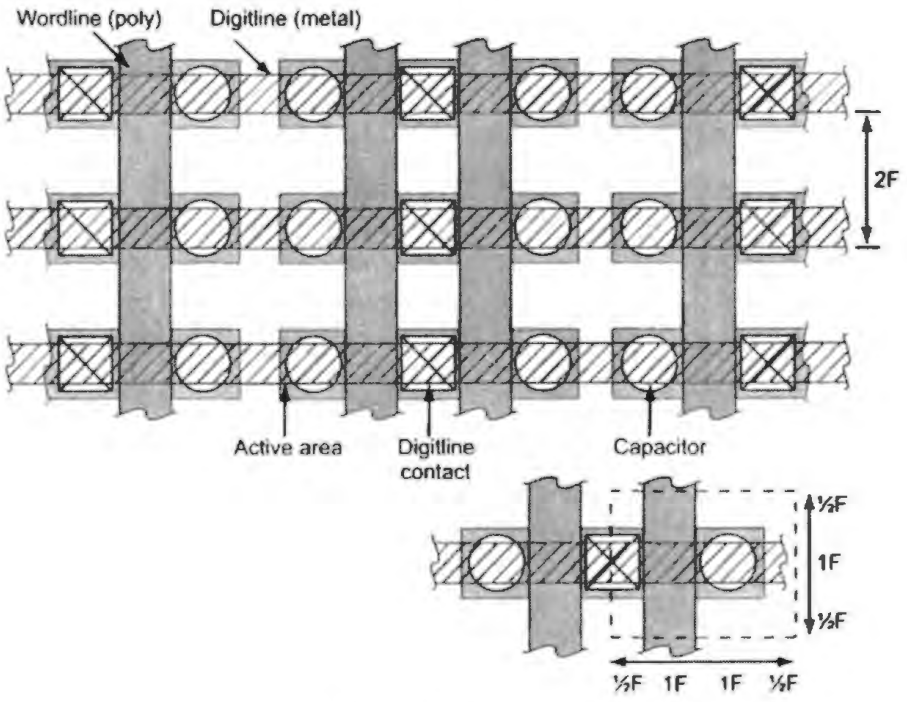


Figure 2.7 Open digitline array layout.
(Feature size, *F*, is equal to one-half digitline pitch.)

Digitline capacitive components, contributed by each mbit, include junction capacitance, digitline-to-cellplate (poly3), digitline-to-wordline, digitline-to-digitline, digitline-to-substrate, and, in some cases, digitline-to-storage cell (poly2) capacitance. Therefore, each mbit connected to the digitline adds a specific amount of capacitance to the digitline. Most modern DRAM designs have no more than 512 mbits connected to a digitline segment.

Two factors dictate this quantity. First, for a given cell size, as determined by row and column pitches, a maximum storage capacitance can be achieved without resorting to exotic processes or excessive cell height. For processes in which the digitline is above the storage capacitor (buried capacitor), contact technology determines the maximum allowable cell height. This fixes the volume available (cell area multiplied by cell height) in which to build the storage capacitor. Second, as the digitline capacitance increases, the power associated with charging and discharging this capacitance during Read and Write operations increases. Any given wordline essentially accesses (crosses) all of the columns within a DRAM. For a 256-Meg DRAM, each wordline crosses 16,384 columns. With a multiplier such as that, it is easy to appreciate why limits to digitline capacitance are necessary to keep power dissipation low.

Figure 2.8 presents a process cross section for the buried capacitor mbit depicted in Figure 2.2, and Figure 2.9 shows a SEM image of the buried capacitor mbit. This type of mbit, employing a buried capacitor structure, places the digitline physically above the storage capacitor [10]. The digitline is constructed from either metal or polycide, while the digitline contact is formed using a metal or polysilicon plug technology. The mbit capacitor is formed with polysilicon (poly2) as the bottom plate, an oxide-nitride-oxide (ONO) dielectric, and a sheet of polysilicon (poly3). This top sheet of polysilicon becomes a common node shared by all mbit capacitors. The capacitor shape can be simple, such as a rectangle, or complex, such as concentric cylinders or stacked discs. The most complex capacitor structures are the topic of many DRAM process papers [11][12][13].

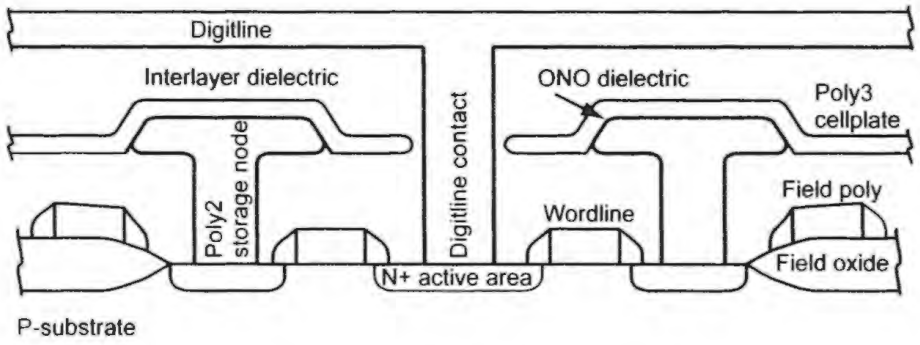


Figure 2.8 Buried capacitor cell process cross section.

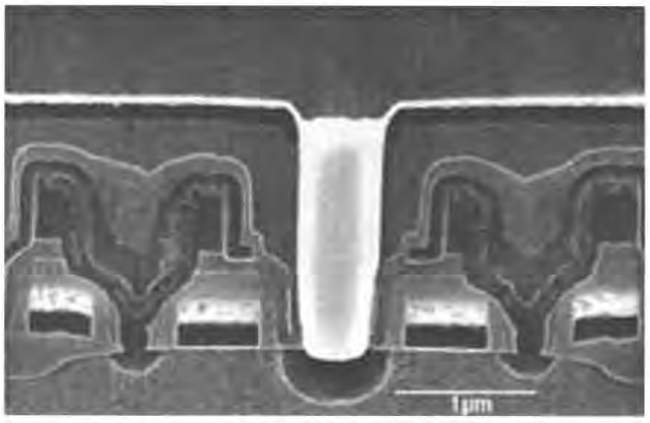


Figure 2.9 Buried capacitor cell process SEM image.

The ONO dielectric undergoes optimization to achieve maximum capacitance with minimum leakage. It must also tolerate the maximum DRAM operating voltage without breaking down. For this reason, the cellplate (poly3) is normally biased at $+V_{CC}/2$ V, ensuring that the dielectric

has no more than $V_{CC}/2$ V across it for either stored logic state, a logic one at $+V_{CC}/2$ V or a logic zero at $-V_{CC}/2$ V.

Two other basic mbit configurations are used in the DRAM industry. The first, shown in Figures 2.10, 2.11, and 2.12, is referred to as a *buried digitline* or *capacitor over bitline* (COB) cell [14][15]. The digitline in this cell is almost always formed of polysilicon rather than of metal.

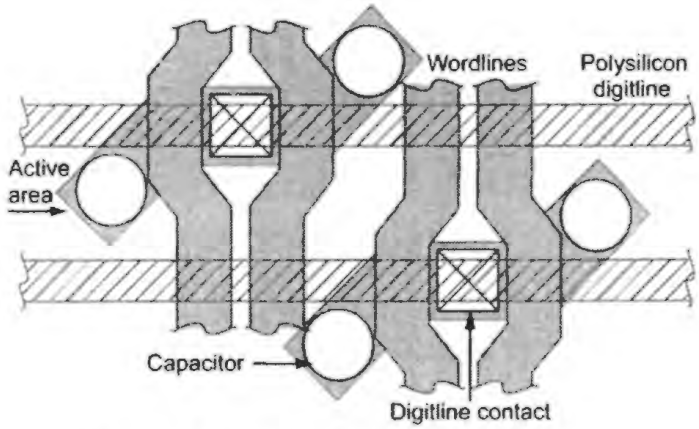


Figure 2.10 Buried digitline Mbit cell layout.

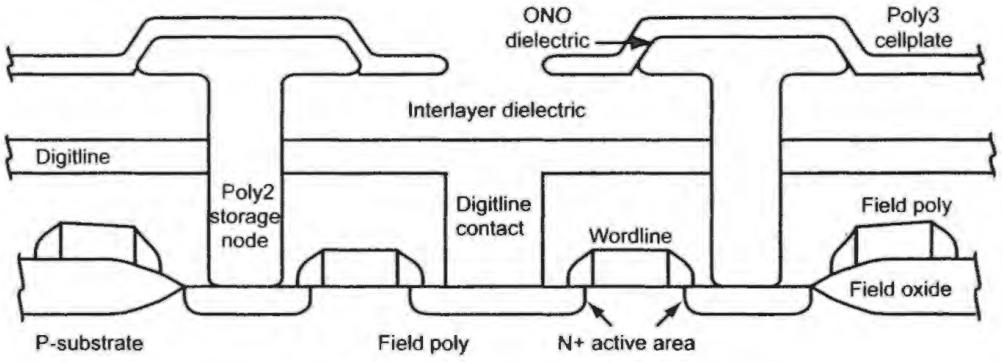


Figure 2.11 Buried digitline mbit process cross section.

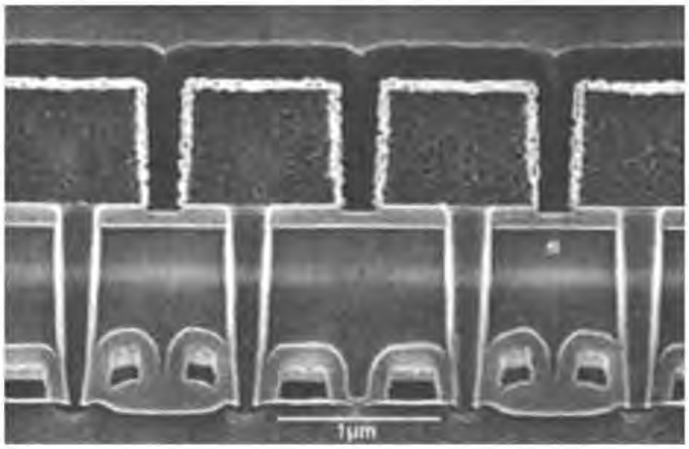


Figure 2.12 Buried digitline Mbit process SEM image.

As viewed from the top, the active area is normally bent or angled to accommodate the storage capacitor contact that must drop between digitlines. An advantage of the buried digitline cell over the buried capacitor cell of Figure 2.8 is that its digitline is physically very close to the silicon surface, making digitline contacts much easier to produce. The angled active area, however, reduces the effective active area pitch, constraining the isolation process even further. In buried digitline cells, it is also very difficult to form the capacitor contact. Because the digitline is at or near minimum pitch for the process, insertion of a contact between digitlines can be difficult.

Figures 2.13 and 2.14 present a process cross section of the third type of mbit used in the construction of DRAMs. Using trench storage capacitors, this cell is accordingly called a *trench cell* [12][13]. Trench capacitors are formed in the silicon substrate, rather than above the substrate, after etching deep holes into the wafer. The storage node is a doped polysilicon plug, which is deposited in the hole following growth or deposition of the capaci-

tor dielectric. Contact between the storage node plug and the transistor drain is usually made through a poly strap.

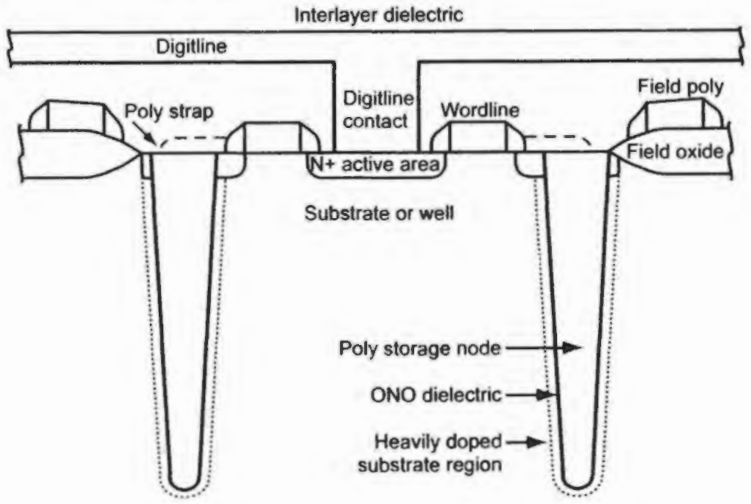


Figure 2.13 Trench capacitor Mbit process cross section.



Figure 2.14 Trench capacitor Mbit process SEM image.

With most trench capacitor designs, the substrate serves as the common-node connection to the capacitors, preventing the use of $+V_{CC}/2$ bias and thinner dielectrics. The substrate is heavily doped around the capacitor to reduce resistance and improve the capacitor's CV characteristics. A real

advantage of the trench cell is that the capacitance can be increased by merely etching a deeper hole into the substrate [16].

Furthermore, the capacitor does not add stack height to the design, greatly simplifying contact technology. The disadvantage of trench capacitor technology is the difficulty associated with reliably building capacitors in deep silicon holes and connecting the trench capacitor to the transistor drain terminal.

2.2 THE SENSE AMP

The term *sense amplifier* actually refers to a collection of circuit elements that pitch up to the digitlines of a DRAM array. This collection most generally includes isolation transistors, devices for digitline equilibration and bias, one or more Nsense amplifiers, one or more Psense amplifiers, and devices connecting selected digitlines to *I/O* signal lines. All of the circuits along with the wordline driver circuits, as discussed in Section 2.3, are called *pitch cells*. This designation comes from the requirement that the physical layout for these circuits is constrained by the digitline and wordline pitches of an array of mbits. For example, the sense amplifiers for a specific digitline pair (column) are generally laid out within the space of four digitlines. With one sense amplifier for every four digitlines, this is commonly referred to as quarter pitch or four pitch.

2.2.1 Equilibration and Bias Circuits

The first elements analyzed are the equilibration and bias circuits. As we can recall from the discussions of DRAM operation in Section 1.1, the digitlines start at $V_{CC}/2$ V prior to cell Access and Sensing [17]. It is vitally important to the Sensing operation that both digitlines, which form a column pair, are at the same voltage before the wordline is driven HIGH. Any offset voltage appearing between the pair directly reduces the effective signal produced during the Access operation [5]. Equilibration of the digitlines is accomplished with one or more NMOS transistors connected between the digitline conductors. NMOS are used because of their higher drive capability and the resulting faster equilibration.

An equilibration transistor, together with bias transistors, is shown in Figure 2.15. The gate terminal is connected to a signal called *equilibrate (EQ)*. *EQ* is held to V_{CC} whenever the external row address strobe signal *RAS** is HIGH. This indicates an inactive or *PRECHARGE* state for the DRAM. After *RAS** has fallen, *EQ* transitions LOW, turning the equilibration transistor OFF just prior to a wordline going HIGH. *EQ* will again transition HIGH at the end of a *RAS** cycle to force equilibration of the

digitlines. The equilibration transistor is sized large enough to ensure rapid equilibration of the digitlines to prepare the part for a subsequent access.

As shown in Figure 2.15, two more NMOS transistors accompany the *EQ* transistor to provide a bias level of $V_{CC}/2$ V. These devices operate in conjunction with equilibration to ensure that the digitline pair remains at the prescribed voltage for Sensing. Normally, digitlines that are at V_{CC} and ground equilibrate to $V_{CC}/2$ V [5]. The bias devices ensure that this occurs and also that the digitlines remain at $V_{CC}/2$, despite leakage paths that would otherwise discharge them. Again, for the same reasons as for the equilibration transistor, NMOS transistors are used. Most often, the bias and equilibration transistors are integrated to reduce their overall size. $V_{CC}/2$ V *PRECHARGE* is used on most modern DRAMs because it reduces power consumption and Read-Write times and improves Sensing operations. Power consumption is reduced because a $V_{CC}/2$ *PRECHARGE* voltage can be obtained by equilibrating the digitlines (which are at V_{CC} and ground, respectively) at the end of each cycle.

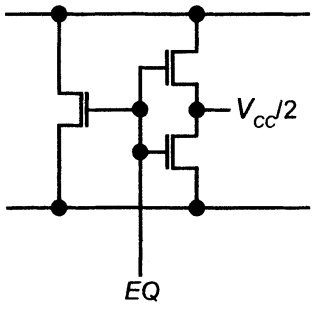


Figure 2.15 Equilibration schematic.

The charge-sharing between the digitlines produces $V_{CC}/2$ without additional I_{CC} current. Some designs, however, equilibrate the digitlines to V_{DD} [18]. Because the wordlines and digitlines are both at V_{CC} when the part is inactive, row-to-column shorts that exist do not contribute to increased standby current. $V_{CC}/2$ *PRECHARGE* DRAMs, on the other hand, suffer higher standby current with row-to-column shorts because the wordlines and digitlines are at different potentials when the part is inactive. A layout for the equilibration and bias circuits is shown in Figure 2.16.

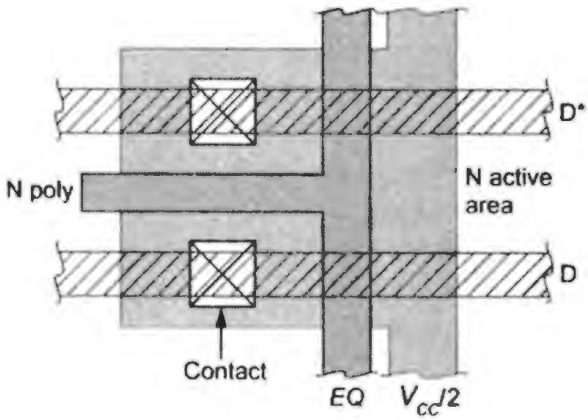


Figure 2.16 Equilibration and bias circuit layout.

2.2.2 Isolation Devices

Isolation devices are important to the sense amplifier circuits. These devices are NMOS transistors placed between the array digitlines and the sense amplifiers (see Figure 2.19). Isolation transistors are physically located on both ends of the sense amplifier layout. In quarter-pitch sense amplifier designs, there is one isolation transistor for every two digitlines. Although this is twice the active area width and space of an array, it nevertheless sets the limit for isolation processing in the pitch cells.

The isolation devices provide two functions. First, if the sense amps are positioned between and connected to two arrays, they electrically isolate one of the two arrays. This is necessary whenever a wordline fires in one array because isolation of the second array reduces the digitline capacitance driven by the sense amplifiers, thus speeding Read-Write times, reducing power consumption, and extending Refresh for the isolated array. Second, the isolation devices provide resistance between the sense amplifier and the digitlines. This resistance stabilizes the sense amplifiers and speeds up the Sensing operation by somewhat isolating the highly capacitive digitlines from the low-capacitance sense nodes [19]. Capacitance of the sense nodes between isolation transistors is generally less than 15fF, permitting the sense amplifier to latch much faster than if it were solidly connected to the digitlines. The isolation transistors slow Write-Back to the mbits, but this is far less of a problem than initial Sensing.

2.2.3 Input/Output Transistors

The input/output (*I/O*) transistors allow data to be read from and written to specific digitline pairs. A single *I/O* transistor is connected to each sense node as shown in Figure 2.17. The outputs of each *I/O* transistor are connected to *I/O* signal pairs. Commonly, there are two pairs of *I/O* signal lines,

which permit four *I/O* transistors to share a single *column select (CSEL)* control signal. DRAM designs employing two or more metal layers run the column select lines across the arrays in either Metal2 or Metal3. Each column select can activate four *I/O* transistors on each side of an array to connect four digitline pairs (columns) to peripheral data path circuits. The *I/O* transistors must be sized carefully to ensure that instability is not introduced into the sense amplifiers by the *I/O* bias voltage or remnant voltages on the *I/O* lines.

Although designs vary significantly as to the numerical ratio, *I/O* transistors are designed to be two to eight times smaller than the Nsense amplifier transistors. This is sometimes referred to as *beta ratio*. A beta ratio between five and eight is considered standard; however, it can only be verified with silicon. Simulations may fail to adequately predict sense amplifier instability, although theory would predict better stability with higher beta ratio and better Write times with lower beta ratio. During Write, the sense amplifier remains ON and must be overdriven by the Write driver (see Section 1.2.2).

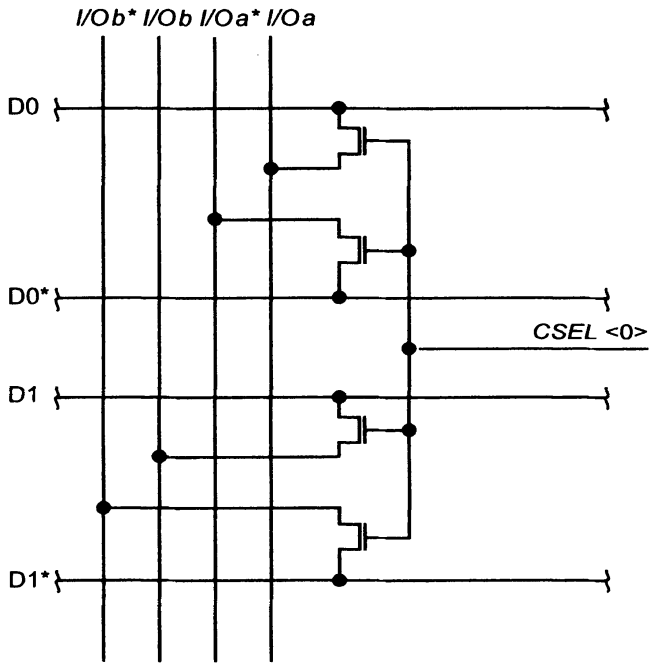


Figure 2.17 *I/O* transistors.

2.2.4 Nsense and Psense Amplifiers

The fundamental elements of any sense amplifier block are the Nsense amplifier and the Psense amplifier. These amplifiers, as previously discussed, work together to detect the access signal voltage and drive the digitlines, accordingly to V_{CC} and ground. The Nsense amplifier depicted in Figure 2.18 consists of cross-coupled NMOS transistors. The Nsense ampli-

fier drives the low-potential digitline to ground. Similarly, the Psense amplifier consists of cross-coupled PMOS transistors and drives the HIGH-potential digitline to V_{CC}.

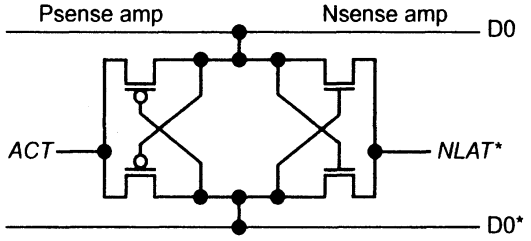


Figure 2.18 Basic sense amplifier block.

The sense amplifiers are carefully designed to guarantee correct detection and amplification of the small signal voltage produced during cell access (less than 200mV) [5]. Matching of transistor V_{TH}, transconductance, and junction capacitance within close tolerances helps ensure reliable sense amplifier operation. Ultimately, the layout dictates the overall balance and performance of the sense amplifier block. As a result, a tremendous amount of time is spent ensuring that the sense amplifier layout is optimum. Symmetry and exact duplication of elements are critical to a successful design. This includes balanced coupling to all sources of noise, such as *I/O* lines and latch signals *(NLAT** and *ACT).* Balance is especially critical for layout residing inside the isolation transistors. And because the sense node capacitance is very low, it is more sensitive to noise and circuit imbalances.

While the majority of DRAM designs latch the digitlines to V_{CC} and ground, a certain number of designs choose to reduce these levels. Various technical papers report improved Refresh times and lower power dissipation through reductions in latch voltages [20][21]. At first, this appears contradictory: writing a smaller charge into the memory cell would require refreshing the cell more often. However, by maintaining lower *drain-to-source voltages* (V_{DS}) and negative *gate-to-source voltages* (V_{GS}) across non-accessed mbit transistors, substantially lower subthreshold leakage and longer Refresh times can be realized, despite the smaller stored charge. An important concern that we are not mentioning here is the leakage resulting from defects in the silicon crystal structure. These defects can result in an increase in the drain/substrate diode saturation current and can practically limit the Refresh times.

Most designs that implement reduced latch voltages generally raise the Nsense amplifier latch voltage without lowering the Psense amplifier latch voltage. Designated as boosted sense ground designs, they write data into each mbit using full V_{CC} for a logic one and boosted ground for a logic zero. The sense ground level is generally a few hundred millivolts above true ground. In standard DRAMs, which drive digitlines fully to ground, the V_{GS}

of nonaccessed mbits becomes zero when the digitlines are latched. This results in high subthreshold leakage for a stored one level because full V_{CC} exists across the mbit transistor while the V_{GS} is held to zero. Stored zero levels do not suffer from prolonged subthreshold leakage: any amount of cell leakage produces a negative V_{GS} for the transistor. The net effect is that a stored one level leaks away much faster than a stored zero level. One's level retention, therefore, establishes the maximum Refresh period for most DRAM designs. Boosted sense ground extends Refresh by reducing subthreshold leakage for stored ones. This is accomplished by guaranteeing negative gate-to-source bias on nonaccessed mbit transistors. The benefit of extended Refresh from these designs is somewhat diminished, though, by the added complexity of generating boosted ground levels and the problem of digitlines that no longer equilibrate at $V_{CC}/2$ V.

2.2.5 Rate of Activation

The rate at which the sense amplifiers are activated has been the subject of some debate. A variety of designs use multistage circuits to control the rate at which *NLAT** fires. Especially prevalent with boosted sense ground designs are two-stage circuits that initially drive *NLAT** quickly toward true ground to speed sensing and then bring *NLAT** to the boosted ground level to reduce cell leakage. An alternative to this approach, again using two-stage drivers, drives *NLAT** to ground, slowly at first to limit current and digitline disturbances. This is followed by a second phase in which *NLAT** is driven more strongly toward ground to complete the Sensing operation. This phase usually occurs in conjunction with *ACT* activation. Although these two designs have contrary operation, each meets specific performance objectives: trading off noise and speed.

2.2.6 Configurations

Figure 2.19 shows a sense amplifier block commonly used in double- or triple-metal designs. It features two Psense amplifiers outside the isolation transistors, a pair of *EQ*/bias *(EQ*b*)* devices, a single Nsense amplifier, and a single *I/O* transistor for each digitline. Because only half of the sense amplifiers for each array are on one side, this design is quarter pitch, as are the designs in Figures 2.20 and 2.21. Placement of the Psense amplifiers outside the isolation devices is necessary because a full one level *(V_{CC})* cannot pass through unless the gate terminal of the *ISO* transistors is driven above V_{CC}. *EQ*/bias transistors are placed outside of the *ISO* devices to permit continued equilibration of digitlines in arrays that are isolated. The *I/O* transistor gate terminals are connected to a common *CSEL* signal for four adjacent digitlines. Each of the four *I/O* transistors is tied to a separate *I/O*

bus. This sense amplifier, though simple to implement, is somewhat larger than other designs due to the presence of two Psense amplifiers.

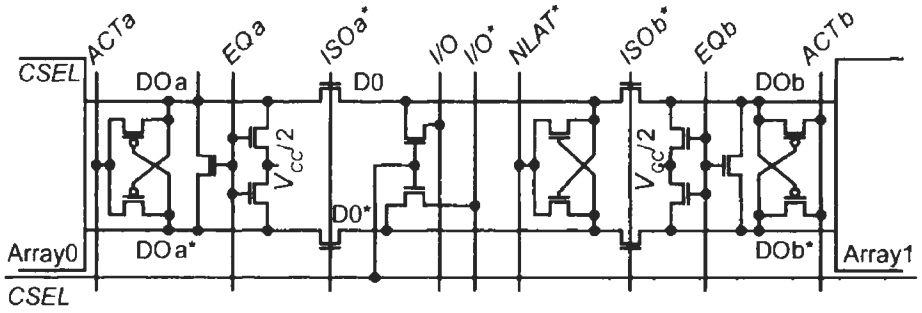


Figure 2.19 Standard sense amplifier block.

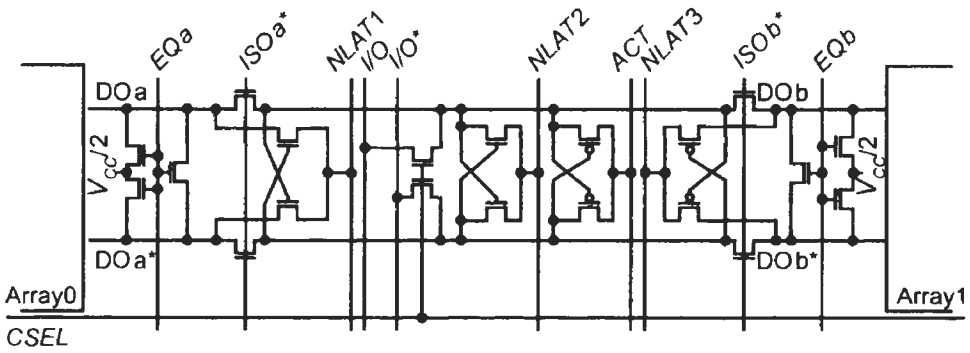


Figure 2.20 Complex sense amplifier block.

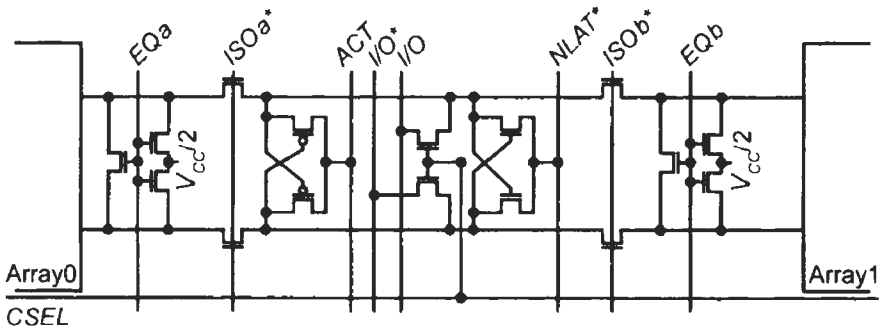


Figure 2.21 Reduced sense amplifier block.

Figure 2.20 shows a second, more complicated style of sense amplifier block. This design employs a single Psense amplifier and three sets of Nsense amplifiers. Because the Psense amplifier is within the isolation transistors, the isolation devices must be the NMOS depletion type, the PMOS enhancement type, or the NMOS enhancement type with boosted gate drive

to permit writing a full logic one into the array mbits. The triple Nsense amplifier is suggestive of PMOS isolation transistors; it prevents full zero levels to be written unless the Nsense amplifiers are placed adjacent to the arrays. In this more complicated style of sense amplifier block, using three Nsense amplifiers guarantees faster sensing and higher stability than a similar design using only two Nsense amplifiers. The inside Nsense amplifier is fired before the outside Nsense amplifiers. However, this design will not yield a minimum layout, an objective that must be traded off against performance needs.

The sense amplifier block of Figure 2.21 can be considered a reduced configuration. This design has only one Nsense-amp and one Psense-amp, both of which are placed within the isolation transistors. To write full logic levels, either the isolation transistors must be depletion mode devices or the gate voltage must be boosted above V_{CC} by at least one V_{TH}. This design still uses a pair of *EQ*/bias circuits to maintain equilibration on isolated arrays.

Only a handful of designs operates with a single *EQ*/bias circuit inside the isolation devices, as shown in Figure 2.22. Historically, DRAM engineers tended to shy away from designs that permitted digitlines to float for extended periods of time. However, as of this writing, at least three manufacturers in volume production have designs using this scheme.

A sense amplifier design for single-metal DRAMs is shown in Figure 2.23. Prevalent on 1-Meg and 4-Meg designs, single-metal processes conceded to multi-metal processes at the 16-Meg generation. Unlike the sense amplifiers shown in Figures 2.19, 2.20, 2.21, and 2.22, single-metal sense amps are laid out at half pitch: one amplifier for every two array digitlines. This type of layout is extremely difficult and places tight constraints on process design margins. With the loss of Metal2, the column select signals are not brought across the memory arrays. Generating column select signals locally for each set of *I/O* transistors requires a full column decode block.

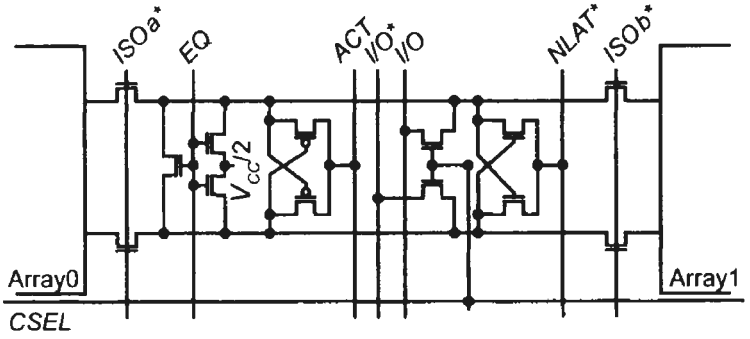


Figure 2.22 Minimum sense amplifier block.

As shown in Figure 2.23, the Nsense-amp and Psense-amps are placed on separate ends of the arrays. Sharing sense amps between arrays is especially beneficial for single-metal designs. As illustrated in Figure 2.23, two Psense-amps share a single Nsense-amp. In this case, with the *I/O* devices on only one end, the right Psense-amp is activated only when the right array is accessed. Conversely, the left Psense-amp is always activated, regardless of the accessed array, because all Read and Write data must pass through the left Psense-amp to get to the *I/O* devices.

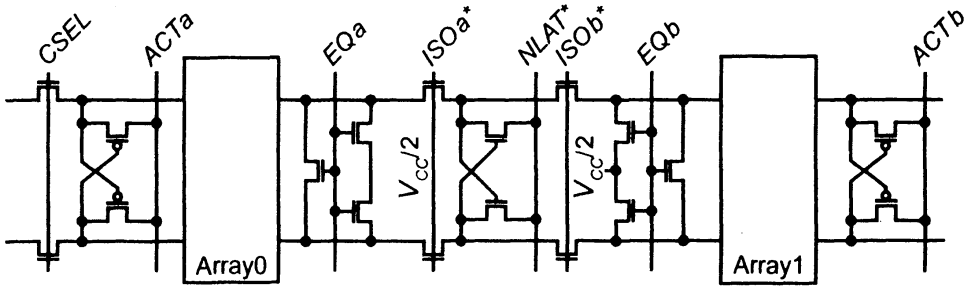


Figure 2.23 Single-metal sense amplifier block.

2.2.7 Operation

A set of signal waveforms is illustrated in Figure 2.24 for the sense amplifier of Figure 2.19. These waveforms depict a Read-Modify-Write cycle (Late Write) in which the cell data is first read out and then new data is written back. In this example, a one level is read out of the cell, as indicated by *D*0 rising above D0 during cell access. A one level is always $+V_{CC}/2$ in the mbit cell, regardless of whether it is connected to a true or complement digitline. The correlation between mbit cell data and the data appearing at the DRAM's data terminal *(DQ)* is a function of the data topology and the presence of data scrambling. Data or topo scrambling is implemented at the circuit level: it ensures that the mbit data state and the *DQ* logic level are in agreement. An mbit one level ($+V_{CC}/2$) corresponds to a logic one at the *DQ*, and an mbit zero level ($-V_{CC}/2$) corresponds to a logic zero at the *DQ* terminal.

Writing specific data patterns into the memory arrays is important to DRAM testing. Each type of data pattern identifies the weaknesses or sensitivities of each cell to the data in surrounding cells. These patterns include solids, row stripes, column stripes, diagonals, checkerboards, and a variety of moving patterns. Test equipment must be programmed with the data topology of each type of DRAM to correctly write each pattern. Often the tester itself guarantees that the pattern is correctly written into the arrays, unscrambling the complicated data and address topology as necessary to write a specific pattern. On some newer DRAM designs, part of this task is

implemented on the DRAM itself, in the form of a topo scrambler, such that the mbit data state matches the *DQ* logic level. This implementation somewhat simplifies tester programming.

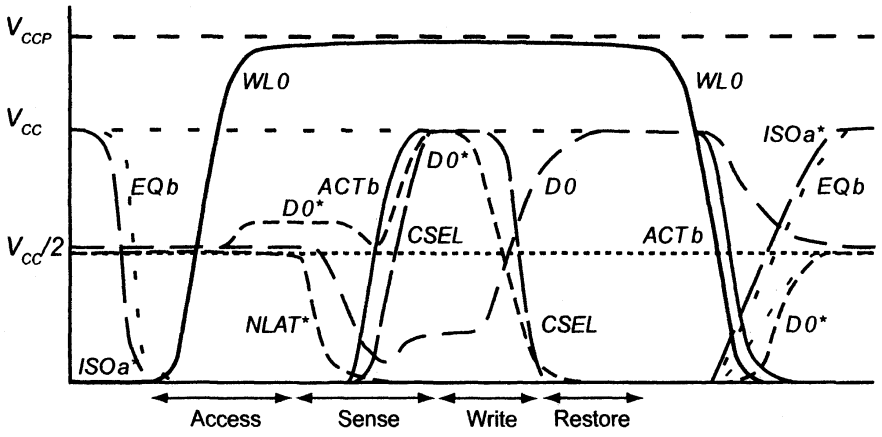


Figure 2.24 Waveforms for the Read-Modify-Write cycle.

Returning to Figure 2.24, we see that a wordline is fired in Array1. Prior to this, *ISO*a* will go LOW to isolate Array0, and *EQ*b will go LOW to disable the *EQ*/bias transistors connected to Array1. The wordline then fires HIGH, accessing an mbit, which dumps charge onto *D*0*. *NLAT**, which is initially at $V_{CC}/2$, drives LOW to begin the Sensing operation and pulls *D*0 toward ground. Then, *ACT* fires, moving from ground to V_{CC}, activating the Psense amplifier and driving *D*0* toward V_{CC}. After separation has commenced, *CSEL*0 rises to V_{CC}, turning ON the *I/O* transistors so that the cell data can be read by peripheral circuits. The *I/O* lines are biased at a voltage close to V_{CC}, which causes *D*1 to rise while the column is active. After the Read is complete, Write drivers in the periphery turn ON and drive the *I/O* lines to opposite data states (in our example).

This new data propagates through the *I/O* devices and writes over the existing data stored in the sense amplifiers. Once the sense amplifiers latch the new data, the Write drivers and the *I/O* devices can be shut down, allowing the sense amplifiers to finish restoring the digitlines to full levels. Following this restoration, the wordline transitions LOW to shut OFF the mbit transistor. Finally, *EQ*b and *ISO*a* fire HIGH to equilibrate the digitlines back to $V_{CC}/2$ and reconnect Array0 to the sense amplifiers. The timing for each event of Figure 2.24 depends on circuit design, transistor sizes, layout, device performance, parasitics, and temperature. While timing for each event must be minimized to achieve optimum DRAM performance, it cannot be pushed so far as to eliminate all timing margins. Margins are necessary to ensure proper device operation over the expected range of process variations and the wide range of operating conditions.

Again, there is not one set of timing waveforms that covers all design options. The sense amps of Figures 2.19–2.23 all require slightly different signals and timings. Various designs actually fire the Psense amplifier prior to or coincident with the Nsense amplifier. This obviously places greater constraints on the Psense amplifier design and layout, but these constraints are balanced by potential performance benefits. Similarly, the sequence of events as well as the voltages for each signal can vary. There are almost as many designs for sense amplifier blocks as there are DRAM design engineers. Each design reflects various influences, preconceptions, technologies, and levels of understanding. The bottom line is to maximize yield and performance and minimize everything else.

2.3 ROW DECODER ELEMENTS

Row decode circuits are similar to sense amplifier circuits in that they pitch up to mbit arrays and have a variety of implementations. A row decode block is comprised of two basic elements: a wordline driver and an address decoder tree. There are three basic configurations for wordline driver circuits: the NOR driver, the inverter (CMOS) driver, and the bootstrap driver. In addition, the drivers and associated decode trees can be configured either as local row decodes for each array section or as global row decodes that drive a multitude of array sections.

Global row decodes connect to multiple arrays through metal wordline straps. The straps are stitched to the polysilicon wordlines at specific intervals dictated by the polysilicon resistance and the desired *RC* wordline time constant. Some processes that strap wordlines with metal do not silicide the polysilicon, although doing so would reduce the number of stitch regions required. Strapping wordlines and using global row decoders obviously reduces die size [22], very dramatically in some cases. The disadvantage of strapping is that it requires an additional metal layer at minimum array pitch. This puts a tremendous burden on process technologists in which three conductors are at minimum pitch: wordlines, digitlines, and wordline straps.

Local row decoders, on the other hand, require additional die size rather than metal straps. It is highly advantageous to reduce the polysilicon resistance in order to stretch the wordline length and lower the number of row decodes needed. This is commonly achieved with silicided polysilicon processes. On large DRAMs, such as the 1Gb, the area penalty can be prohibitive, making low-resistance wordlines all the more necessary.

2.3.1 Bootstrap Wordline Driver

The bootstrap wordline driver shown in Figure 2.25 is built exclusively from NMOS transistors [22], resulting in the smallest layout of the three types of driver circuits. The absence of PMOS transistors eliminates large Nwell regions from the layout. As the name denotes, this driver relies on bootstrapping principles to bias the output transistor's gate terminal. This bias voltage must be high enough to allow the NMOS transistor to drive the wordline to the boosted wordline voltage V_{CCP}.

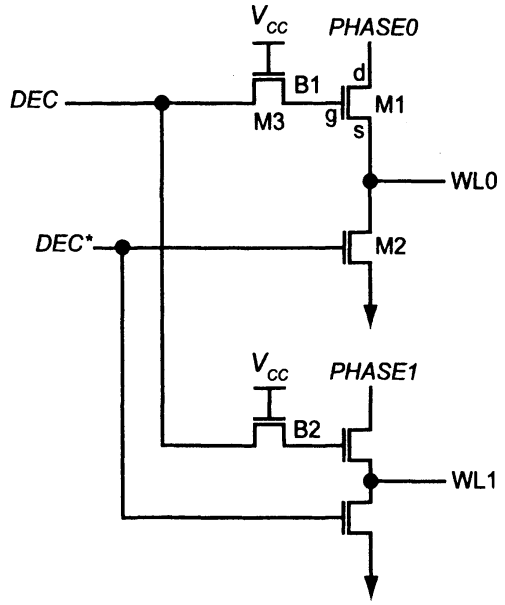


Figure 2.25 Bootstrap wordline driver.

Operation of the bootstrap driver is described with help from Figure 2.26. Initially, the driver is OFF with the wordline and *PHASE* terminals at ground. The wordline is held at ground by transistor M2 because the decoder output signal *DEC** is at V_{CC}. The gate of M3, the pass transistor, is fixed at V_{CC}. The signals *DEC* and *DEC** are fed from a decode circuit that will be discussed later. As a complement pair, *DEC* and *DEC** represent the first of two terms necessary to decode the correct wordline. *PHASE*0, which is also fed from a decode circuit, represents the second term. If *DEC* rises to V_{CC} and *DEC** drops to ground, as determined by the decoder, the boot node labeled *B*1 will rise to $V_{CC} - V_{TN}$ V, and M2 will turn OFF. The wordline continues to be held to ground by M1 because *PHASE*0 is still grounded. After *B*1 rises to $V_{CC} - V_{TN}$, the *PHASE*0 signal fires to the boosted wordline voltage V_{CCP}. Because of the gate-to-drain and gate-to-source capacitance of M1, the gate of M1 boots to an elevated voltage, V_{boot}. This voltage is determined by the parasitic capacitance of node *B*1, C_{GS1}, C_{GD1}, V_{CCP}, and the initial voltage at *B*1, $V_{CC} - V_{TN}$. Accordingly,

$$V_{boot} \cong \frac{(V_{CCP} \cdot C_{GD1})}{(C_{GS1} + C_{GD1} + C_{B1})} + (V_{CC} - V_{TN}) \tag{2.3}$$

In conjunction with the wordline voltage rising from ground to V_{CCP}, the gate-to-source capacitance of M1 provides a secondary boost to the boot node. The secondary boost helps to ensure that the boot voltage is adequate to drive the wordline to a full V_{CCP} level.

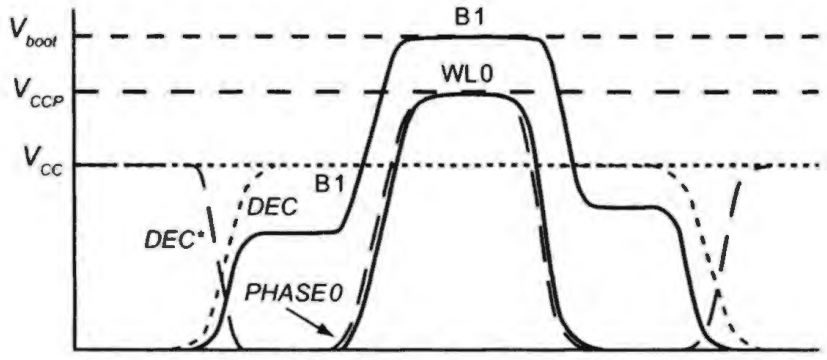


Figure 2.26 Bootstrap operation waveforms.

The layout of the boot node is very important to the bootstrap wordline driver. First, the parasitic capacitance of node $B1$, which includes routing, junction, and overlap components, must be minimized to achieve maximum boot efficiency. Second, charge leakage from the boot node must be minimized to ensure adequate V_{GS} for transistor M1 such that the wordline remains at V_{CCP} for the maximum *RAS** LOW period. Low leakage is often achieved by minimizing the source area for M3 or by using donut gate structures that surround the source area, as illustrated in Figure 2.27.

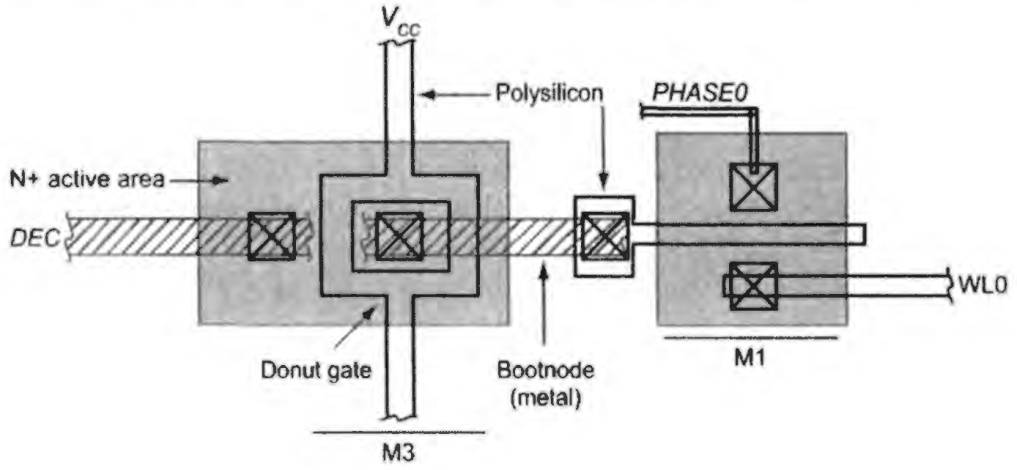


Figure 2.27 Donut gate structure layout.

The bootstrap driver is turned OFF by first driving the *PHASE*0 signal to ground. M1 remains ON because node *B*1 cannot drop below $V_{CC}-V_{TH}$; M1 substantially discharges the wordline toward ground. This is followed by the address decoder turning OFF, bringing *DEC* to ground and *DEC** to V_{CC}. With *DEC** at V_{CC}, transistor M2 turns ON and fully clamps the wordline to ground. A voltage level translator is required for the *PHASE* signal because it operates between ground and the boosted voltage V_{CCP}. For a global row decode configuration, this requirement is not much of a burden. For a local row decode configuration, however, the requirement for level translators can be very troublesome. Generally, these translators are placed either in the array gap cells at the intersection of the sense amplifier blocks and row decode blocks or distributed throughout the row decode block itself. The translators require both PMOS and NMOS transistors and must be capable of driving large capacitive loads. Layout of the translators is exceedingly difficult, especially because the overall layout needs to be as small as possible.

2.3.2 NOR Driver

The second type of wordline driver is similar to the bootstrap driver in that two decode terms drive the output transistor from separate terminals. The NOR driver, as shown in Figure 2.28, uses a PMOS transistor for M1 and does not rely on bootstrapping to derive the gate bias. Rather, the gate is driven by a voltage translator that converts *DEC* from V_{CC} to V_{CCP} voltage levels. This conversion is necessary to ensure that M1 remains OFF for unselected wordlines as the *PHASE* signal, which is shared by multiple drivers, is driven to V_{CCP}.

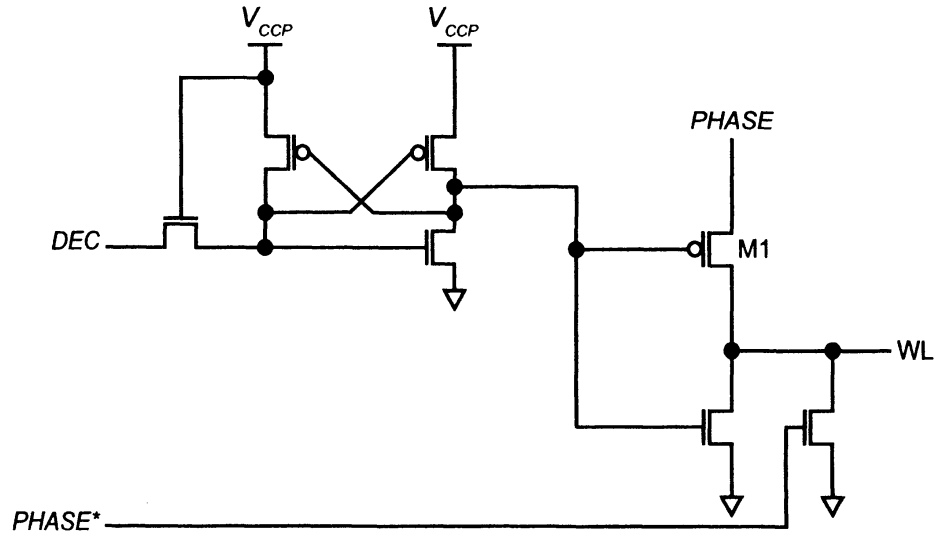


Figure 2.28 NOR driver.

To fire a specific wordline, *DEC* must be HIGH and the appropriate *PHASE* must fire HIGH. Generally, there are four to eight *PHASE* signals per row decoder block. The NOR driver requires one level translator for each *PHASE* and *DEC* signal. By comparison, the bootstrap driver only requires level translators for the *PHASE* signal.

2.3.3 CMOS Driver

The final wordline driver configuration is shown in Figure 2.29. In general, it lacks a specific name: it is sometimes referred to as a CMOS inverter driver or a CMOS driver. Unlike the first two drivers, the output transistor M1 has its source terminal permanently connected to V_{CCP}. This driver, therefore, features a voltage translator for each and every wordline. Both decode terms *DEC* and *PHASE** combine to drive the output stage through the translator. The advantage of this driver, other than simple operation, is low power consumption. Power is conserved because the translators drive only the small capacitance associated with a single driver. The *PHASE* translators of both the bootstrap and NOR drivers must charge considerable junction capacitance. The disadvantages of the CMOS driver are layout complexity and standby leakage current. Standby leakage current is a product of V_{CCP} voltage applied to M1 and its junction and subthreshold leakage currents. For a large DRAM with high numbers of wordline drivers, this leakage current can easily exceed the entire standby current budget unless great care is exercised in designing output transistor M1.

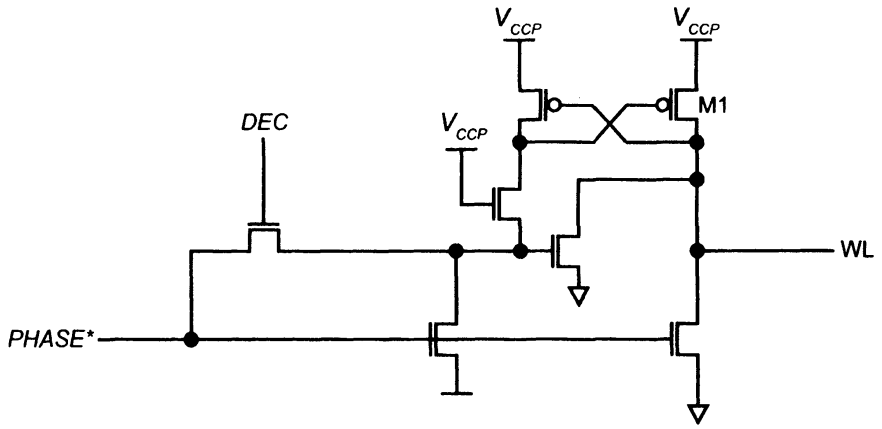


Figure 2.29 CMOS driver.

2.3.4 Address Decode Tree

With the wordline driver circuits behind us, we can turn our attention to the address decoder tree. There is no big secret to address decoding in the row decoder network. Just about any type of logic suffices: static, dynamic,

pass gate, or a combination thereof. With any type of logic, however, the primary objectives in decoder design are to maximize speed and minimize die area. Because a great variety of methods have been used to implement row address decoder trees, it is next to impossible to cover them all. Instead, we will give an insight into the possibilities by discussing a few of them.

Regardless of the type of logic with which a row decoder is implemented, the layout must completely reside beneath the row address signal lines to constitute an efficient, minimized design. In other words, the metal address tracks dictate the die area available for the decoder. Any additional tracks necessary to complete the design constitute wasted silicon. For DRAM designs requiring global row decode schemes, the penalty for inefficient design may be insignificant; however, for distributed local row decode schemes, the die area penalty may be significant. As with mbits and sense amplifiers, time spent optimizing row decode circuits is time well spent.

2.3.5 Static Tree

The most obvious form of address decode tree uses static CMOS logic. As shown in Figure 2.30, a simple tree can be designed using two-input NAND gates. While easy to design schematically, static logic address trees are not popular. They waste silicon and are very difficult to lay out efficiently. Static logic requires two transistors for each address term, one NMOS and one PMOS, which can be significant for many address terms. Furthermore, static gates must be cascaded to accumulate address terms, adding gate delays at each level. For these and other reasons, static logic gates are not used in row decode address trees in today's state-of-the-art DRAMs.

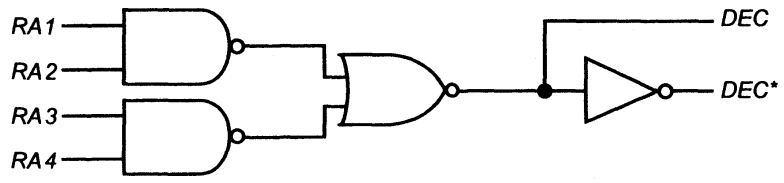


Figure 2.30 Static decode tree.

2.3.6 P&E Tree

The second type of address tree uses dynamic logic, the most prevalent being *precharge and evaluate* (P&E) logic. Used by the majority of DRAM manufacturers, P&E address trees come in a variety of configurations, although the differences between them can be subtle. Figure 2.31 shows a simplified schematic for one version of a P&E address tree designed for use with bootstrapped wordline drivers. P&E address tree circuits feature one or

more PMOS *PRECHARGE* transistors and a cascade of NMOS *ENABLE* transistors M2–M4. This P&E design uses half of the transistors required by the static address tree of Figure 2.30. As a result, the layout of the P&E tree is much smaller than that of the static tree and fits more easily under the address lines. The *PRE* transistor is usually driven by a *PRECHARGE** signal under the control of the *RAS** chain logic. *PRECHARGE** and transistor M1 ensure that *DEC** is precharged HIGH, disabling the wordline driver and preparing the tree for row address activation.

M7 is a weak PMOS transistor driven by the *DEC* inverter (M5 and M6). Together, M7 and the inverter form a latch to ensure that *DEC** remains HIGH for all decoders that are not selected by the row addresses. At the beginning of any *RAS** cycle, *PRECHARGE** is LOW and the row addresses are all disabled (LOW). After *RAS** falls, *PRECHARGE** transitions HIGH to turn OFF M1; then the row addresses are enabled. If *RA*1–*RA*3 all go HIGH, then M2–M4 turn ON, overpowering M7 and driving *DEC** to ground and subsequently *DEC* to V_{CC}. The output of this tree segment normally drives four bootstrapped wordline drivers, each connected to a separate *PHASE* signal. Therefore, for an array with 256 wordlines, there will be 64 such decode trees.

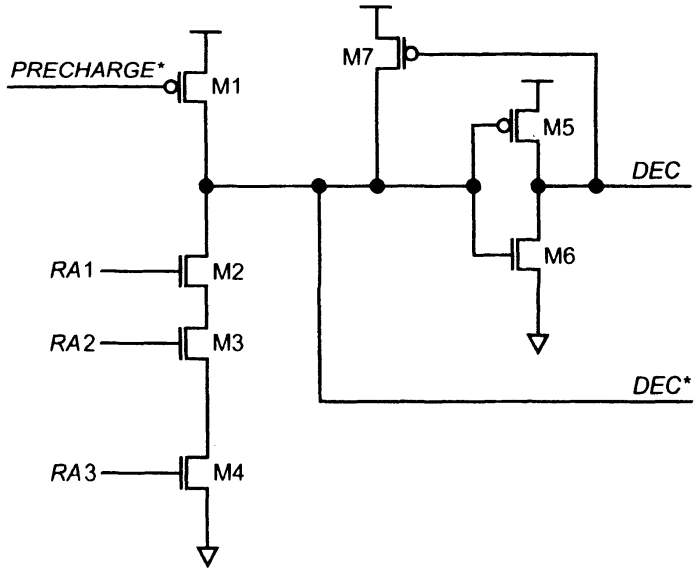


Figure 2.31 P&E decode tree.

2.3.7 Predecoding

The row address lines shown as *RA*1–*RA*3 can be either true and complement or predecoded. Predecoded address lines are formed by logically combining (AND) addresses as shown in Table 2.1.

The advantages of using predecoded addresses include lower power (fewer signals make transitions during address changes) and higher efficiency (only three transistors are necessary to decode six addresses for the circuit of Figure 2.31). Predecoding is especially beneficial in redundancy circuits. In fact, predecoded addresses are used throughout most DRAM designs today.

Table 2.1 Predecoded address truth table.

RA0	RA1	PR01<n>	PR01<0>	PR01<1>	PR01<2>	PR01<3>
0	0	0	1	0	0	0
1	0	1	0	1	0	0
0	1	2	0	0	1	0
1	1	3	0	0	0	1

2.3.8 Pass Transistor Tree

The final type of address tree to be examined, shown in Figure 2.32, uses pass transistor logic. Pass transistor address trees are similar to P&E trees in numerous ways. Both designs use PMOS *PRECHARGE* transistors and NMOS address *ENABLE* transistors. Unlike P&E logic, however, the NMOS cascade does not terminate at ground. Rather, the cascade of M2–M4 goes to a *PHASE** signal, which is HIGH during *PRECHARGE* and LOW during *EVALUATE*. The address signals operate the same as in the P&E tree: HIGH to select and LOW to deselect. The pass transistor tree is shown integrated into a CMOS wordline driver. This is necessary because the pass transistor tree and the CMOS wordline driver are generally used together and their operation is complementary. The cross-coupled PMOS transistors of the CMOS level translator provide a latch necessary to keep the final interstage node biased at V_{CC}. Again the latch has a weak pull-up, which is easily overpowered by the cascaded NMOS *ENABLE* transistors. A pass transistor address tree is not used with bootstrapped wordline drivers because the *PHASE* signal feeds into the address tree logic rather than into the driver, as required by the bootstrap driver.

2.4 DISCUSSION

We have briefly examined the basic elements required in DRAM row decoder blocks. Numerous variations are possible. No single design is best for all applications. As with sense amplifiers, design depends on technology and performance and cost trade-offs.

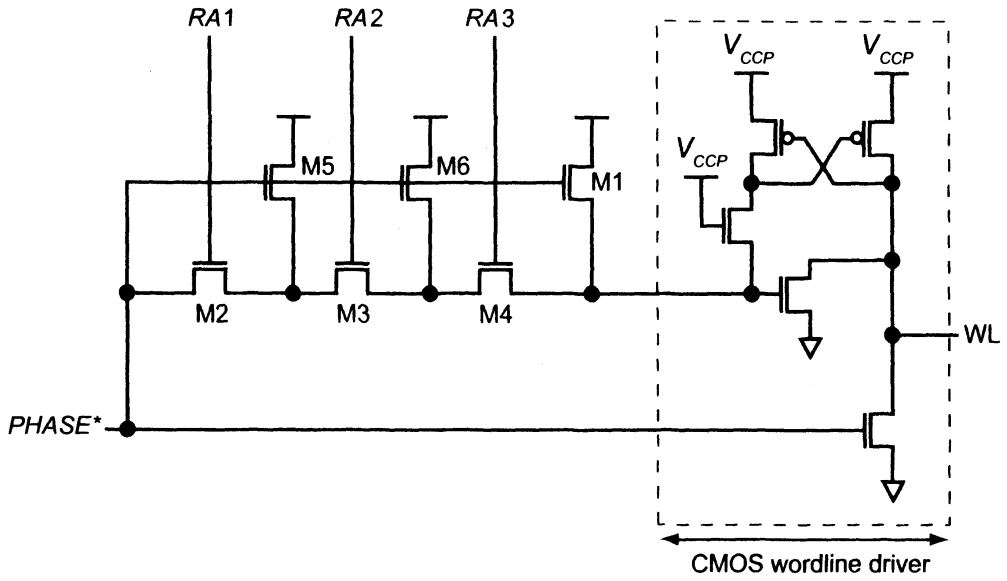


Figure 2.32 Pass transistor decode tree.

REFERENCES

[1] K. Itoh, "Trends in megabit DRAM circuit design," *IEEE Journal of Solid-State Circuits,* vol. 25, pp. 778–791, June 1990.

[2] D. Takashima, S. Watanabe, H. Nakano, Y. Oowaki, and K. Ohuchi, "Open/folded bit-line arrangement for ultra-high-density DRAMs," *IEEE Journal of Solid-State Circuits,* vol. 29, pp. 539–542, April 1994.

[3] Hideto Hidaka, Yoshio Matsuda, and Kazuyasu Fujishima, "A divided/shared bit-line sensing scheme for ULSI DRAM cores," *IEEE Journal of Solid-State Circuits,* vol. 26, pp. 473–477, April 1991.

[4] M. Aoki, Y. Nakagome, M. Horiguchi, H. Tanaka, S. Ikenaga, J. Etoh, Y. Kawamoto, S. Kimura, E. Takeda, H. Sunami, and K. Itoh, "A 60-ns 16-Mbit CMOS DRAM with a transposed data-line structure," *IEEE Journal of Solid-State Circuits,* vol. 23, pp. 1113–1119, October 1988.

[5] R. Kraus and K. Hoffmann, "Optimized sensing scheme of DRAMs," *IEEE Journal of Solid-State Circuits,* vol. 24, pp. 895–899, August 1989.

[6] T. Yoshihara, H. Hidaka, Y. Matsuda, and K. Fujishima, "A twisted bitline technique for multi-Mb DRAMs," in *1988 IEEE ISSCC Digest of Technical Papers,* pp. 238–239.

[7] Yukihito Oowaki, Kenji Tsuchida, Yohji Watanabe, Daisaburo Takashima, Masako Ohta, Hiroaki Nakano, Shigeyoshi Watanabe, Akihiro Nitayama, Fumio Horiguchi, Kazunori Ohuchi, and Fujio Masuoka, "A 33-ns 64Mb DRAM," *IEEE Journal of Solid-State Circuits,* vol. 26, pp. 1498–1505, November 1991.

[8] M. Inoue, H. Kotani, T. Yamada, H. Yamauchi, A. Fujiwara, J. Matsushima, H. Akamatsu, M. Fukumoto, M. Kubota, I. Nakao, N. Aoi, G. Fuse, S. Ogawa, S. Odanaka. A. Ueno, and Y. Yamamoto, "A 16Mb DRAM with an open bit-line architecture," in *1988 IEEE ISSCC Digest of Technical Papers,* pp. 246–247.

[9] Y. Kubota, Y. Iwase, K. Iguchi, J. Takagi, T. Watanabe, and K. Sakiyama, "Alternatively-activated open bitline technique for high density DRAM's," *IEICE Trans. Electron.,* vol. E75-C, pp. 1259–1266, October 1992.

[10] T. Hamada, N. Tanabe, H. Watanabe, K. Takeuchi, N. Kasai, H. Hada, K. Shibahara, K. Tokashiki, K. Nakajima, S. Hirasawa, E. Ikawa, T. Saeki, E. Kakehashi, S. Ohya, and T. Kunio, "A split-level diagonal bit-line (SLDB) stacked capacitor cell for 256Mb DRAMs," *1992 IEDM Technical Digest,* pp. 799–802.

[11] Toshinori Morihara, Yoshikazu Ohno, Takahisa Eimori, Toshiharu Katayama, Shinichi Satoh, Tadashi Nishimura, and Hirokazu Miyoshi, "Disk-shaped stacked capacitor cell for 256Mb dynamic random-access memory," *Japan Journal of Applied Physics,* vol. 33, Part 1, pp. 4570–4575, August 1994.

[12] J. H. Ahn, Y. W. Park, J. H. Shin, S. T. Kim, S. P. Shim, S. W. Nam, W. M. Park, H. B. Shin, C. S. Choi, K. T. Kim, D. Chin, O. H. Kwon, and C. G. Hwang, "Micro villus patterning (MVP) technology for 256Mb DRAM stack cell," in *1992 Symposium on VLSI Tech. Digest of Technical Papers*, pp. 12–13.

[13] Kazuhiko Sagara, Tokuo Kure, Shoji Shukuri, Jiro Yugami, Norio Hasegawa, Hidekazu Goto, and Hisaomi Yamashita, "Recessed memory array technology for a double cylindrical stacked capacitor cell of 256M DRAM," *IEICE Trans. Electron.,* vol. E75-C, pp. 1313–1322, November 1992.

[14] S. Ohya. 1994. Semiconductor memory device having stacked-type capacitor of large capacitance. US Patent 5,298,775, March 29, 1994.

[15] M. Sakao, N. Kasai, T. Ishijima, E. Ikawa, H. Watanabe, K. Terada, and T. Kikkawa, "A capacitor-over-bit-line (COB) cell with hemispherical-grain storage node for 64Mb DRAMs," in *1990 IEDM Technical Digest,* pp. 655–658.

[16] G. Bronner, H. Aochi, M. Gall, J. Gambino, S. Gernhardt, E. Hammerl, H. Ho, J. Iba, H. Ishiuchi, M. Jaso, R. Kleinhenz, T. Mii, M. Narita, L. Nesbit, W. Neumueller, A. Nitayama, T. Ohiwa, S. Parke, J. Ryan, T. Sato, H. Takato, and S. Yoshikawa, "A fully planarized 0.25μm CMOS technology for 256Mbit DRAM and beyond," in *1995 Symposium on VLSI Tech. Digest of Technical Papers*, pp. 15–16.

[17] N. C.-C. Lu and H. H. Chao, "Half-V_{DD} / bit-line sensing scheme in CMOS DRAMs," in *IEEE Journal of Solid-State Circuits,* vol. SC19, p. 451, August 1984.

[18] E. Adler; J. K. DeBrosse; S. F. Geissler; S. J. Holmes; M. D. Jaffe; J. B. Johnson; C. W. Koburger, III; J. B. Lasky; B. Lloyd; G. L. Miles;

J. S. Nakos; W. P. Noble, Jr.; S. H. Voldman; M. Armacost; and R. Ferguson; "The evolution of IBM CMOS DRAM technology," *IBM Journal of Research and Development,* vol. 39, pp. 167–188, March 1995.

[19] R. Kraus, "Analysis and reduction of sense-amplifier offset," in *IEEE Journal of Solid-State Circuits,* vol. 24, pp. 1028–1033, August 1989.

[20] M. Asakura, T. Ohishi, M. Tsukude, S. Tomishima, H. Hidaka, K. Arimoto, K. Fujishima, T. Eimori, Y. Ohno, T. Nishimura, M. Yasunaga, T. Kondoh, S. I. Satoh, T. Yoshihara, and K. Demizu, "A 34ns 256Mb DRAM with boosted sense-ground scheme," in *1994 IEEE ISSCC Digest of Technical Papers,* pp. 140–141.

[21] T. Ooishi, K. Hamade, M. Asakura, K. Yasuda, H. Hidaka, H. Miyamoto, and H. Ozaki, "An automatic temperature compensation of internal sense ground for sub-quarter micron DRAMs," in *1994 Symposium on VLSI Circuits Digest of Technical Papers*, pp. 77–78.

[22] K. Noda, T. Saeki, A. Tsujimoto, T. Murotani, and K. Koyama, "A boosted dual word-line decoding scheme for 256 Mb DRAMs," in *1992 Symposium on VLSI Circuits Digest of Technical Papers*, pp. 112–113.

Chapter
3

Array Architectures

This chapter presents a detailed description of the two most prevalent array architectures under consideration for future large-scale DRAMs: the aforementioned open architectures and olded digitline architectures.

3.1 TYPES OF ARRAY ARCHITECTURES

To provide a viable point for comparison, each architecture is employed in the theoretical construction of 32-Mbit memory blocks for use in a 256-Mbit DRAM. Design parameters and layout rules from a typical 0.25 μm DRAM process provide the necessary dimensions and constraints for the analysis. Some of these parameters are shown in Table 3.1. By examining DRAM architectures in the light of a real-world design problem, an objective and unbiased comparison can be made. In addition, using this approach, we readily detect the strengths and weaknesses of either architecture.

3.1.1 Open Digitline Array Architecture

The open digitline array architecture was the prevalent architecture prior to the 64kbit DRAM. A modern embodiment of this architecture, as shown in Figure 3.1 [1] [2], is constructed with multiple crosspoint array cores separated by strips of sense amplifier blocks in one axis and either row decode blocks or wordline stitching regions in the other axis. Each 128kbit array core is built using $6F^2$ Mbit cell pairs. There are 131,072 (2^{17}) functionally addressable Mbits arranged in 264 rows and 524 digitlines. In the 264 rows, there are 256 actual wordlines, 4 redundant wordlines, and 4 dummy wordlines. In the 524 digitlines, there are 512 actual digitlines, 8 redundant digitlines, and 4 dummy digitlines. Photolithography problems usually occur at the edge of large repetitive structures, such as Mbit arrays.

These problems produce malformed or nonuniform structures, rendering the edge cells useless. Therefore, including dummy Mbits, wordlines, and digitlines on each array edge ensures that these problems occur only on dummy cells, leaving live cells unaffected. Although dummy structures enlarge each array core, they also significantly improve device yield. Thus, they are necessary on all DRAM designs.

Table 3.1 0.25 μm design parameters.

Parameter	Value
Digitline width W_{DL}	0.3 μm
Digitline pitch P_{DL}	0.6 μm
Wordline width W_{WL}	0.3 μm
Wordline pitch for $8F^2$ Mbit P_{WL8}	0.6 μm
Wordline pitch for $6F^2$ Mbit P_{WL6}	0.9 μm
Cell capacitance C_C	30fF
Digitline capacitance per Mbit C_{DM}	0.8fF
Wordline capacitance per $8F^2$ Mbit C_{W8}	0.6fF
Wordline capacitance per $6F^2$ Mbit C_{W6}	0.5fF
Wordline sheet resistance R_S	6Ω/sq

Array core size, as measured in the number of Mbits, is restricted by two factors: a desire to keep the quantity of Mbits a power of two and the practical limits on wordline and digitline length. The need for a binary quantity of Mbits in each array core derives from the binary nature of DRAM addressing. Given N row addresses and M column addresses for a given part, there are 2^{N+M} addressable Mbits. Address decoding is greatly simplified within a DRAM if array address boundaries are derived directly from address bits.

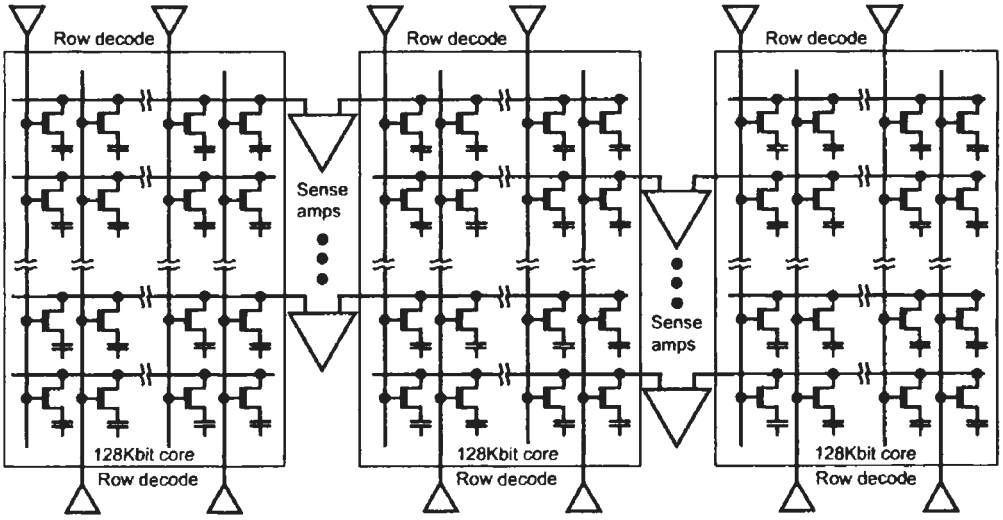


Figure 3.1 Open digitline architecture schematic.

Because addressing is binary, the boundaries naturally become binary. Therefore, the size of each array core must necessarily have 2^X addressable rows and 2^Y addressable digitlines. The resulting array core size is 2^{X+Y} Mbits, which is, of course, a binary number.

The second set of factors limiting array core size involves practical limits on digitline and wordline length. From earlier discussions in Section 2.1, the digitline capacitance is limited by two factors. First, the ratio of cell capacitance to digitline capacitance must fall within a specified range to ensure reliable sensing. Second, operating current and power for the DRAM is in large part determined by the current required to charge and discharge the digitlines during each active cycle. Power considerations restrict digitline length for the 256-Mbit generation to approximately 128 Mbit pairs (256 rows), with each Mbit connection adding capacitance to the digitline. The power dissipated during a Read or Refresh operation is proportional to the *digitline capacitance* (C_D), the supply voltage (V_{CC}), the number of active columns (N), and the Refresh period (P). Accordingly, the power dissipated is given as

$$P_D = V_{CC}2 \cdot \frac{N(C_D + C_C)}{2P} \; watts \tag{3.1}$$

On a 256-Mbit DRAM in 8k (rows) Refresh, there are 32,768 (2^{15}) active columns during each Read, Write, or Refresh operation. The active array current and power dissipation for a 256-Mbit DRAM appear in Table 3.2 for a 90 ns Refresh period (–5 timing) at various digitline lengths. The bud-

get for the active array current is limited to 200mA for this 256-Mbit design. To meet this budget, the digitline cannot exceed a length of 256 Mbits.

Wordline length, as described in Section 2.1, is limited by the maximum allowable *RC* time constant of the wordline. To ensure acceptable access time for the 256-Mbit DRAM, the wordline time constant should be kept below 4 nanoseconds. For a wordline connected to *N* Mbits, the total resistance and capacitance follow:

$$R_{WL} = \frac{(R_S \cdot N \cdot P_{DL})}{W_{LW}} \tag{3.2}$$

$$C_{WL} = C_{W8} \cdot N farads \tag{3.3}$$

where P_{DL} is the digitline pitch, W_{LW} is the wordline width, and C_{W8} is the wordline capacitance in an $8F^2$ Mbit cell.

Table 3.2 Active current and power versus digitline length.

Digitline Length (Mbits)	Digitline Capacitance (fF)	Active Current (mA)	Power Dissipation (mW)
128	102	60	199
256	205	121	398
512	410	241	795

Table 3.3 contains the effective wordline time constants for various wordline lengths. As shown in the table, the wordline length cannot exceed 512 Mbits (512 digitlines) if the wordline time constant is to remain under 4 nanoseconds.

The open digitline architecture does not support digitline twisting because the true and complement digitlines, which constitute a column, are in separate array cores. Therefore, no silicon area is consumed for twist regions. The 32-Mbit array block requires a total of two hundred fifty-six 128kbit array cores in its construction. Each 32-Mbit block represents an address space comprising a total of 4,096 rows and 8,192 columns. A practical configuration for the 32-Mbit block is depicted in Figure 3.2.

Table 3.3 Wordline time constant versus wordline length.

Wordline Length (Mbits)	R_{WL} (ohms)	C_{WL} (fF)	Time Constant (ns)
128	1,536	64	0.098
256	3,072	128	0.39
512	6,144	256	1.57
1024	12,288	512	6.29

In Figure 3.2, the 256 array cores appear in a 16 x 16 arrangement. The x16 arrangement produces 2-Mbit sections of 256 wordlines and 8,192 digitlines (4,096 columns). Sixteen 2-Mbit sections are required to form the complete 32-Mbit block: sense amplifier strips are positioned vertically between each 2-Mbit section, and row decode strips or wordline stitching strips are positioned horizontally between each array core.

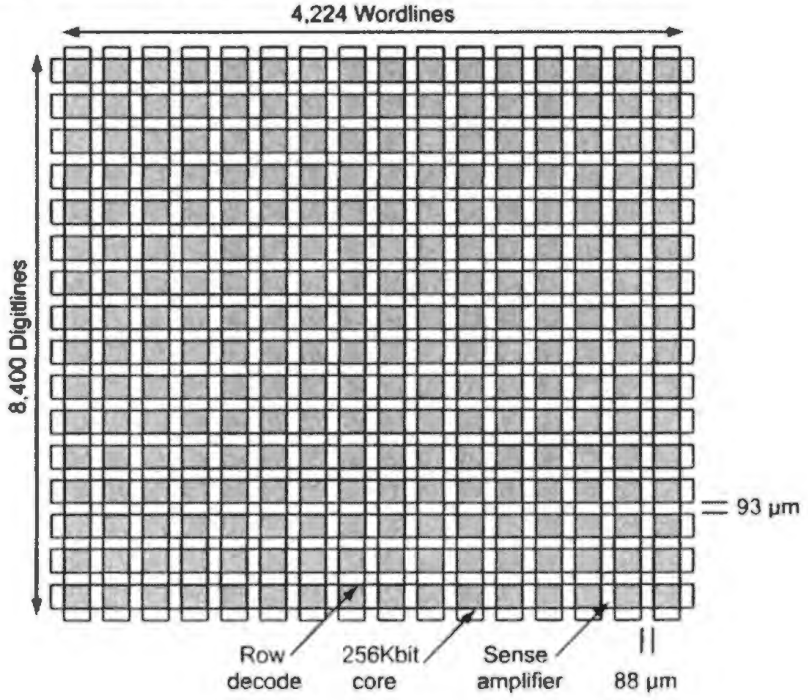


Figure 3.2 Open digitline 32-Mbit array block.

Layout can be generated for the various 32-Mbit elements depicted in Figure 3.2. This layout is necessary to obtain reasonable estimates for pitch cell size. With these size estimates, overall dimensions of the 32-Mbit memory block can be calculated. The results of these estimates appear in

Table 3.4. Essentially, the overall height of the 32-Mbit block can be found by summing the height of the row decode blocks (or stitch regions) together with the product of the digitline pitch and the total number of digitlines.

Accordingly,

$$Height32 = (T_R \cdot H_{LDEC}) + (T_{DL} \cdot P_{DL})\ microns \tag{3.4}$$

where T_R is the number of local row decoders, H_{LDEC} is the height of each decoder, T_{DL} is the number of digitlines including redundant and dummy lines, and P_{DL} is the digitline pitch. Similarly, the width of the 32-Mbit block is found by summing the total width of the sense amplifier blocks with the product of the wordline pitch and the number of wordlines. This bit of math yields

$$Width32 = (T_{SA} \cdot W_{AMP}) + (T_{WL} \cdot P_{WL6})\ microns \tag{3.5}$$

where T_{SA} is the number of sense amplifier strips, W_{AMP} is the width of the sense amplifiers, T_{WL} is the total number of wordlines including redundant and dummy lines, and P_{WL6} is the wordline pitch for the $6F^2$ Mbit.

Table 3.4 contains calculation results for the 32-Mbit block shown in Figure 3.2. Although overall size is the best measure of architectural efficiency, a second popular metric is array efficiency. Array efficiency is determined by dividing the area consumed by functionally addressable Mbits by the total die area. To simplify the analysis in this text, peripheral circuits are ignored in the array efficiency calculation. Rather, the calculation considers only the 32-Mbit memory block, ignoring all other factors. With this simplification, the array efficiency for a 32-Mbit block is given as

$$Efficiency = \frac{(100 \cdot 2^{25} \cdot P_{DL} \cdot P_{WL6})}{Area32}\ percent \tag{3.6}$$

where 2^{25} is the number of addressable Mbits in each 32-Mbit block. The open digitline architecture yields a calculated array efficiency of 51.7%.

Unfortunately, the ideal open digitline architecture presented in Figure 3.2 is difficult to realize in practice. The difficulty stems from an interdependency between the memory array and sense amplifier layouts in which each array digitline must connect to one sense amplifier and each sense amplifier must connect to two array digitlines.

Table 3.4 Open digitline (local row decode): 32-Mbit size calculations.

Description	Parameter	Size
Number of sense amplifier strips	T_{SA}	17
Width of sense amplifiers	W_{AMP}	88 μm
Number of local decode strips	T_{LDEC}	17
Height of local decode strips	H_{LDEC}	93 μm
Number of digitlines	T_{DL}	8,400
Number of wordlines	T_{WL}	4,224
Height of 32-Mbit block	*Height*32	6,621 μm
Width of 32-Mbit block	*Width*32	5,298 μm
Area of 32-Mbit block	*Area*32	35,078,058 μm^2

This interdependency, which exists for all array architectures, becomes problematic for the open digitline architecture. The two digitlines, which connect to a sense amplifier, must come from two separate memory arrays. As a result, sense amplifier blocks must always be placed between memory arrays for open digitline array architectures [3], unlike the depiction in Figure 3.2.

Two layout approaches may be used to achieve this goal. First, design the sense amplifiers so that the sense amplifier block contains a set of sense amplifiers for each digitline in the array. This single-pitch solution, shown in Figure 3.3, eliminates the need for sense amplifiers on both sides of an array core because all of the digitlines connect to a single sense amplifier block. Not only does this solution eliminate the edge problem, but it also reduces the 32-Mbit block size. There are now only eight sense amplifier strips instead of the seventeen of Figure 3.2. Unfortunately, it is nearly impossible to lay out sense amplifiers in this fashion [4]. Even a single-metal sense amplifier layout, considered the tightest layout in the industry, achieves only one sense amplifier for every two digitlines (double-pitch).

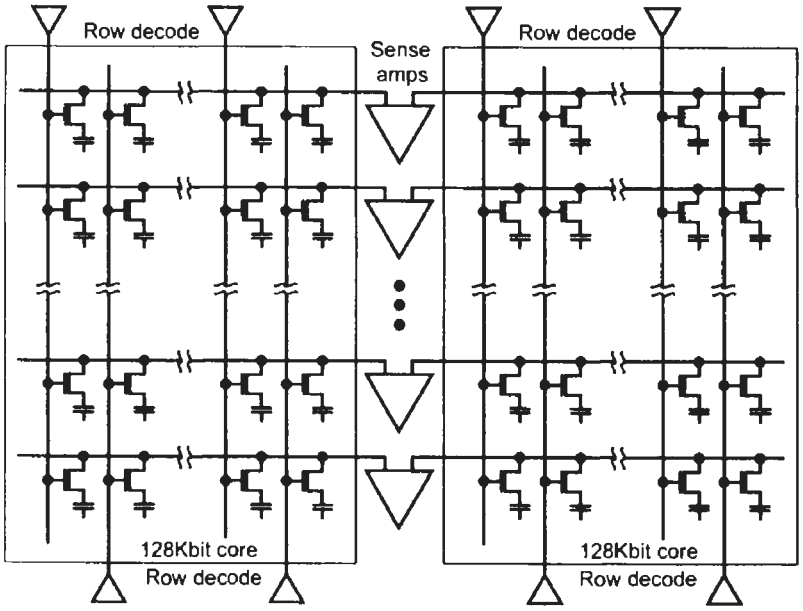


Figure 3.3 Single-pitch open digitline architecture.

A second approach to solving the interdependency problem in open digitline architectures is to maintain the configuration shown in Figure 3.2 but include some form of reference digitline for the edge sense amplifiers. The reference digitline can assume any form as long as it accurately models the capacitance and behavior of a true digitline. Obviously, the best type of reference digitline is a true digitline. Therefore, with this approach, more dummy array cores are added to both edges of the 32-Mbit memory block, as shown in Figure 3.4. The dummy array cores need only half as many wordlines as the true array core because only half of the digitlines are connected to any single sense amplifier strip. The unconnected digitlines double the effective length of the reference digitlines.

This approach solves the array edge problem. However, by producing a larger 32-Mbit memory block, array efficiency is reduced. Dummy arrays solve the array edge problem inherent in open digitline architecture but require sense amplifier layouts that are on the edge of impossible. The problem of sense amplifier layout is all the more difficult because global column select lines must be routed through. For all intents and purposes, therefore, the sense amplifier layout cannot be completed without the presence of an additional conductor, such as a third metal, or without time multiplexed sensing. Thus, for the open digitline architecture to be successful, an additional conductor must be added to the DRAM process.

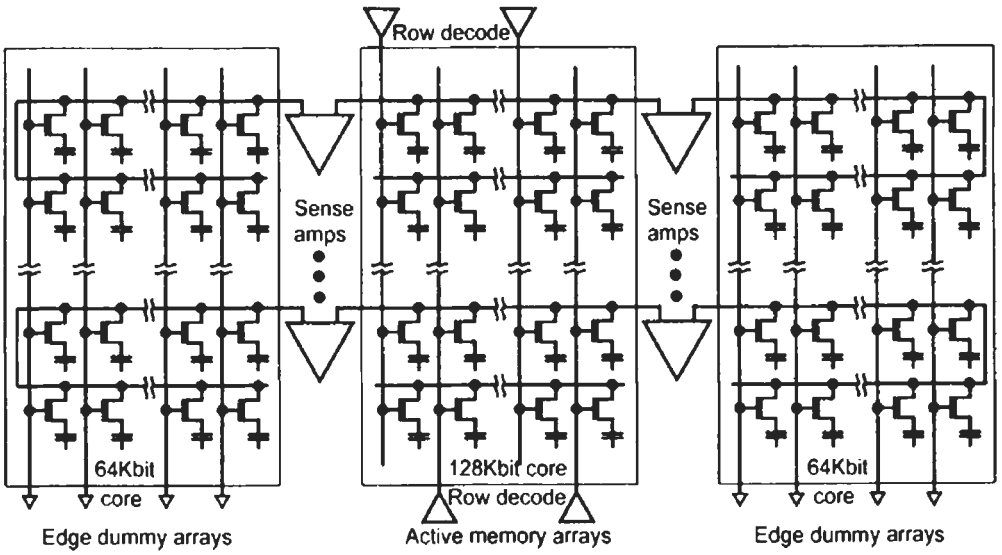


Figure 3.4 Open digitline architecture with dummy arrays.

With the presence of the additional conductor, such as Metal3, the sense amplifier layout and either a full or hierarchical global row decoding scheme is made possible. A full global row decoding scheme using wordline stitching places great demands on metal and contact/via technologies; however, it represents the most efficient use of the additional metal. Hierarchical row decoding using bootstrap wordline drivers is slightly less efficient. Wordlines no longer need to be strapped with metal on pitch, and, thus, process requirements are relaxed significantly [5].

For a balanced perspective, both global and hierarchical approaches are analyzed. The results of this analysis for the open digitline architecture are summarized in Tables 3.5 and 3.6. Array efficiency for global and hierarchical row decoding calculate to 60.5% and 55.9%, respectively, for the 32 Mbit memory blocks based on data from these tables.

Table 3.5 Open digitline (dummy arrays and global row decode): 32-Mbit size calculations.

Description	Parameter	Size
Number of sense amplifier strips	T_{SA}	17
Width of sense amplifiers	W_{AMP}	88 μm
Number of global decode strips	T_{GDEC}	1
Height of global decode strips	H_{GDEC}	200 μm
Number of stitch regions	N_{ST}	17
Height of stitch regions	H_{ST}	10 μm
Number of digitlines	T_{DL}	8,400
Number of wordlines	T_{WL}	4,488
Height of 32-Mbit block	*Height*32	5,410 μm
Width of 32-Mbit block	*Width*32	5,535 μm
Area of 32-Mbit block	*Area*32	29,944,350 μm^2

Table 3.6 Open digitline (dummy arrays and hierarchical row decode): 32-Mbit size calculations.

Description	Parameter	Size
Number of sense amplifier strips	T_{SA}	17
Width of sense amplifiers	W_{AMP}	88 μm
Number of global decode strips	T_{GDEC}	1
Height of global decode strips	H_{GDEC}	190 μm
Number of hier decode strips	T_{HDEC}	17
Height of hier decode strips	H_{HDEC}	37 μm
Number of digitlines	T_{DL}	8,400
Number of wordlines	T_{WL}	4,488
Height of 32-Mbit block	*Height*32	5,859 μm
Width of 32-Mbit block	*Width*32	5,535 μm
Area of 32-Mbit block	*Area*32	32,429,565 μm^2

3.1.2 Folded Array Architecture

The folded array architecture depicted in Figure 3.5 is the standard architecture of many modern DRAM designs. The folded architecture is constructed with multiple array cores separated by strips of sense amplifiers and either row decode blocks or wordline stitching regions. Unlike the open digitline architecture, which uses 6F^2 Mbit cell pairs, the folded array core uses 8F^2 Mbit cell pairs [6]. Modern array cores include 262,144 (2^{18}) functionally addressable Mbits arranged in 532 rows and 1,044 digitlines. In the 532 rows, there are 512 actual wordlines, 4 redundant wordlines, and 16 dummy wordlines. Each row (wordline) connects to Mbit transistors on alternating digitlines. In the 1,044 digitlines, there are 1,024 actual digitlines (512 columns), 16 redundant digitlines (8 columns), and 4 dummy digitlines. As discussed in Section 3.1.1, because of the additive or subtractive photolithography effects, dummy wordlines and digitlines are necessary to

guardband live digitlines. These photo effects are pronounced at the edges of large repetitive structures such as the array cores.

Sense amplifier blocks are placed on both sides of each array core. The sense amplifiers within each block are laid out at quarter pitch: one sense amplifier for every four digitlines. Each sense amplifier connects through isolation devices to columns (digitline pairs) from both adjacent array cores. Odd columns connect on one side of the core, and even columns connect on the opposite side. Each sense amplifier block is therefore connected to only odd or even columns and is never connected to both odd *and* even columns within the same block. Connecting to both odd and even columns requires a half-pitch sense amplifier layout: one sense amplifier for every two digitlines. While half-pitch layout is possible with certain DRAM processes, the bulk of production DRAM designs remain quarter pitch due to the ease of laying them out. The analysis presented in this chapter is accordingly based on quarter-pitch design practices.

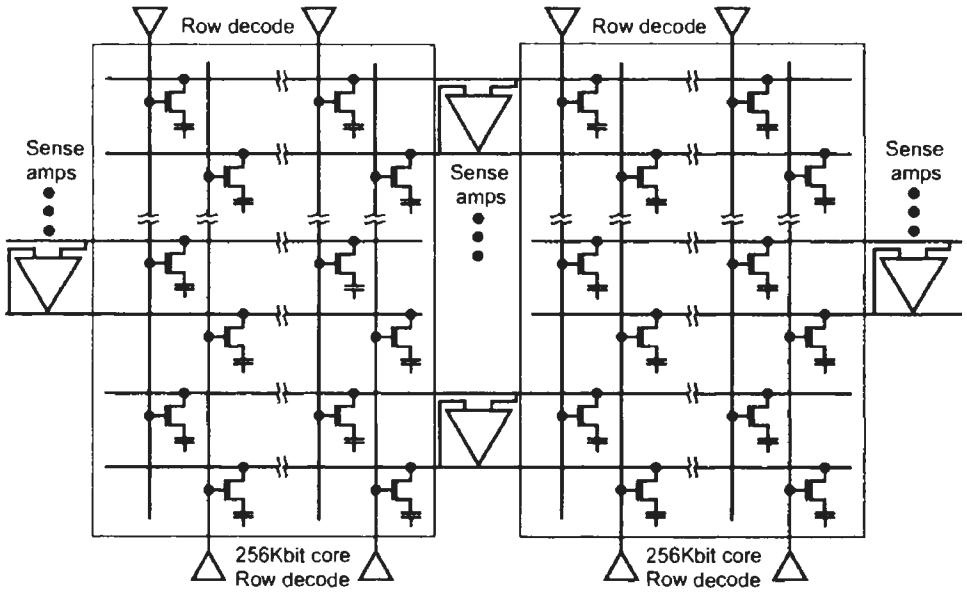


Figure 3.5 Folded digitline array architecture schematic.

The location of row decode blocks for the array core depends on the number of available metal layers. For one- and two-metal processes, local row decode blocks are located at the top and bottom edges of the core. For three- and four-metal processes, global row decodes are used. Global row decodes require only stitch regions or local wordline drivers at the top and bottom edges of the core [7]. Stitch regions consume much less silicon area than local row decodes, substantially increasing array efficiency for the DRAM. The array core also includes digitline twist regions running parallel to the wordlines. These regions provide the die area required for digitline twisting. Depending on the particular twisting scheme selected for a design

(see Section 2.1), the array core needs between one and three twist regions. For the sake of analysis, a triple twist is assumed, as it offers the best overall noise performance and has been chosen by some DRAM manufacturers for select advanced large-scale applications [8]. Because each twist region constitutes a break in the array structure, it is necessary to use dummy wordlines. For this reason, there are 16 dummy wordlines (2 for each array edge) in the folded array core rather than 4 dummy wordlines as in the open digitline architecture.

There are more Mbits in the array core for folded digitline architectures than there are for open digitline architectures. Larger core size is an inherent feature of folded architectures, arising from the very nature of the architecture. The term *folded architecture* comes from the fact that folding two open digitline array cores one on top of the other produces a folded array core. The digitlines and wordlines from each folded core are spread apart (double pitch) to allow room for the other folded core. After folding, each constituent core remains intact and independent, except for the Mbit changes ($8F^2$ conversion) necessary in the folded architecture. The array core size doubles because the total number of digitlines and wordlines doubles in the folding process. It does not quadruple as one might suspect because the two constituent folded cores remain independent: the wordlines from one folded core do not connect to Mbits in the other folded core.

Digitline pairing (column formation) is a natural outgrowth of the folding process; each wordline only connects to Mbits on alternating digitlines. The existence of digitline pairs (columns) is the one characteristic of folded digitline architectures that produces superior signal-to-noise performance. Furthermore, the digitlines that form a column are physically adjacent to one another. This feature permits various digitline twisting schemes to be used, as discussed in Section 2.1, further improving signal-to-noise performance.

Similar to the open digitline architecture, digitline length for the folded digitline architecture is again limited by power dissipation and minimum cell-to-digitline capacitance ratio. For the 256-Mbit generation, digitlines were restricted from connecting to more than 256 cells (128 Mbit pairs). The analysis used to arrive at this quantity is similar to that for the open digitline architecture. (Refer to Table 3.2 to view the calculated results of power dissipation versus digitline length for a 256-Mbit DRAM in 8k Refresh.) Wordline length is again limited by the maximum allowable *RC* time constant of the wordline.

Contrary to an open digitline architecture in which each wordline connects to Mbits on each digitline, the wordlines in a folded digitline architecture connect to Mbits only on alternating digitlines. Therefore, a wordline can cross 1,024 digitlines while connecting to only 512 Mbit transistors.

The wordlines have twice the overall resistance, but only slightly more capacitance because they run over field oxide on alternating digitlines. Table 3.7 presents the effective wordline time constants for various wordline lengths for a folded array core. For a wordline connected to N Mbits, the total resistance and capacitance follow:

$$R_{WL} = (2 \cdot R_S \cdot N \cdot P_{DL})\ ohms \tag{3.7}$$

$$C_{WL} = C_{W8} \cdot N\ farads \tag{3.8}$$

where P_{DL} is the digitline pitch and C_{W8} is the wordline capacitance in an $8F^2$ Mbit cell. As shown in Table 3.7, the wordline length cannot exceed 512 Mbits (1,024 digitlines) for the wordline time constant to remain under 4 nanoseconds. Although the wordline connects to only 512 Mbits, it is two times longer (1,024 digitlines) than wordlines in open digitline array cores. The folded digitline architecture therefore requires half as many row decode blocks or wordline stitching regions as the open digitline architecture.

Table 3.7 Wordline time constant versus wordline length (folded).

Wordline Length (Mbits)	R_{WL} (ohms)	C_{WL} (fF)	Time Constant (ns)
128	3,072	77	0.24
256	6,144	154	0.95
512	12,288	307	3.77
1,024	24,576	614	15.09

A diagram of a 32-Mbit array block using folded digitline architecture is shown in Figure 3.6. This block requires a total of one hundred twenty-eight 256kbit array cores. In this figure, the array cores are arranged in an 8-row and 16-column configuration. The x8 row arrangement produces 2-Mbit sections of 256 wordlines and 8,192 digitlines (4,096 columns). A total of sixteen 2-Mbit sections form the complete 32-Mbit array block. Sense amplifier strips are positioned vertically between each 2-Mbit section, as in the open digitline architecture. Again, row decode blocks or wordline stitching regions are positioned horizontally between the array cores.

The 32-Mbit array block shown in Figure 3.6 includes size estimates for the various pitch cells. The layout was generated where necessary to arrive at the size estimates. The overall size for the folded digitline 32-Mbit block can be found by again summing the dimensions for each component. Accordingly,

$$Height32 = (T_R \cdot H_{RDEC}) + (T_{DL} \cdot P_{DL})\ microns \tag{3.9}$$

where T_R is the number of row decoders, H_{RDEC} is the height of each decoder, T_{DL} is the number of digitlines including redundant and dummy, and P_{DL} is the digitline pitch. Similarly,

$$Width32 = (T_{SA} \cdot W_{AMP}) + (T_{WL} \cdot P_{WL8}) + (T_{TWIST} \cdot W_{TWIST})\ microns \tag{3.10}$$

where T_{SA} is the number of sense amplifier strips, W_{AMP} is the width of the sense amplifiers, T_{WL} is the total number of wordlines including redundant and dummy, P_{WL8} is the wordline pitch for the $8F^2$ Mbit, T_{TWIST} is the total number of twist regions, and W_{TWIST} is the width of the twist regions.

Table 3.8 shows the calculated results for the 32-Mbit block shown in Figure 3.6. In this table, a double-metal process is used, which requires local row decoder blocks. Note that Table 3.8 for the folded digitline architecture contains approximately twice as many wordlines as does Table 3.5 for the open digitline architecture. The reason for this is that each wordline in the folded array only connects to Mbits on alternating digitlines, whereas each wordline in the open array connects to Mbits on every digitline. A folded digitline design therefore needs twice as many wordlines as a comparable open digitline design.

Array efficiency for the 32-Mbit memory block from Figure 3.6 is again found by dividing the area consumed by functionally addressable Mbits by the total die area. For the simplified analysis presented in this text, the peripheral circuits are ignored. Array efficiency for the 32-Mbit block is therefore given as which yields 59.5% for the folded array design example.

$$Efficiency = \frac{(100 \cdot 2^{25} \cdot P_{DL} \cdot 2 \cdot P_{WL8})}{Area32}\ percent \tag{3.11}$$

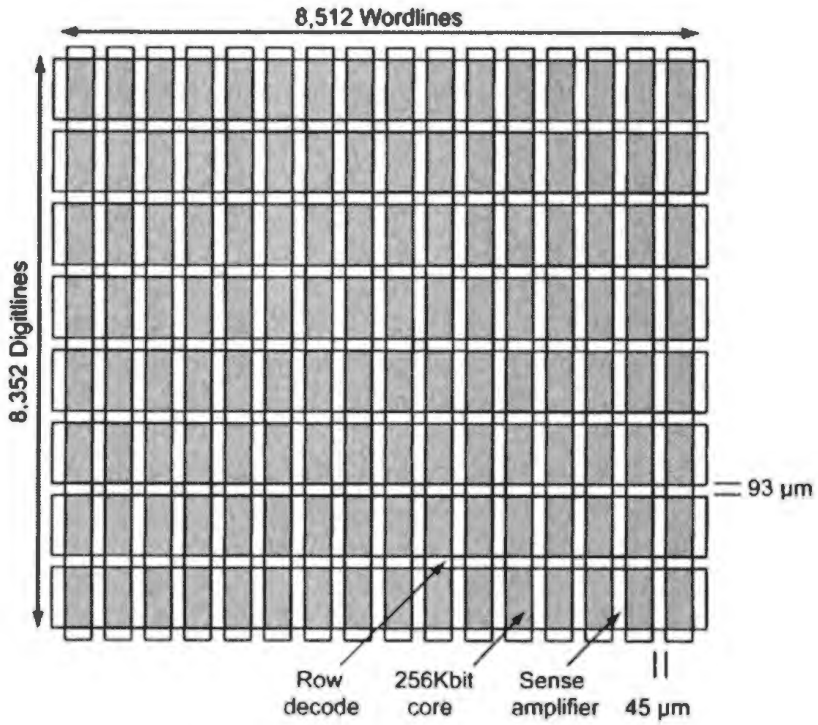


Figure 3.6 Folded digitline architecture 32-Mbit array block.

With the addition of Metal3 to the DRAM process, either a global or hierarchical row decoding scheme, similar to the open digitline analysis, can be used. While global row decoding and stitched wordlines achieve the smallest die size, they also place greater demands on the fabrication process. For a balanced perspective, both approaches were analyzed for the folded digitline architecture. The results of this analysis are presented in Tables 3.9 and 3.10. Array efficiency for the 32-Mbit memory blocks using global and hierarchical row decoding calculate to 74.0% and 70.9%, respectively.

Table 3.8 Folded digitline (local row decode): 32-Mbit size calculations.

Description	Parameter	Size
Number of sense amplifier strips	T_{SAF}	17
Width of sense amplifiers	W_{AMP}	45 μm
Number of local decode strips	T_{LDEC}	9
Height of local decode strips	H_{LDEC}	93 μm
Number of digitlines	T_{DL}	8,352
Number of wordlines	T_{WL1}	8,512
Number of twist regions	T_{TWIST}	48
Width of twist regions	W_{TWIST}	6 μm
Height of 32-Mbit block	*Height*32	6,592 μm
Width of 32-Mbit block	*Width*32	6,160 μm
Area of 32-Mbit block	*Area*32	40,606,720 μm^2

Table 3.9 Folded digitline (global decode): 32-Mbit size calculations.

Description	Parameter	Size
Number of sense amplifier strips	T_{SA}	17
Width of sense amplifiers	W_{AMP}	45 μm
Number of global decode strips	T_{GDEC}	1
Height of global decode strips	H_{GDEC}	200 μm
Number of stitch regions	N_{ST}	9
Height of stitch regions	H_{ST}	10 μm
Number of digitlines	T_{DL}	8,352
Number of wordlines	T_{WL}	8,512
Number of twist regions	T_{TWIST}	48
Width of twist regions	W_{TWIST}	6 μm
Height of 32-Mbit block	*Height*32	5,301 μm
Width of 32-Mbit block	*Width*32	6,160 μm
Area of 32-Mbit block	*Area*32	32,654,160 μm^2

Table 3.10 Folded digitline (hierarchical row decode): 32-Mbit size calculations.

Description	Parameter	Size
Number of sense amplifier strips	T_{SA}	17
Width of sense amplifiers	W_{AMP}	45 μm
Number of global decode strips	T_{GDEC}	1
Height of global decode strips	H_{GDEC}	190 μm
Number of hier decode strips	N_{HDEC}	9
Height of hier decode strips	H_{HEC}	37 μm
Number of digitlines	T_{DL}	8,352
Number of wordlines	T_{WL}	8,512
Number of twist regions	T_{TWIST}	48
Width of twist regions	W_{TWIST}	6 μm
Height of 32-Mbit block	*Height*32	5,534 μm
Width of 32-Mbit block	*Width*32	6,160 μm
Area of 32-Mbit block	*Area*32	34,089,440 μm^2

3.2 DESIGN EXAMPLES: ADVANCED BILEVEL DRAM ARCHITECTURE

This section introduces a novel, advanced architecture possible for use on future large-scale DRAMs. First, we discuss technical objectives for the proposed architecture. Second, we develop and describe the concept for an advanced array architecture capable of meeting these objectives. Third, we conceptually construct a 32-Mbit memory block with this new architecture

for a 256-Mbit DRAM. Finally, we compare the results achieved with the new architecture to those obtained for the open digitline and folded digitline architectures from Section 3.1.

3.2.1 Array Architecture Objectives

Both the open digitline and folded digitline architectures have distinct advantages and disadvantages. While open digitline architectures achieve smaller array layouts by virtue of using smaller $6F^2$ Mbit cells, they suffer from poor signal-to-noise performance. A relaxed wordline pitch, which stems from the $6F^2$ Mbit, simplifies the task of wordline driver layout. Sense amplifier layout, however, is difficult because the array configuration is inherently half pitch: one sense amplifier for every two digitlines. The superior signal-to-noise performance [9] of folded digitline architectures comes at the expense of larger, less efficient array layouts. Good signal-to-noise performance stems from the adjacency of true and complement digitlines and the capability to twist these digitline pairs. Sense amplifier layout is simplified because the array configuration is quarter pitch—that is, one sense amplifier for every four digitlines. Wordline driver layout is more difficult because the wordline pitch is effectively reduced in folded architectures.

The main objective of the new array architecture is to combine the advantages, while avoiding the disadvantages, of both folded and open digitline architectures. To meet this objective, the architecture needs to include the following features and characteristics:

- Open digitline Mbit configuration
- Small $6F^2$ Mbit
- Small, efficient array layout
- Folded digitline sense amplifier configuration
- Adjacent true and complement digitlines
- Twisted digitline pairs
- Relaxed wordline pitch
- High signal-to-noise ratio

An underlying goal of the new architecture is to reduce overall die size beyond that obtainable from either the folded or open digitline architectures. A second yet equally important goal is to achieve signal-to-noise performance that meets or approaches that of the folded digitline architecture.

3.2.2 Bilevel Digitline Construction

A bilevel digitline architecture resulted from 256-Mbit DRAM research and design activities carried out at Micron Technology, Inc., in Boise, Idaho. This bilevel digitline architecture is an innovation that evolved from a comparative analysis of open and folded digitline architectures. The analysis served as a design catalyst, ultimately leading to the creation of a new DRAM array configuration—one that allows the use of $6F^2$ Mbits in an otherwise folded digitline array configuration. These memory cells are a by-product of crosspoint-style (open digitline) array blocks. Crosspoint-style array blocks require that every wordline connect to Mbit transistors on every digitline, precluding the formation of digitline pairs. Yet, digitline pairs (columns) remain an essential element in folded digitline-type operation. Digitline pairs and digitline twisting are important features that provide for good signal-to-noise performance.

The bilevel digitline architecture solves the crosspoint and digitline pair dilemma through vertical integration. In vertical integration, essentially, two open digitline crosspoint array sections are placed side by side, as seen in Figure 3.7. Digitlines in one array section are designated as true digitlines, while digitlines from the second array section are designated as complement digitlines. An additional conductor is added to the DRAM process to complete the formation of the digitline pairs. The added conductor allows digitlines from each array section to route across the other array section with both true and complement digitlines vertically aligned. At the juncture between each section, the true and complement signals are vertically twisted. With this twisting, the true digitline connects to Mbits in one array section, and the complement digitline connects to Mbits in the other array section. This twisting concept is illustrated in Figure 3.8.

To improve the signal-to-noise characteristics of this design, the single twist region is replaced by three twist regions, as illustrated in Figure 3.9. A benefit of adding multiple twist regions is that only half of the digitline pairs actually twist within each region, making room in each region for the twists to occur. The twist regions are equally spaced at the 25%, 50%, and 75% marks in the overall array. Assuming that even digitline pairs twist at the 50% mark, odd digitlines twist at the 25% and 75% marks. Each component of a digitline pair, true and complement, spends half of its overall length on the bottom conductor connecting to Mbits and half of its length on the top conductor. This characteristic balances the capacitance and the number of Mbits associated with each digitline. Furthermore, the triple twisting scheme guarantees that the noise terms are balanced for each digitline, producing excellent signal-to-noise performance.

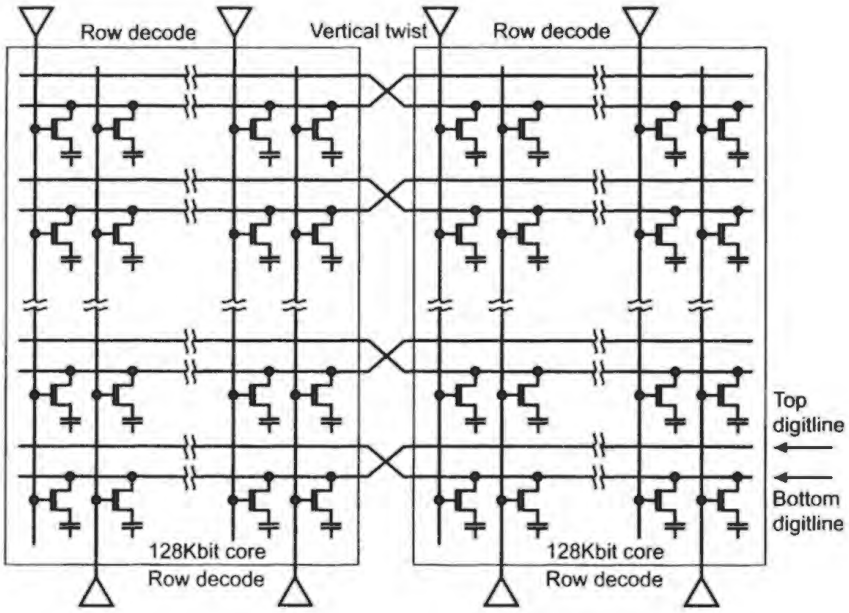


Figure 3.7 Development of bilevel digitline architecture.

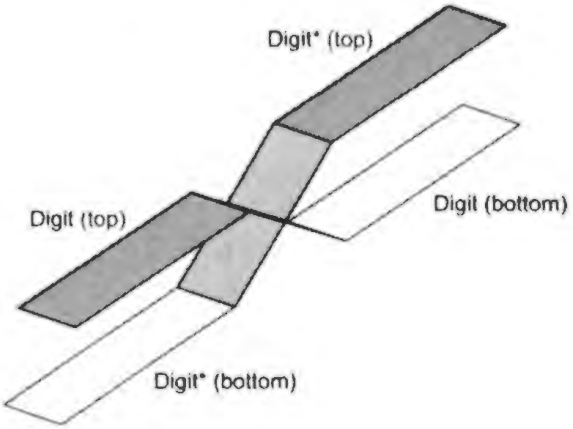


Figure 3.8 Digitline vertical twisting concept.

A variety of vertical twisting schemes is possible with the bilevel digitline architecture. As shown in Figure 3.10, each scheme uses conductive layers already present in the DRAM process to complete the twist. Vertical twisting is simplified because only half of the digitlines are involved in a given twist region. The final selection of a twisting scheme is based on yield factors, die size, and available process technology.

To further advance the bilevel digitline architecture concept, its $6F^2$ Mbit was modified to improve yield. Shown in an arrayed form in Figure 3.11, the *plaid* Mbit is constructed using long parallel strips of active

area vertically separated by traditional field oxide isolation. Wordlines run perpendicular to the active area in straight strips of polysilicon. Plaid Mbits are again constructed in pairs that share a common contact to the digitline. Isolation gates (transistors) formed with additional polysilicon strips provide horizontal isolation between Mbits. Isolation is obtained from these gates by permanently connecting the isolation gate polysilicon to either a ground or a negative potential. Using isolation gates in this Mbit design eliminates one- and two-dimensional encroachment problems associated with normal isolation processes. Furthermore, many photolithography problems are eliminated from the DRAM process as a result of the straight, simple design of both the active area and the polysilicon in the Mbit. The *plaid* designation for this Mbit is derived from the similarity between an array of Mbits and tartan fabric that is apparent in a color array plot.

In the bilevel and folded digitline architectures, both true and complement digitlines exist in the same array core. Accordingly, the sense amplifier block needs only one sense amplifier for every two digitline pairs. For the folded digitline architecture, this yields one sense amplifier for every four Metal1 digitlines—quarter pitch. The bilevel digitline architecture that uses vertical digitline stacking needs one sense amplifier for every two Metal1 digitlines—half pitch. Sense amplifier layout is therefore more difficult for bilevel than for folded designs. The three-metal DRAM process needed for bilevel architectures concurrently enables and simplifies sense amplifier layout. Metal1 is used for lower level digitlines and local routing within the sense amplifiers and row decodes. Metal2 is available for upper level digitlines and column select signal routing through the sense amplifiers. Metal3 can therefore be used for column select routing across the arrays and for control and power routing through the sense amplifiers. The function of Metal2 and Metal3 can easily be swapped in the sense amplifier block depending on layout preferences and design objectives.

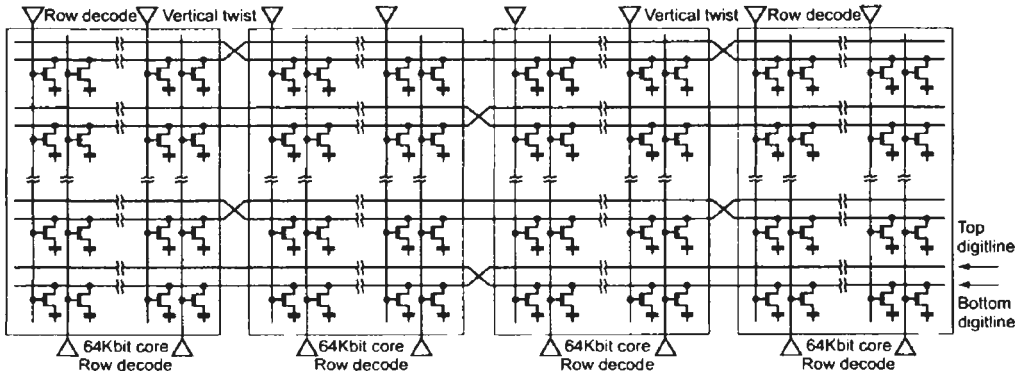


Figure 3.9 Bilevel digitline architecture schematic.

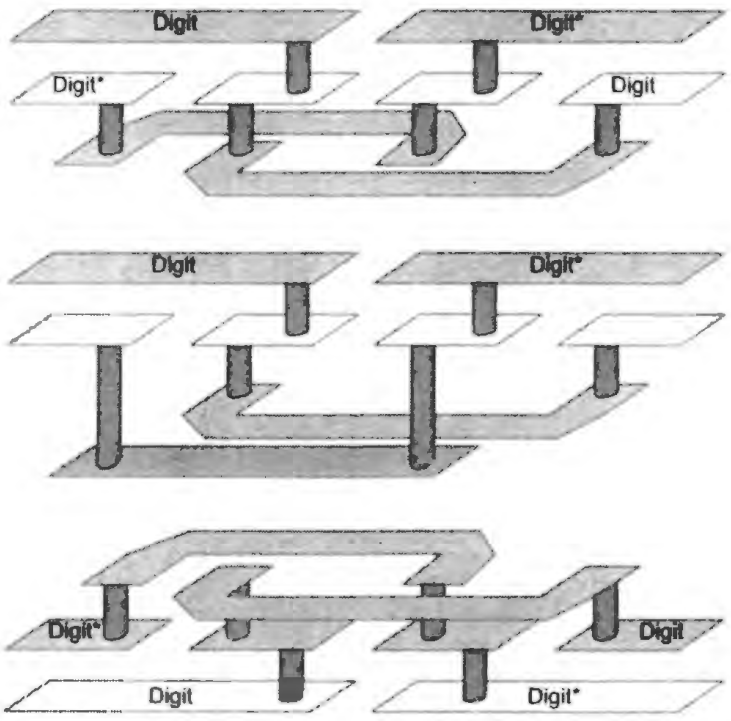


Figure 3.10 Vertical twisting schemes.

Wordline pitch is effectively relaxed for the plaid $6F^2$ Mbit of the bilevel digitline architecture. The Mbit is still built using the minimum process feature size of 0.3 μm. The relaxed wordline pitch stems from structural differences between a folded digitline Mbit and an open digitline or plaid Mbit. There are essentially four wordlines running across each folded digitline Mbit pair compared to two wordlines running across each open digitline or plaid Mbit pair. Although the plaid Mbit is 25% shorter than a folded Mbit (three versus four features), it also has half as many wordlines, effectively reducing the wordline pitch. This relaxed wordline pitch makes layout of the wordline drivers and the address decode tree much easier. In fact, both odd and even wordlines can be driven from the same row decoder block, thus eliminating half of the row decoder strips in a given array block. This is an important distinction, as the tight wordline pitch for folded digitline designs necessitates separate odd and even row decode strips.

3.2.3 Bilevel Digitline Array Architecture

The bilevel digitline array architecture depicted in Figure 3.12 is a potential architecture for tomorrow's large-scale DRAM designs. The bilevel architecture is constructed with multiple array cores separated by strips of sense amplifiers and either row decode blocks or wordline stitching regions. Wordline stitching requires a four-metal process, while row decode

blocks can be implemented in a three-metal process. The array cores include 262,144 (2^{25}) functionally addressable plaid $6F^2$ Mbits arranged in 532 rows and 524 bilevel digitline pairs. The 532 rows comprise 512 actual wordlines, 4 redundant wordlines, and 16 dummy wordlines. There are also 267 isolation gates in each array due to the use of plaid Mbits. Because they are accounted for in the wordline pitch, however, they can be ignored. The 524 bilevel digitline pairs comprise 512 actual digitline pairs, 8 redundant digitline pairs, and 4 dummy digitline pairs. The term *digitline pair* describes the array core structure because pairing is a natural product of the bilevel architecture. Each digitline pair consists of one digitline on Metal1 and a vertically aligned complementary digitline on Metal2.

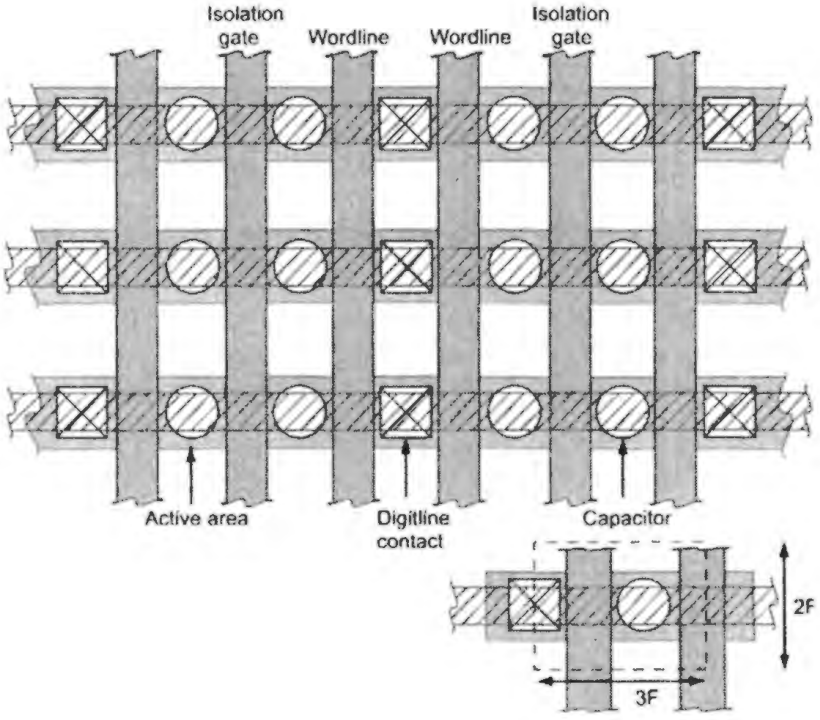


Figure 3.11 Plaid $6F^2$ Mbit array.
(The isolation gates are tied to ground or a negative voltage.

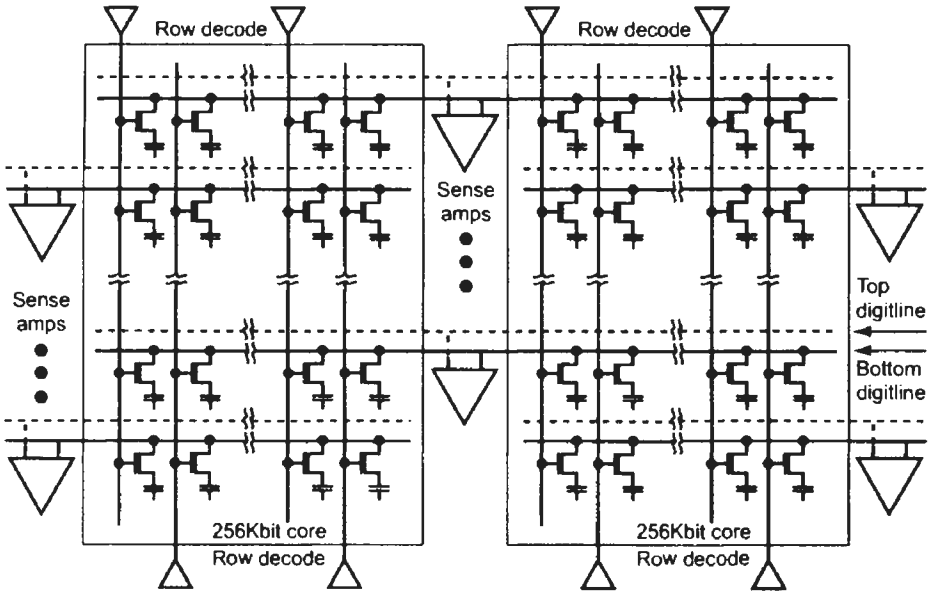


Figure 3.12 Bilevel digitline array schematic.

Sense amplifier blocks are placed on both sides of each array core. The sense amplifiers within each block are laid out at half pitch: one sense amplifier for every two Metal1 digitlines. Each sense amplifier connects through isolation devices to columns (digitline pairs) from two adjacent array cores. Similar to the folded digitline architecture, odd columns connect on one side of the array core, and even columns connect on the other side. Each sense amplifier block is then exclusively connected to either odd or even columns, never to both.

Unlike a folded digitline architecture that uses a local row decode block connected to both sides of an array core, the bilevel digitline architecture uses a local row decode block connected to only one side of each core. As stated earlier, both odd and even rows can be driven from the same local row decoder block with the relaxed wordline pitch. Because of this feature, the bilevel digitline architecture is more efficient than alternative architectures. A four-metal DRAM process allows local row decodes to be replaced by either stitch regions or local wordline drivers. Either approach could substantially reduce die size. The array core also includes the three twist regions necessary for the bilevel digitline architecture. The twist region is larger than that used in the folded digitline architecture, owing to the complexity of twisting digitlines vertically. The twist regions again constitute a break in the array structure, making it necessary to include dummy wordlines.

As with the open digitline and folded digitline architectures, the bilevel digitline length is limited by power dissipation and a minimum cell-to-digit-

line capacitance ratio. In the 256-Mbit generation, the digitlines are again restricted from connecting to more than 256 Mbits (128 Mbit pairs). The analysis to arrive at this quantity is the same as that for the open digitline architecture, except that the overall digitline capacitance is higher. The bilevel digitline runs over twice as many cells as the open digitline with the digitline running in equal lengths in both Metal2 and Metal1. The capacitance added by the Metal2 component is small compared to the already present Metal1 component because Metal2 does not connect to Mbit transistors. Overall, the digitline capacitance increases by about 25% compared to an open digitline. The power dissipated during a Read or Refresh operation is proportional to the *digitline capacitance* (C_D), the supply (internal) voltage (V_{CC}), the external voltage (V_{CCX}), the number of active columns (N), and the Refresh period (P). It is given as

$$P_D = \frac{V_{CCX} \cdot (N \cdot V_{CC} \cdot (C_D + C_C))}{2 \cdot P} \tag{3.12}$$

On a 256-Mbit DRAM in 8k Refresh, there are 32,768 (2^{15}) active columns during each Read, Write, or Refresh operation. Active array current and power dissipation for a 256-Mbit DRAM are given in Table 3.11 for a 90 ns Refresh period (–5 timing) at various digitline lengths. The budget for active array current is limited to 200mA for this 256-Mbit design. To meet this budget, the digitline cannot exceed a length of 256 Mbits.

Wordline length is again limited by the maximum allowable (RC) time constant of the wordline. The calculation for bilevel digitline is identical to that performed for open digitline due to the similarity of array core design. These results are given in Table 3.3. Accordingly, if the wordline time constant is to remain under the required 4-nanosecond limit, the wordline length cannot exceed 512 Mbits (512 bilevel digitline pairs).

Table 3.11 Active current and power versus bilevel digitline length.

Digitline Length (Mbits)	Digitline Capacitance (fF)	Active Current	Power Dissipation (mW)
128	128	75	249
256	256	151	498
512	513	301	994

A layout of various bilevel elements was generated to obtain reasonable estimates of pitch cell size. With these size estimates, overall dimen-

sions for a 32-Mbit array block could be calculated. The diagram for a 32-Mbit array block using the bilevel digitline architecture is shown in Figure 3.13. This block requires a total of one hundred twenty-eight 256kbit array cores. The 128 array cores are arranged in 16 rows and 8 columns. Each 4-Mbit vertical section consists of 512 wordlines and 8,192 bilevel digitline pairs (8,192 columns). Eight 4-Mbit strips are required to form the complete 32-Mbit block. Sense amplifier blocks are positioned vertically between each 4-Mbit section.

Row decode strips are positioned horizontally between every array core. Only eight row decode strips are needed for the sixteen array cores, for each row decode contains wordline drivers for both odd and even rows. The 32-Mbit array block shown in Figure 3.13 includes pitch cell layout estimates. Overall size for the 32-Mbit block is found by summing the dimensions for each component.

As before,

$$Height32 = (T_R \cdot H_{RDEC}) + (T_{DL} \cdot P_{DL})\ microns \tag{3.13}$$

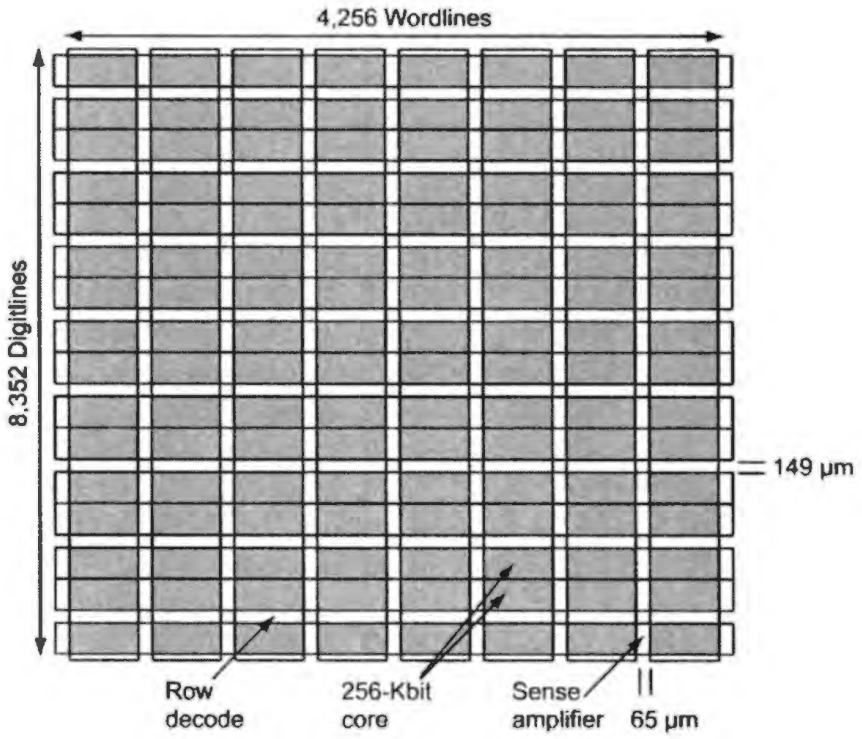


Figure 3.13 Bilevel digitline architecture 32-Mbit array block.

where T_R is the number of bilevel row decoders, H_{RDEC} is the height of each decoder, T_{DL} is the number of bilevel digitline pairs including redundant and dummy, and P_{DL} is the digitline pitch. Also,

$$Height32 = (T_{SA} \cdot W_{AMP}) + (T_{WL} \cdot P_{WL6}) + (T_{TWIST} \cdot W_{TWIST})\ microns \tag{3.14}$$

where T_{SA} is the number of sense amplifier strips, W_{AMP} is the width of the sense amplifiers, T_{WL} is the total number of wordlines including redundant and dummy, P_{WL6} is the wordline pitch for the plaid $6F^2$ Mbit, T_{TWIST} is the total number of twist regions, and W_{TWIST} is the width of the twist regions. Table 3.12 shows the calculated results for the bilevel 32-Mbit block shown in Figure 3.13. A three-metal process is assumed in these calculations because it requires the local row decoders. Array efficiency for the bilevel digitline 32-Mbit array block, which yields 63.1% for this design example, is given as

$$Efficiency = \frac{(100 \cdot 2^{25} \cdot P_{DL} \cdot 2 \cdot P_{WL6})}{Area32}\ percent \tag{3.15}$$

With Metal4 added to the bilevel DRAM process, the local row decoder scheme can be replaced by a global or hierarchical row decoder scheme. The addition of a fourth metal to the DRAM process places even greater demands on process engineers. Regardless, an analysis of 32-Mbit array block size was performed assuming the availability of Metal4. The results of the analysis are shown in Tables 3.13 and 3.14 for the global and hierarchical row decode schemes. Array efficiency for the 32-Mbit memory block using global and hierarchical row decoding is 74.5% and 72.5%, respectively.

3.2.4 Architectural Comparison

Although a straight comparison of DRAM architectures might appear simple, in fact it is very complicated. Profit remains the critical test of architectural efficiency and is the true basis for comparison. This in turn requires accurate yield and cost estimates for each alternative. Without these estimates and a thorough understanding of process capabilities, conclusions are elusive and the exercise is academic. The data necessary to perform the analysis and render a decision also varies from manufacturer to manufacturer. Accordingly, a conclusive comparison of the various array architectures is beyond the scope of this text. Rather, the architectures are compared in light of the available data. To better facilitate a comparison, the 32-Mbit

array block size data from Sections 3.1 and 3.2 is summarized in Table 3.15 for the open digitline, folded digitline, and bilevel digitline architectures.

Table 3.12 Bilevel digitline (local row decode): 32-Mbit size calculations.

Description	Parameter	Size
Number of sense amplifier strips	T_{SA}	9
Width of sense amplifiers	W_{AMP}	65 μm
Number of local decode strips	T_{LDEC}	8
Height of local decode strips	H_{LDEC}	149 μm
Number of digitlines	T_{DL}	8,352
Number of wordlines	T_{WL}	4,256
Number of twist regions	W_{TWIST1}	9 μm
Width of twist regions	W_{TWIST}	9 μm
Height of 32-Mbit block	*Height*32	6,203 μm
Width of 32-Mbit block	*Width*32	4,632 μm
Area of 32-Mbit block	*Area*32	28,732,296 μm^2

From Table 3.15, it seems that overall die size (32-Mbit area) is a better metric for comparison than array efficiency. For instance, the three-metal folded digitline design using hierarchical row decodes has an area of 34,089,440mm^2 and an efficiency of 70.9%. The three-metal bilevel digitline design with local row decodes has an efficiency of only 63.1% but an overall area of 28,732,296 mm^2. Array efficiency for the folded digitline is higher. This is misleading, however, because the folded digitline yields a die that is 18.6% larger for the same number of conductors.

Table 3.13 Bilevel digitline (global decode): 32-Mbit size calculations.

Description	Parameter	Size
Number of sense amplifier strips	T_{SA}	9
Width of sense amplifiers	W_{AMP}	65 μm
Number of global decode strips	T_{GDEC}	1
Height of global decode strips	H_{GDEC}	200 μm
Number of stitch regions	N_{ST}	4
Height of stitch regions	H_{ST}	10 μm
Number of digitlines	T_{DL}	8,352
Number of wordlines	T_{WL}	4,256
Number of twist regions	T_{TWIST}	24
Width of twist regions	W_{TWIST}	9 μm
Height of 32-Mbit block	*Height*32	5,251 μm
Width of 32-Mbit block	*Width*32	4,632 μm
Area of 32-Mbit block	*Area*32	24,322,632 μm^2

Table 3.15 also illustrates that the bilevel digitline architecture always yields the smallest die area, regardless of the configuration. The smallest folded digitline design at 32,654,160mm^2 and the smallest open digitline design at 29,944,350mm^2 are still larger than the largest bilevel digitline design at 28,732,296mm^2. It is also apparent that both the bilevel and open digitline architectures need at least three conductors in their construction.

The folded digitline architecture still has a viable design option using only two conductors. The penalty of using two conductors is a much larger die size—a full 41% than the three-metal bilevel digitline design.

Table 3.14 t Bilevel digitline (hierarchical row decode): 32-Mbit size calculations.

Description	Parameter	Size
Number of sense amplifier strips	T_{SA}	9
Width of sense amplifiers	W_{AMP}	65 µm
Number of global decode strips	T_{GDEC}	1
Height of global decode strips	H_{GDEC}	190 µm
Number of hier decode strips	N_{HDEC}	4
Height of hier decode strips	H_{HDEC}	48 µm
Number of digitlines	T_{DL}	8,352
Number of wordlines	T_{WL}	4,256
Number of twist regions	T_{TWIST}	24
Width of twist regions	W_{TWIST}	9 µm
Height of 32-Mbit block	*Height*32	5,393 µm
Width of 32-Mbit block	*Width*32	4,632 µm
Area of 32-Mbit block	*Area*32	24,930,376 µm^2

Table 3.15 32-Mbit size calculations summary.

Architecture	Row Decode	Metals	32-Mbit Area (μm^2)	Efficiency (%)
Open digit	Global	3	29,944,350	60.5
Open digit	Hier	3	32,429,565	55.9
Folded digit	Local	2	40,606,720	59.5
Folded digit	Global	3	32,654,160	74.0
Folded digit	Hier	3	34,089,440	70.9
Bilevel digit	Local	3	28,732,296	63.1
Bilevel digit	Global	4	24,322,632	74.5
Bilevel digit	Hier	4	24,980,376	72.5

REFERENCES

[1] H. Hidaka, Y. Matsuda, and K. Fujishima, "A divided/shared bit-line sensing scheme for ULSI DRAM cores," *IEEE Journal of Solid-State Circuits,* vol. 26, pp. 473–477, April 1991.

[2] T. Hamada, N. Tanabe, H. Watanabe, K. Takeuchi, N. Kasai, H. Hada, K. Shibahara, K. Tokashiki, K. Nakajima, S. Hirasawa, E. Ikawa, T. Saeki, E. Kakehashi, S. Ohya, and T. A. Kunio, "A split-level diagonal bit-line (SLDB) stacked capacitor cell for 256Mb DRAMs," in *1992 IEDM Technical Digest,* pp. 799–802.

[3] M. Inoue, H. Kotani, T. Yamada, H. Yamauchi, A. Fujiwara, J. Matsushima, H. Akamatsu, M. Fukumoto, M. Kubota, I. Nakao, N. Aoi, G. Fuse, S. Ogawa, S. Odanaka, A. Ueno, and H. Yamamoto, "A 16Mb DRAM with an open bit-line architecture," in *1988 IEEE ISSCC Digest of Technical Papers,* pp. 246–247.

[4] M. Inoue, T. Yamada, H. Kotani, H. Yamauchi, A. Fujiwara, J. Matsushima, H. Akamatsu, M. Fukumoto, M. Kubota, I. Nakao, N. Aoi, G. Fuse, S. Ogawa, S. Odanaka, A. Ueno, and H. Yamamoto, "A 16-Mbit DRAM with a relaxed sense-amplifier-pitch open-bit-line architecture," *IEEE Journal of Solid-State Circuits,* vol. 23, pp. 1104–1112, October 1988.

[5] K. Noda, T. Saeki, A. Tsujimoto, T. Murotani, and K. Koyama, "A boosted dual word-line decoding scheme for 256Mb DRAMs," in *1992 Symposium on VLSI Circuits Digest of Technical Papers*, pp. 112–113.

[6] D. Takashima, S. Watanabe, H. Nakano, Y. Oowaki, and K. Ohuchi, "Open/folded bit-line arrangement for ultra-high-density DRAMs," *IEEE Journal of Solid-State Circuits,* vol. 29, pp. 539–542, April 1994.

[7] Y. Oowaki, K. Tsuchida, Y. Watanabe, D. Takashima, M. Ohta, H. Nakano, S. Watanabe, A. Nitayama, F. Horiguchi, K. Ohuchi, and F. Masuoka, "A 33-ns 64Mb DRAM," *IEEE Journal of Solid-State Circuits,* vol. 26, pp. 1498–1505, November 1991.

[8] Y. Nakagome, M. Aoki, S. Ikenaga, M. Horiguchi, S. Kimura, Y. Kawamoto, and K. Itoh, "The impact of data-line interference noise on DRAM scaling," *IEEE Journal of Solid-State Circuits,* vol. 23, pp. 1120–1127, October 1988.

[9] T. Yoshihara, H. Hidaka, Y. Matsuda, and K. Fujishima, "A twisted bitline technique for multi-Mb DRAMs," in *1988 IEEE ISSCC Digest of Technical Papers,* pp. 238–239.

Chapter
4

The Peripheral Circuitry

In this chapter, we briefly discuss the peripheral circuitry. In particular, we discuss the column decoder and its implementation. We also cover the implementation of row and column redundancy.

4.1 COLUMN DECODER ELEMENTS

The column decoder circuits represent the final DRAM elements that pitch up to the array mbits. Historically, column decode circuits were simple and straightforward: static logic gates were generally used for both the decode tree elements and the driver output. Static logic was used primarily because of the nature of column addressing in *fast page mode* (FPM) and *extended data out* (EDO) devices. Unlike row addressing, which occurred once per *RAS** cycle, column addressing could occur multiple times per *RAS** cycle, with each column held open until a subsequent column address appeared. The transition interval from one column to the next had to be minimized, allowing just enough time to turn OFF the previous column, turn ON the new column, and equilibrate the necessary data path circuits. Furthermore, because column address transitions were unpredictable and asynchronous, a column decode logic that was somewhat forgiving of random address changes was required—hence the use of static logic gates.

With the advent of *synchronous DRAMs* (SDRAMs) and high-speed, packet-based DRAM technology, the application of column, or row, addresses became synchronized to a clock. More importantly, column addressing and column timing became more predictable, allowing design engineers to use pipelining techniques along with dynamic logic gates in constructing column decode elements.

Column redundancy adds complexity to the column decoder because the redundancy operation in FPM and EDO DRAMs requires the decode

circuit to terminate column transitions, prior to completion. In this way, redundant column elements can replace normal column elements. Generally, the addressed column select is allowed to fire normally. If a redundant column match occurs for this address, the normal column select is subsequently turned OFF; the redundant column select is fired. The redundant match is timed to disable the addressed column select before enabling the I/O devices in the sense amplifier.

The fire-and-cancel operation used on the FPM and EDO column decoders is best achieved with static logic gates. In packet-based and synchronous DRAMs, column select firing can be synchronized to the clock. Synchronous operation, however, does not favor a fire-and-cancel mode, preferring instead that the redundant match be determined prior to firing either the addressed or redundant column select. This match is easily achieved in a pipeline architecture because the redundancy match analysis can be performed upstream in the address pipeline before presenting the address to the column decode logic.

A typical FPM- or EDO-type column decoder realized with static logic gates is shown schematically in Figure 4.1. The address tree is composed of combinations of NAND or NOR gates. In this figure, the address signals are active HIGH, so the tree begins with two-input NAND gates. Using predecoded address lines is again preferred. Predecoded address lines both simplify and reduce the decoder logic because a single input can represent two or more address terms. In the circuit shown in Figure 4.1, the four input signals *CA*23, *CA*45, *CA*67, and *CA*8 represent seven address terms, permitting 1 of 128 decoding. Timing of the column selection is controlled by a signal called *column decode enable (CDE)*, which is usually combined with an input signal, as shown in Figure 4.2, or as an additional term in the tree.

The final signal shown in the figure is labeled *RED*. This signal disables the normal column term and enables redundant column decoders whenever the column address matches any of the redundant address locations, as determined by the column-redundant circuitry. A normal column select begins to turn ON before *RED* makes a LOW-to-HIGH transition. This fire-and-cancel operation maximizes the DRAM speed between column accesses, making column timing all the more critical. An example, which depicts a page mode column transition, is shown in Figure 4.2. During the fire-and-cancel transition, *I/O equilibration (EQIO)* envelops the deselection of the old column select *CSEL*<0> and the selection of a new column select *RCSEL*<1>. In this example, *CSEL*<1> initially begins to turn ON until *RED* transitions HIGH, disabling *CSEL*<1> and enabling redundant column *RCSEL*<1>.

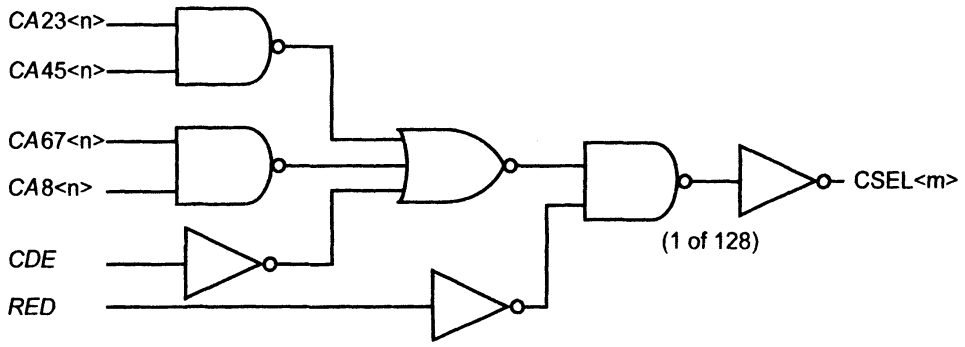


Figure 4.1 Column decode.

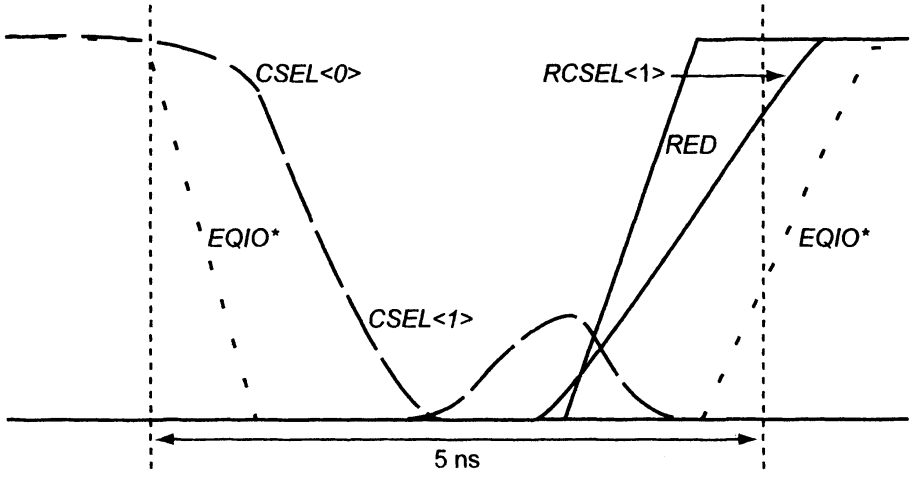


Figure 4.2 Column selection timing.

The column decode output driver is a simple CMOS inverter because the *column select signal (CSEL)* only needs to drive to V_{CC}. On the other hand, the wordline driver, as we have seen, is rather complex; it needed to drive to a boosted voltage, V_{CCP}. Another feature of column decoders is that their pitch is very relaxed relative to the pitch of the sense amplifiers and row decoders. From our discussion in Section 2.2 concerning *I/O* transistors and *CSEL* lines, we learned that a given *CSEL* is shared by four to eight *I/O* transistors. Therefore, the *CSEL* pitch is one *CSEL* for every eight to sixteen digitlines. As a result, the column decoder layout is much less difficult to implement than either the row decoder or the sense amplifiers.

A second type of column decoder, realized with dynamic P&E logic, is shown in Figure 4.3. This particular design was first implemented in an 800MB/s packet-based SLDRAM device. The packet nature of SLDRAM and the extensive use of pipelining supported a column decoder built with P&E logic. The column address pipeline included redundancy match circuits upstream from the actual column decoder, so that both the column

address and the corresponding match data could be presented at the same time. There was no need for the fire-and-cancel operation: the match data was already available.

Therefore, the column decoder fires either the addressed column select or the redundant column select in synchrony with the clock. The decode tree is similar to that used for the CMOS wordline driver; a pass transistor was added so that a decoder enable term could be included. This term allows the tree to disconnect from the latching column select driver while new address terms flow into the decoder. A latching driver was used in this pipeline implementation because it held the previously addressed column select active with the decode tree disconnected. Essentially, the tree would disconnect after a column select was fired, and the new address would flow into the tree in anticipation of the next column select. Concurrently, redundant match information would flow into the phase term driver along with *CA*45 address terms to select the correct phase signal. A redundant match would then override the normal phase term and enable a redundant phase term.

Operation of this column decoder is shown in Figure 4.4. Once again, deselection of the old column select *CSEL*<0> and selection of a new column select *RCSEL*<1> are enveloped by *EQIO*. Column transition timing is under the control of the column latch signal *CLATCH**. This signal shuts OFF the old column select and enables firing of the new column select. Concurrent with *CLATCH** firing, the decoder is enabled with *decoder enable (DECEN)* to reconnect the decode tree to the column select driver. After the new column select fires, *DECEN* transitions LOW to once again isolate the decode tree.

4.2 COLUMN AND ROW REDUNDANCY

Redundancy has been used in DRAM designs since the 256k generation to improve yield and profitability. In redundancy, spare elements such as rows and columns are used as logical substitutes for defective elements. The substitution is controlled by a physical encoding scheme. As memory density and size increase, redundancy continues to gain importance. The early designs might have used just one form of repairable elements, relying exclusively on row or column redundancy. Yet as processing complexity increased and feature size shrank, both types of redundancy—row and column—became mandatory.

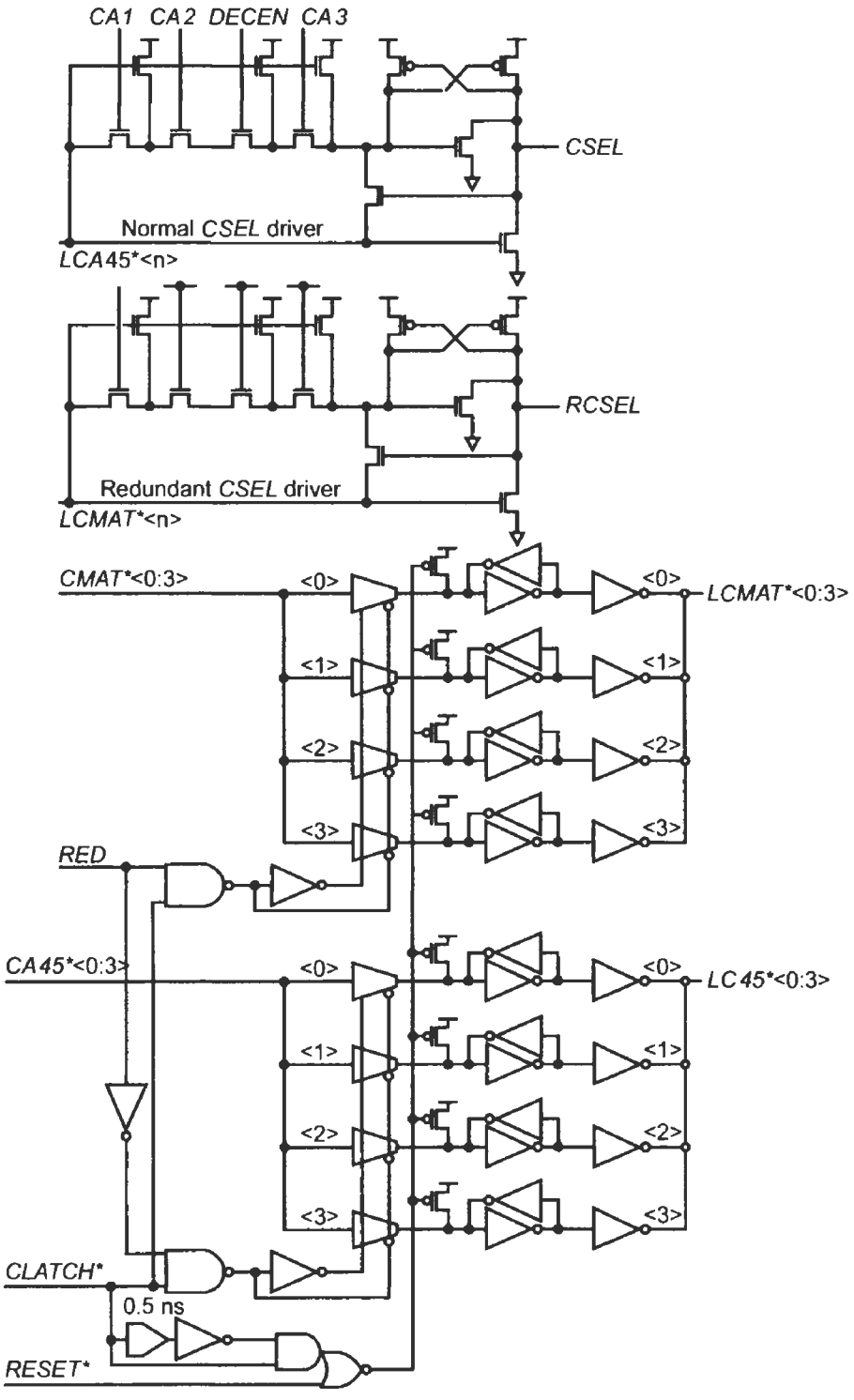


Figure 4.3 Column decode: P&E logic.

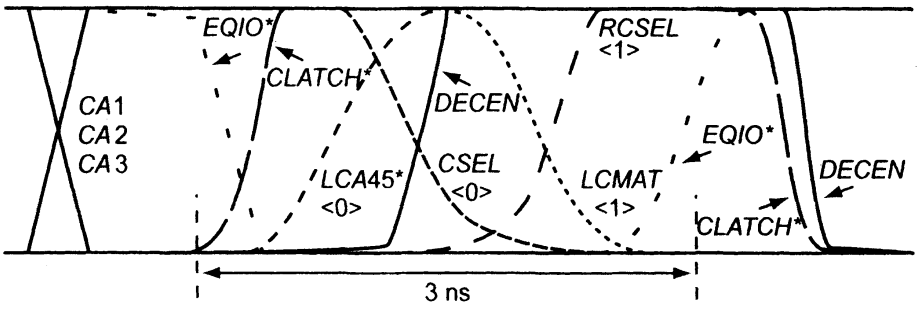


Figure 4.4 Column decode waveforms.

Today various DRAM manufacturers are experimenting with additional forms of repair, including replacing entire subarrays. The most advanced type of repair, however, involves using movable saw lines as realized on a prototype 1Gb design [1]. Essentially, any four good adjacent quadrants from otherwise bad die locations can be combined into a good die by simply sawing the die along different scribe lines. Although this idea is far from reaching production, it illustrates the growing importance of redundancy.

4.2.1 Row Redundancy

The concept of row redundancy involves replacing bad wordlines with good wordlines. There could be any number of problems on the row to be repaired, including shorted or open wordlines, wordline-to-digitline shorts, or bad Mbit transistors and storage capacitors. The row is not physically but logically replaced. In essence, whenever a row address is strobed into a DRAM by *RAS**, the address is compared to the addresses of known bad rows. If the address comparison produces a match, then a replacement wordline is fired in place of the normal (bad) wordline.

The replacement wordline can reside anywhere on the DRAM. Its location is not restricted to the array containing the normal wordline, although its range may be restricted by architectural considerations. In general, the redundancy is considered local if the redundant wordline and the normal wordline must always be in the same subarray.

If, however, the redundant wordline can exist in a subarray that does not contain the normal wordline, the redundancy is considered global. Global repair generally results in higher yield because the number of rows that can be repaired in a single subarray is not limited to the number of its redundant rows. Rather, global repair is limited only by the number of fuse banks, termed *repair elements,* that are available to any subarray.

Local row repair was prevalent through the 16-Meg generation, producing adequate yield for minimal cost. Global row repair schemes are becom-

ing more common for 64-Meg or greater generations throughout the industry. Global repair is especially effective for repairing clustered failures and offers superior repair solutions on large DRAMs.

Dynamic logic is a traditional favorite among DRAM designers for row redundant match circuits. Dynamic gates are generally much faster than static gates and well suited to row redundancy because they are used only once in an entire *RAS** cycle operation. The dynamic logic we are referring to again is called *precharge and evaluate* (P&E). Match circuits can take many forms; a typical row match circuit is shown in Figure 4.5. It consists of a *PRECHARGE* transistor M1, match transistors M2–M5, laser fuses F1–F4, and static gate I1. In addition, the node labeled *row PRECHARGE (RPRE*)* is driven by static logic gates. The fuses generally consist of narrow polysilicon lines that can be blown or opened with either a precision laser or a fusing current provided by additional circuits (not shown).

For our example using predecoded addresses, three of the four fuses shown must be blown in order to program a match address. If, for instance, F2–F4 were blown, the circuit would match for *RA*12<0> but not for *RA*12<1:3>. Prior to *RAS** falling, the node labeled *RED** is precharged to V_{CC} by the signal *RPRE**, which is LOW. Assuming that the circuit is enabled by fuse F5, *EVALUATE** will be LOW. After *RAS** falls, the row addresses eventually propagate into the redundant block. If *RA*12<0> fires HIGH, *RED** discharges through M2 to ground. If, however, *RA*12<0> does not fire HIGH, *RED** remains at V_{CC}, indicating that a match did not occur. A weak latch composed of I1 and M6 ensures that *RED** remains at V_{CC} and does not discharge due to junction leakage.

This latch can easily be overcome by any of the match transistors. The signal *RED** is combined with static logic gates that have similar *RED** signals derived from the remaining predecoded addresses. If all of the *RED** signals for a redundant element go LOW, then a match has occurred, as indicated by *row match (RMAT*)* firing LOW. The signal *RMAT** stops the normal row from firing and selects the appropriate replacement wordline. The fuse block in Figure 4.5 shows additional fuses F5–F6 for enabling and disabling the fuse bank. Disable fuses are important in the event that the redundant element fails and the redundant wordline must itself be repaired.

The capability to pretest redundant wordlines is an important element in most DRAM designs today. For the schematic shown in Figure 4.5, the pretest is accomplished through the set of static gates driven by the *redundant test (REDTEST)* signal. The input signals labeled *TRA*12n, *TRA*34n, *TRA*56n, and *TODDEVEN* are programmed uniquely for each redundant element through connections to the appropriate predecoded address line.

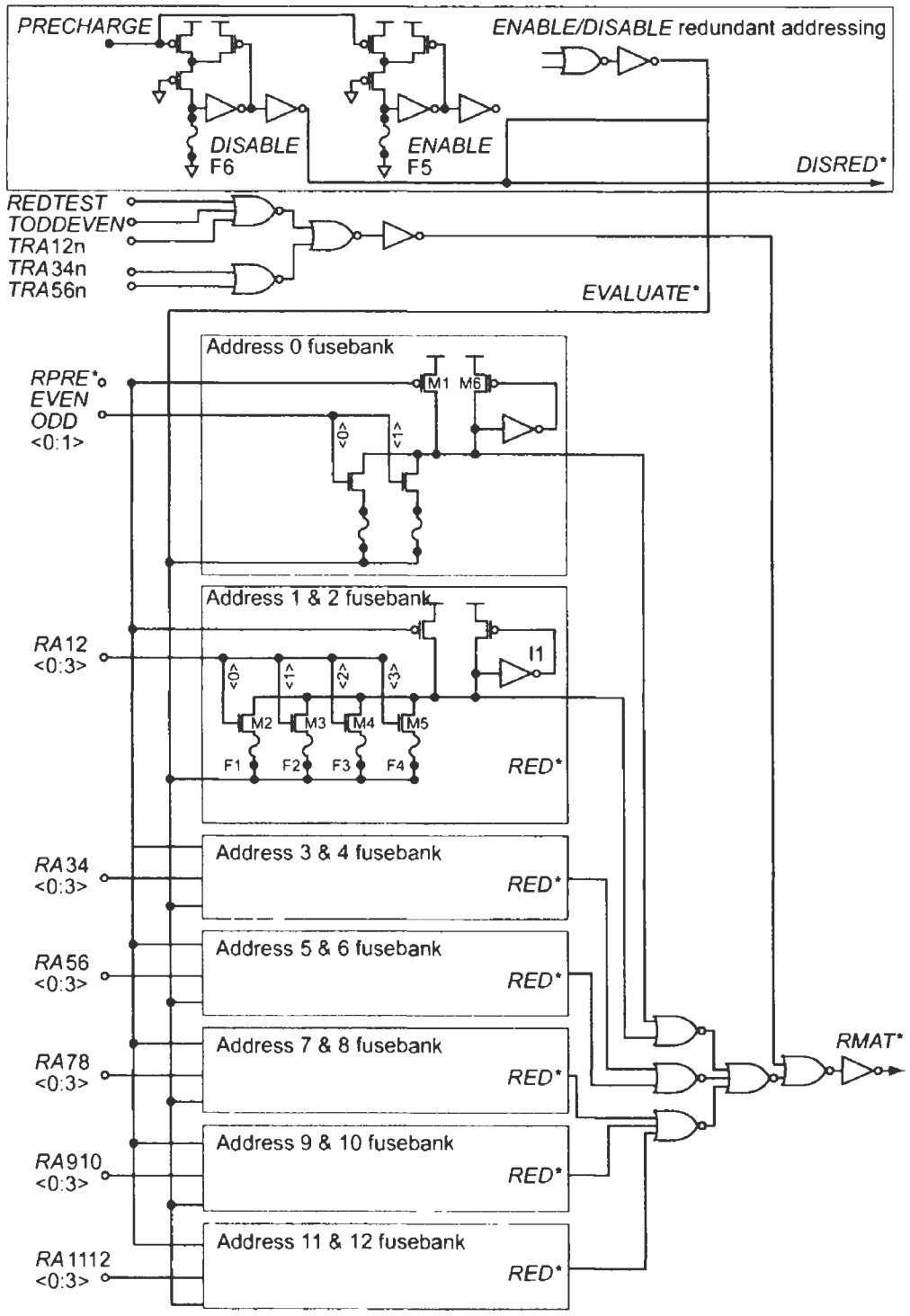


Figure 4.5 Row fuse block.

REDTEST is HIGH whenever the DRAM is in the redundant row pretest mode. If the current row address corresponds to the programmed pretest address, *RMAT** will be forced LOW, and the corresponding redundant

wordline rather than the normal wordline will be fired. This pretest capability permits all of the redundant wordlines to be tested prior to any laser programming.

Fuse banks or redundant elements, as shown in Figure 4.5, are physically associated with specific redundant wordlines in the array. Each element can fire only one specific wordline, although generally in multiple subarrays. The number of subarrays that each element controls depends on the DRAM's architecture, refresh rate, and redundancy scheme. It is not uncommon in 16-Meg DRAMs for a redundant row to replace physical rows in eight separate subarrays at the same time. Obviously, the match circuits must be fast. Generally, firing of the normal row must be held off until the match circuits have enough time to evaluate the new row address. As a result, time wasted during this phase shows up directly on the part's *row access (tRAC)* specification.

4.2.2 Column Redundancy

Column redundancy is the second type of repair available on most DRAM designs. In Section 4.1, it was stated that column accesses can occur multiple times per *RAS** cycle. Each column is held open until a subsequent column appears. Therefore, column redundancy was generally implemented with circuits that are very different from those seen in row redundancy. As shown in Figure 4.6, a typical column fuse block is built from static logic gates rather than from P&E dynamic gates. P&E logic, though extremely fast, needs to be precharged prior to each evaluation. The transition from one column to another must occur within the span of a few nanoseconds. On FPM and EDO devices, which had unpredictable and asynchronous column address transitions, there was no guarantee that adequate time existed for this *PRECHARGE* to occur. Yet the predictable nature of column addressing on the newer type of synchronous and packet-based DRAMs affords today's designers an opportunity to use P&E-type column redundancy circuits. In these cases, the column redundant circuits appear very similar to P&E-style row redundant circuits.

An example of a column fuse block for an FPM or EDO design is shown in Figure 4.6. This column fuse block has four sets of column fuse circuits and additional enable/disable logic. Each column fuse circuit, corresponding to a set of predecoded column addresses, contains compare logic and two sets of laser fuse/latch circuits. The laser fuse/latch reads the laser fuse whenever *column fuse power (CFP)* is enabled, generally on POWERUP and during *RAS** cycles. The fuse values are held by the simple inverter latch circuits composed of I0 and I1. Both true and complement data are fed from the fuse/latch circuit into the comparator logic. The com-

parator logic, which appears somewhat complex, is actually quite simple as shown in the following Boolean expression where $F0$ without the bar indicates a blown fuse:

$$CAM = (\overline{A0} \bullet \overline{F0} \bullet \overline{F1}) + (\overline{A1} \bullet F0 \bullet \overline{F1}) + (\overline{A2} \bullet \overline{F0} \bullet F1) + (\overline{A3} \bullet F0 \bullet F1) \quad \textbf{(4.1)}$$

The *column address match (CAM)* signals from all of the predecoded addresses are combined with static logic gates to create a *column match (CMAT*)* signal for the column fuse block. The *CMAT** signal, when active, cancels normal *CSEL* signals and enables redundant *RCSEL* signals, as described in Section 4.1. Each column fuse block is active only when its corresponding enable fuse is blown. The column fuse block usually contains a disable fuse for the same reason as a row redundant block: to repair a redundant element. Column redundant pretest is implemented somewhat differently in Figure 4.6 than row redundant pretest here. In Figure 4.6, the bottom fuse terminal is not connected directly to ground. Rather, all of the signals for the entire column fuse block are brought out and programmed either to ground or to a column pretest signal from the test circuitry.

During standard part operation, the pretest signal is biased to ground, allowing the fuses to be read normally. However, during column redundant pretest, this signal is brought to V_{CC}, which makes the laser fuses appear to be programmed. The fuse/latch circuits latch the apparent fuse states on the next *RAS** cycle. Subsequent column accesses allow the redundant column elements to be pretested by addressing their pre-programmed match addresses.

The method of pretesting just described always uses the match circuits to select a redundant column. It is a superior method to that described for the row redundant pretest because it tests both the redundant element and its match circuit. Furthermore, as the match circuit is essentially unaltered during redundant column pretest, the test is a better measure of the obtainable DRAM performance when the redundant element is active.

Obviously, the row and column redundant circuits that are described in this section are only one embodiment of what could be considered a wealth of possibilities. It seems that all DRAM designs use some alternate form of redundancy. Other types of fuse elements could be used in place of the laser fuses that are described. A simple transistor could replace the laser fuses in either Figure 4.5 or Figure 4.6, its gate being connected to an alternative fuse element. Furthermore, circuit polarity could be reversed and non-predecoded addressing and other types of logic could be used. The options are nearly limitless. Figure 4.7 shows a SEM image of a typical set of poly fuses.

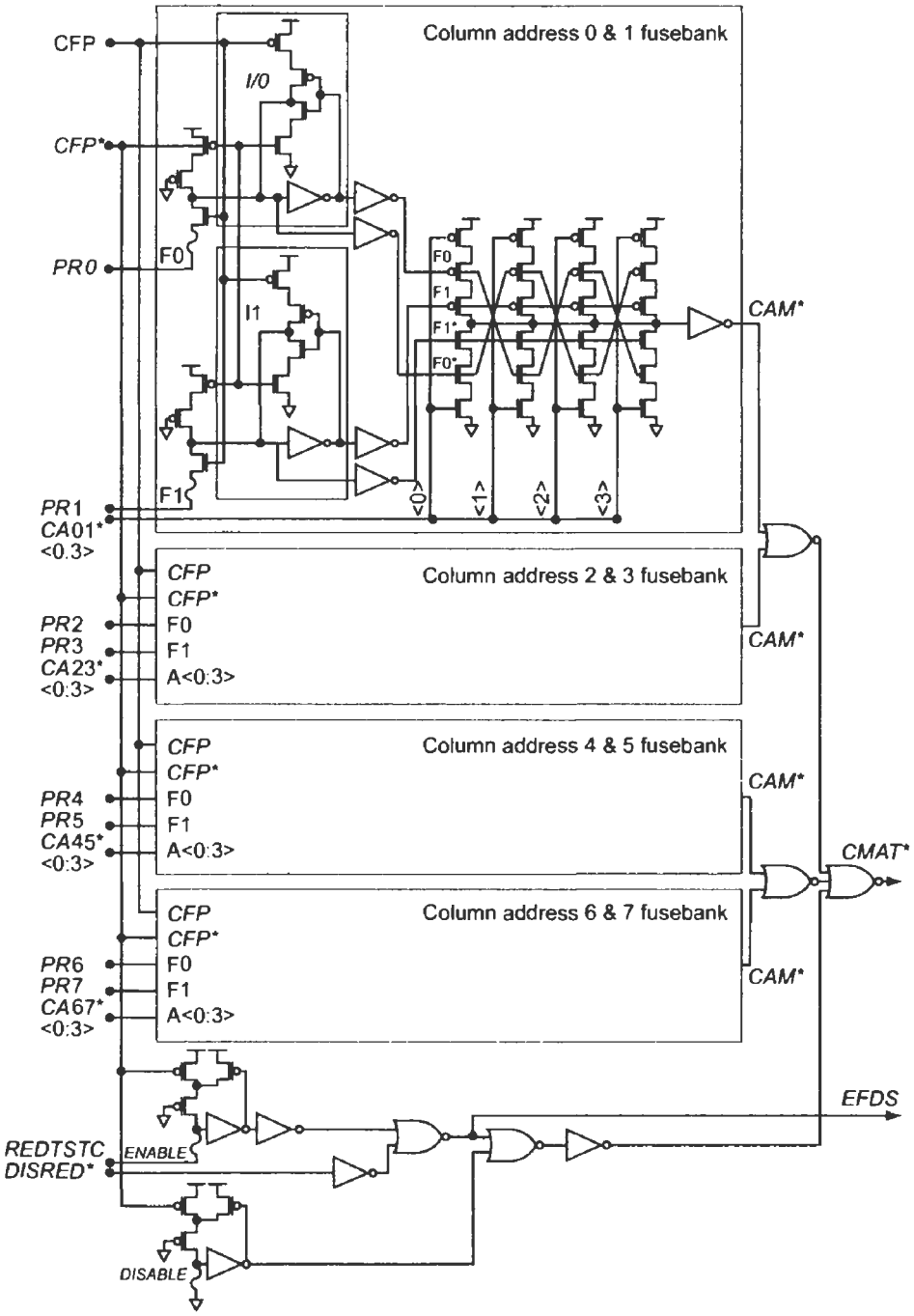


Figure 4.6 Column fuse block.



Figure 4.7 8-Meg x 8-sync DRAM poly fuses.

REFERENCE

[1] T. Sugibayashi, I. Naritake, S. Utsugi, K. Shibahara, R. Oikawa, H. Mori, S. Iwao, T. Murotani, K. Koyama, S. Fukuzawa, T. Itani, K. Kasama, T. Okuda, S. Ohya, and M. Ogawa, "A 1Gbit DRAM for file applications," in *Digest of International Solid-State Circuits Conference*, 1995, pp. 254–255.

Chapter
5

Global Circuitry and Considerations

In this chapter, we discuss the circuitry and design considerations associated with the circuitry external to the DRAM memory array and memory array peripheral circuitry. We call this *global circuitry.*

5.1 DATA PATH ELEMENTS

The typical DRAM data path is bidirectional, allowing data to be both written to and read from specific memory locations. Some of the circuits involved are truly bidirectional, passing data in for Write operations and out for Read operations. Most of the circuits, however, are unidirectional, operating on data in only a Read or Write operation. To support both operations, therefore, unidirectional circuits occur in complementary pairs: one for reading and one for writing. Operating the same regardless of the data direction, sense amplifiers, I/O devices, and data muxes are examples of bidirectional circuits. Write drivers and data amplifiers, such as *direct current sense amplifiers* (DCSAs) or data input buffers and data output buffers, are examples of paired, unidirectional circuits. In this chapter, we explain the operation and design of each of these elements and then show how the elements combine to form a DRAM data path. In addition, we discuss address test compression circuits and data test compression circuits and how they affect the overall data path design.

5.1.1 Data Input Buffer

The first element of any DRAM data path is the data input buffer. An early example is shown in Figure 5.1, the input buffer consists of both

NMOS and PMOS transistors, basically forming a pair of cascaded inverters. The first inverter stage has *ENABLE* transistors M1 and M2, allowing the buffer to be powered down during inactive periods. The transistors are carefully sized to provide high-speed operation and specific input trip points. In first- and second-generation DRAMs, the *high-input trip point* V_{IH} is set to 2.0 V for *low-voltage TTL (LVTTL)*-compatible DRAMs, while the *low-input trip point* V_{IL} is set to 0.8 V.

Designing an input buffer to meet specified input trip points generally requires a flexible design with a variety of transistors that can be added or deleted with edits to the metal mask. This is apparent in Figure 5.1 by the presence of switches in the schematic; each switch represents a particular metal option available in the design. Because of variations in temperature, device, and process, the final input buffer design is determined with actual silicon, not simulations. For a DRAM that is 8 bits wide (x8), there will be eight input buffers, each driving into one or more Write driver circuits through a signal labeled *DW*<n> (Data Write where n corresponds to the specific data bit 0–7).

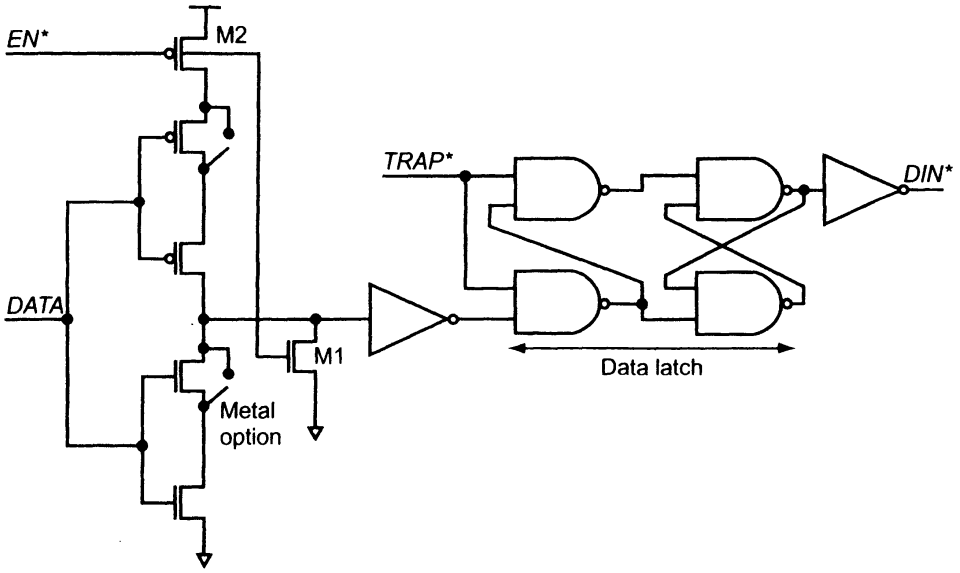


Figure 5.1 Data input buffer.

As supply voltage levels dropped with process scaling, the basic inverter-based input buffer shown in Figure 5.1 found less use in DRAM. The required noise margins and speed of the interconnecting bus between the memory controller and the DRAM were difficult to meet. Other topologies are used. One high-speed bus topology, called *stub series terminated logic* (SSTL), is shown in Figure 5.2 [1]. Tightly controlled transmission line impedances and series resistors transmit high-speed signals with little distortion. Figure 5.2a shows the bus for clocks, command signals, and addresses. Figure 5.2b shows the bidirectional bus for transmitting data to

and from the DRAM controller. In either circuit, V_{TT} and V_{REF} are set to $V_{CC}/2$.

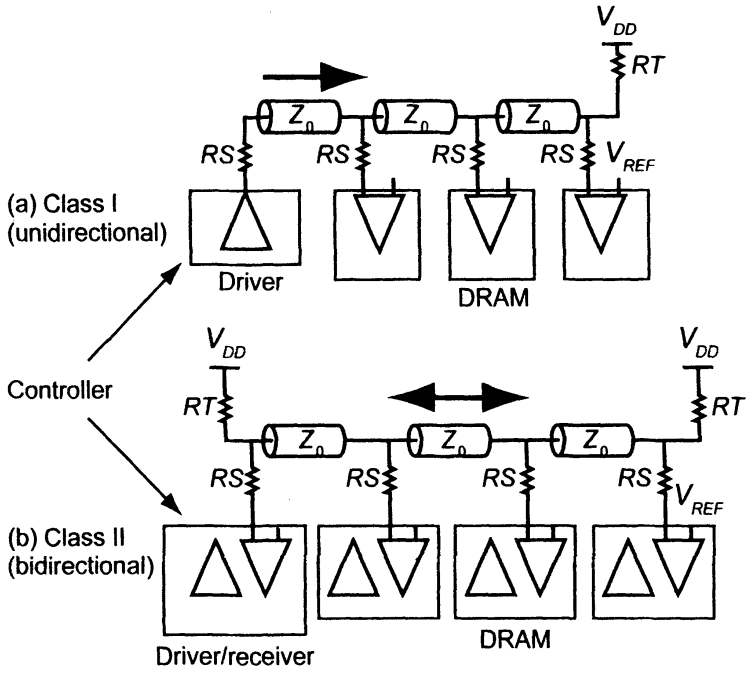


Figure 5.2 Stub series terminated logic (SSTL).

From this topology, we can see that a differential input buffer should be used: an inverter won't work. Some examples of fully differential input buffers are seen in Figure 5.3, Figure 5.4, and Figure 5.5 [1].

Figure 5.3 is simply a CMOS differential amplifier with an inverter output to generate valid CMOS logic levels. Common-mode noise on the diff-amp inputs is, ideally, rejected while amplifying the difference between the input signal and the reference signal. The diff-amp input common-mode range, say a hundred mV, sets the minimum input signal amplitude (centered around V_{REF}) required to cause the output to change stages. The speed of this configuration is limited by the diff-amp biasing current. Using a large current will increase input receiver speed and, at the same time, decrease amplifier gain and reduce the diff-amp's input common-mode range.

The input buffer of Figure 5.3 requires an external biasing circuit. The circuit of Figure 5.4 is *self-biasing*. This circuit is constructed by connecting the gate of the NMOS device to the gates of the PMOS load transistors. This circuit is simple and, because of the adjustable biasing connection, potentially very fast. The output inverter is needed to ensure that valid output logic levels are generated.

Both of the circuits in Figures 5.3 and 5.4 suffer from duty-cycle distortion at high speeds. The *PULLUP* delay doesn't match the *PULLDOWN*

delay. Duty-cycle distortion is more of a factor in input buffer design in synchronous DRAMs where clocking is performed on both the rising and falling edges of the system clock. The differential self-biased input receiver of Figure 5.5 is used to increase the allowable input signal range for wide-swing operation.

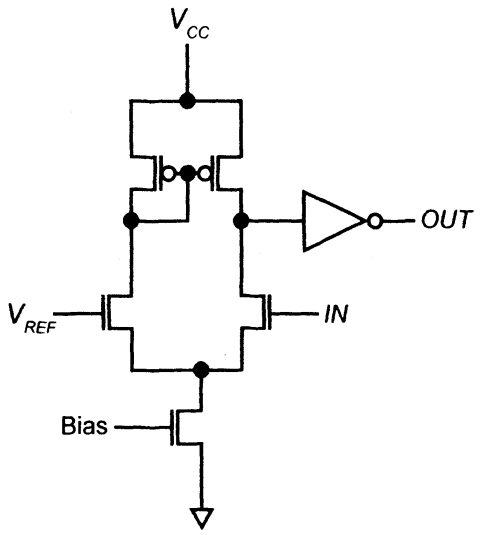


Figure 5.3 Differential amplifier-based input receiver.

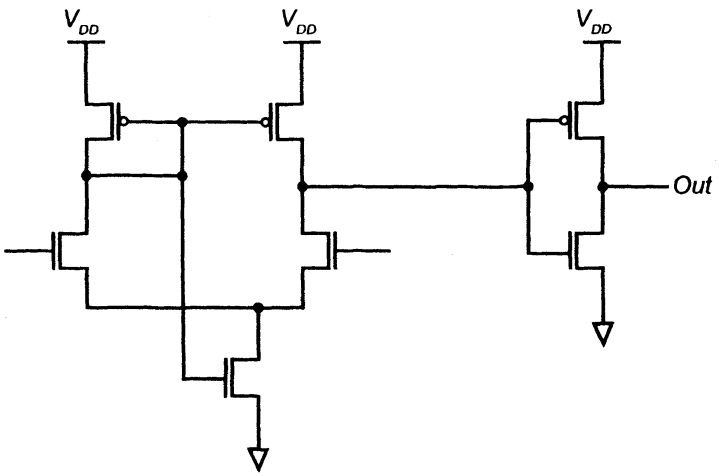


Figure 5.4 An (n-flavor) input buffer for high-speed digital design.

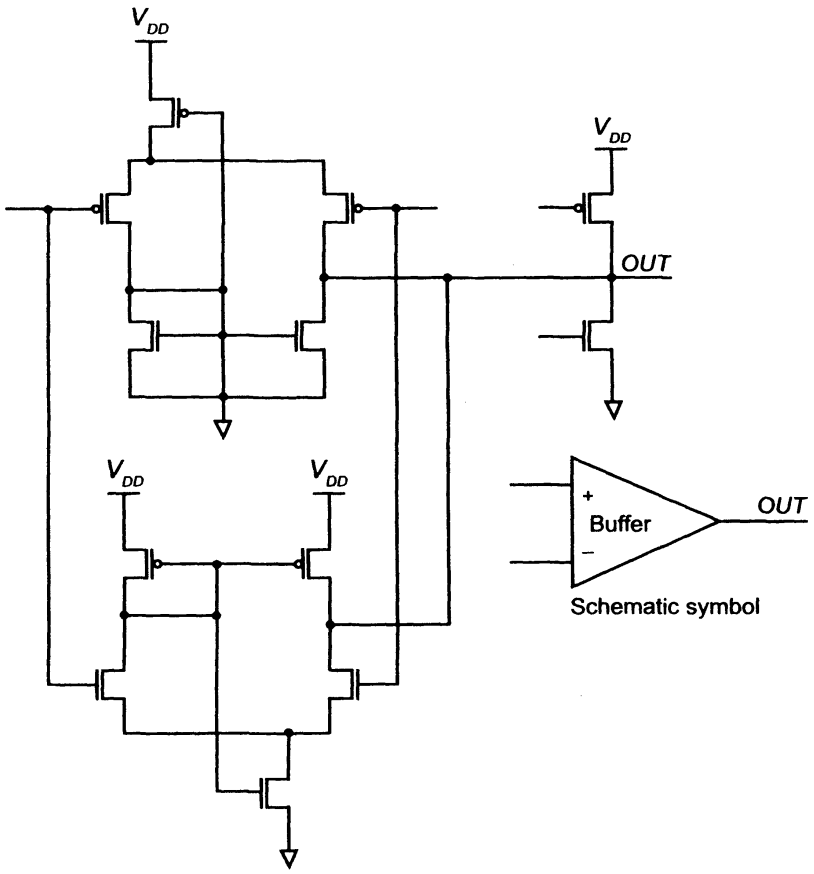


Figure 5.5 A rail-to-rail input buffer.

5.1.2 Data Write Muxes

Data muxes are often used to extend the versatility of a design. Although some DRAM designs connect the input buffer directly to the Write driver circuits, most architectures place a block of Data Write muxes between the input buffers and the Write drivers. The muxes allow a given DRAM design to support multiple configurations, such as x4, x8, and x16 *I/O*. A typical schematic for these muxes is shown in Figure 5.6. As shown in this figure, the muxes are programmed according to the bond option control signals labeled *OPTX*4, *OPTX*8, and *OPTX*16. For x16 operation, each input buffer is muxed to only one set of *DW* lines. For x8 operation, each input buffer is muxed to two sets of *DW* lines, essentially doubling the quantity of Mbits available to each input buffer. For x4 operation, each input buffer is muxed to four sets of *DW* lines, again doubling the number of Mbits available to the remaining four operable input buffers.

Essentially, as the quantity of input buffers is reduced, the amount of column address space for the remaining buffers is increased. This concept is easy to understand as it relates to a 16Mb DRAM. As a x16 part, this DRAM has 1 Mbit per data pin; as a x8 part, 2 Mbits per data pin; and as a

x4 part, 4 Mbits per data pin. For each configuration, the number of array sections available to an input buffer must change. By using Data Write muxes that permit a given input buffer to drive as few or as many Write driver circuits as required, design flexibility is easily accommodated.

5.1.3 Write Driver Circuit

The next element in the data path to be considered is the Write driver circuit. This circuit, as the name implies, writes data from the input buffers into specific memory locations. The Write driver, as shown in Figure 5.7, drives specific *I/O* lines coming from the Mbit arrays. A given Write driver is generally connected to only one set of *I/O* lines, unless multiple sets of *I/O* lines are fed by a single Write driver circuit via additional muxes. Using muxes between the Write driver and the arrays to limit the number of Write drivers and DCSA circuits is quite common. Regardless, the Write driver uses a tristate output stage to connect to the *I/O* lines. Tristate outputs are necessary because the *I/O* lines are used for both Read and Write operations. The Write driver remains in a high-impedance state unless the signal labeled Write is HIGH and either *DW* or *DW** transitions LOW from the initial HIGH state, indicating a Write operation. As shown in Figure 5.7, the Write driver is controlled by specified column addresses, the Write signal, and *DW*<n>. The driver transistors are sized large enough to ensure a quick, efficient Write operation. This is important because the array sense amplifiers usually remain ON during a Write cycle.

The remaining elements of the Write Data path reside in the array and pitch circuits. As previously discussed in Sections 1.2, 2.1, and 2.2, the Mbits and sense amplifier block constitute the end of the Write Data path. The new input data is driven by the Write driver, propagating through the *I/O* transistors and into the sense amplifier circuits. After the sense amplifiers are overwritten and the new data is latched, the Write drivers are no longer needed and can be disabled. Completion of the Write operation into the Mbits is accomplished by the sense amplifiers, which restore the digitlines to full V_{CC} and ground levels. See Sections 1.2 and 2.2 for further discussion.

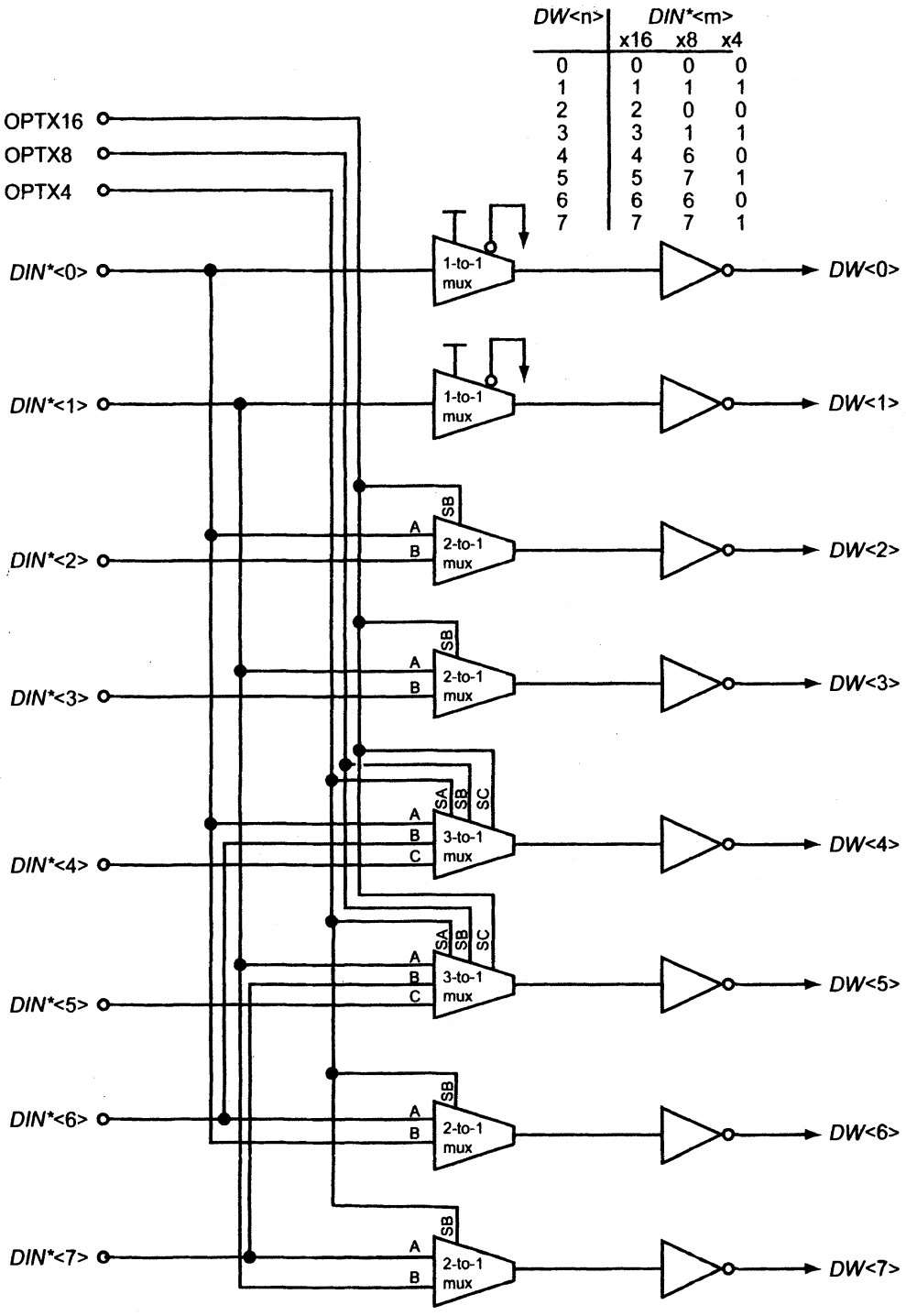


Figure 5.6 Data Write mux.

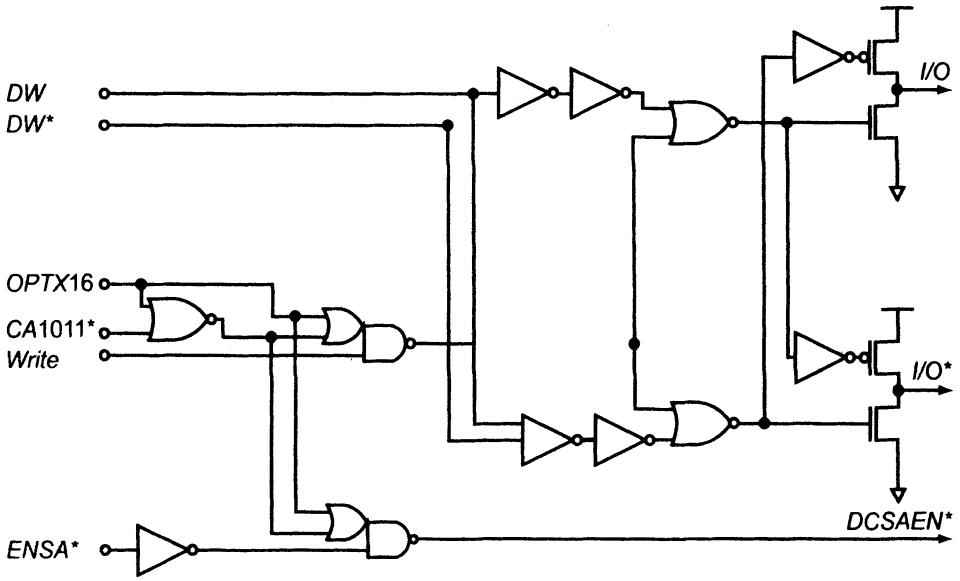


Figure 5.7 Write driver.

5.1.4 Data Read Path

The Data Read path is similar, yet complementary, to the Data Write path. It begins, of course, in the array, as previously discussed in Sections 1.2 and 2.2. After data is read from the Mbit and latched by the sense amplifiers, it propagates through the I/O transistors onto the *I/O* signal lines and into a *DC sense amplifier* (DCSA) or *helper flip-flop* (HFF). The *I/O* lines, prior to the *column select (CSEL)* firing, are equilibrated and biased to a voltage approaching V_{CC}. The actual bias voltage is determined by the *I/O* bias circuit, which serves to control the *I/O* lines through every phase of the Read and Write cycles. This circuit, as shown in Figure 5.8, consists of a group of *bias and equilibrate* transistors that operate in concert with a variety of control signals. When the DRAM is in an idle state, such as when *RAS** is HIGH, the *I/O* lines are generally biased to V_{CC}. During a Read cycle and prior to *CSEL,* the bias is reduced to approximately one V_{TH} below V_{CC}.

The actual bias voltage for a Read operation is optimized to ensure sense amplifier stability and fast sensing by the DCSA or HFF circuits. Bias is maintained continually throughout a Read cycle to ensure proper DCSA operation and to speed equilibration between cycles by reducing the range over which the *I/O* lines operate. Furthermore, because the *DCSA*s or *HFF*s are very high-gain amplifiers, rail-to-rail input signals are not necessary to drive the outputs to CMOS levels. In fact, it is important that the input levels not exceed the DCSA or HFF common-mode operating range. During a Write operation, the bias circuits are disabled by the Write signal, permitting the Write drivers to drive rail-to-rail.

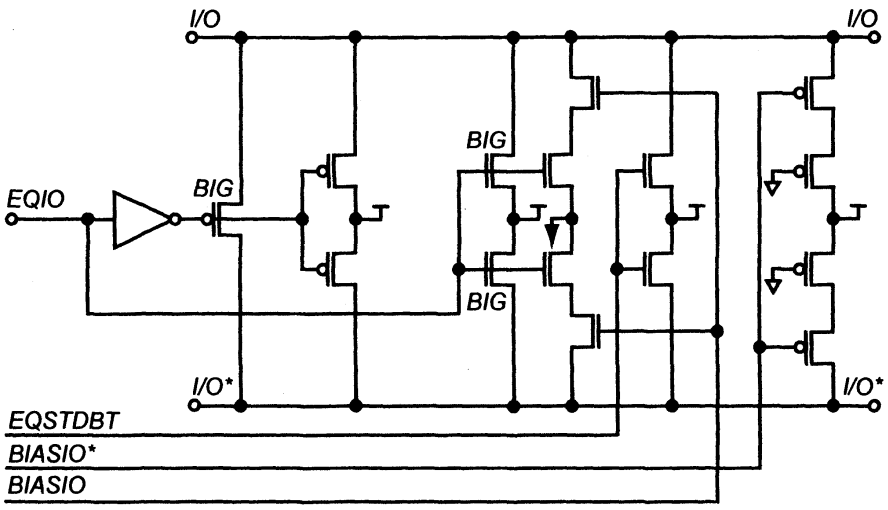


Figure 5.8 *I/O* bias circuit.

Operation of the bias circuits is seen in the signal waveforms shown in Figure 5.9. For the Read-Modify-Write cycle, the *I/O* lines start at V_{CC} during standby; transition to $V_{CC}-V_{TH}$ at the start of a Read cycle; separate but remain biased during the Read cycle; drive rail-to-rail during a Write cycle; recover to Read cycle levels (termed *Write Recovery*); and equilibrate to $V_{CC}-V_{TH}$ in preparation for another Read cycle.

5.1.5 DC Sense Amplifier (DCSA)

The next data path element is the *DC sense amplifier* (DCSA). This amplifier, termed Data amplifier or Read amplifier by DRAM manufacturers, is an essential component in modern, high-speed DRAM designs and takes a variety of forms. In essence, the DCSA is a high-speed, high-gain differential amplifier for amplifying very small Read signals appearing on the *I/O* lines into full CMOS data signals used at the output data buffer. In most designs, the *I/O* lines connected to the sense amplifiers are very capacitive.

The array sense amplifiers have very limited drive capability and are unable to drive these lines quickly. Because the DCSA has very high gain, it amplifies even the slightest separation of the *I/O* lines into full CMOS levels, essentially gaining back any delay associated with the *I/O* lines. Good DCSA designs can output full rail-to-rail signals with input signals as small as 15mV. This level of performance can only be accomplished through very careful design and layout. Layout must follow good analog design principles, with each element a direct copy (no mirrored layouts) of any like elements.

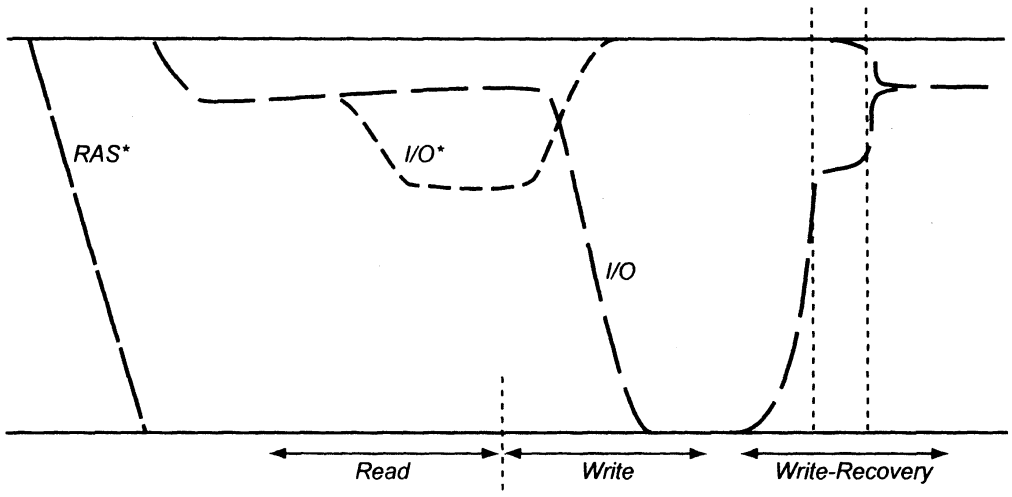


Figure 5.9 *I/O* bias and operation waveforms.

As illustrated in Figure 5.10, a typical DCSA consists of four differential pair amplifiers and self-biasing CMOS stages. The differential pairs are configured as two sets of balanced amplifiers. Generally, the amplifiers are built with an NMOS differential pair using PMOS active loads and NMOS current mirrors. Because NMOS has higher mobility, providing for smaller transistors and lower parasitic loads, NMOS amplifiers usually offer faster operation than PMOS amplifiers. Furthermore, V_{TH} matching is usually better for NMOS, offering a more balanced design. The first set of amplifiers is fed with *I/O* and *I/O** signals from the array; the second set, with the output signals from the first pair, labeled *DX* and *DX**. Bias levels into each stage are carefully controlled to provide optimum performance.

The outputs from the second stage, labeled *DY* and *DY**, feed into self-biasing CMOS inverter stages for fast operation. The final output stage is capable of tristate operation to allow multiple sets of DCSA to drive a given set of *Data Read* lines *(DR*<n> and *DR**<n>*)*. The entire amplifier is equilibrated prior to operation, including the self-biasing CMOS inverter stages, by all of the devices connected to the signals labeled *EQSA*, *EQSA**, and *EQSA*2. Equilibration is necessary to ensure that the amplifier is electrically balanced and properly biased before the input signals are applied. The amplifier is enabled whenever *ENSA** is brought LOW, turning ON the output stage and the current mirror bias circuit, which is connected to the differential amplifiers via the signal labeled *CM*. For a DRAM Read cycle, operation of this amplifier is depicted in the waveforms shown in Figure 5.11. The bias levels are reduced for each amplifier stage, approaching $V_{CC}/2$ for the final stage.

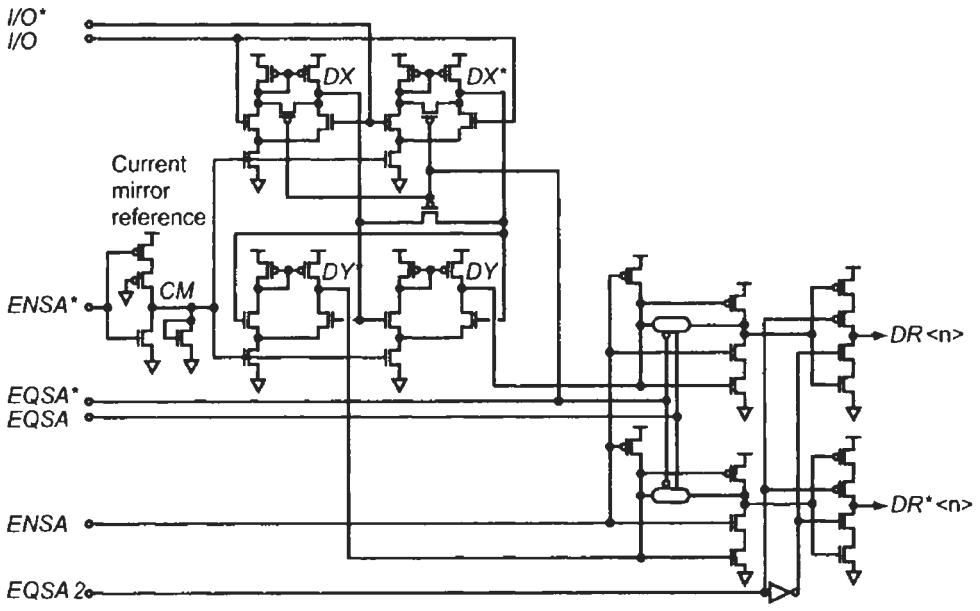


Figure 5.10 DC sense amp.

5.1.6 Helper Flip-Flop (HFF)

The DCSA of the last section can require a large layout area. To reduce the area, a helper flip-flop (HFF) as seen in Figure 5.12, can be used. The HFF is basically a clocked connection of two inverters as a latch [3]. When *CLK* is LOW, the *I/O* lines are connected to the inputs/outputs of the inverters. The inverters don't see a path to ground because M1 is OFF when *CLK* is LOW. When *CLK* transitions HIGH, the outputs of the HFF amplify, in effect, the inputs into full logic levels.

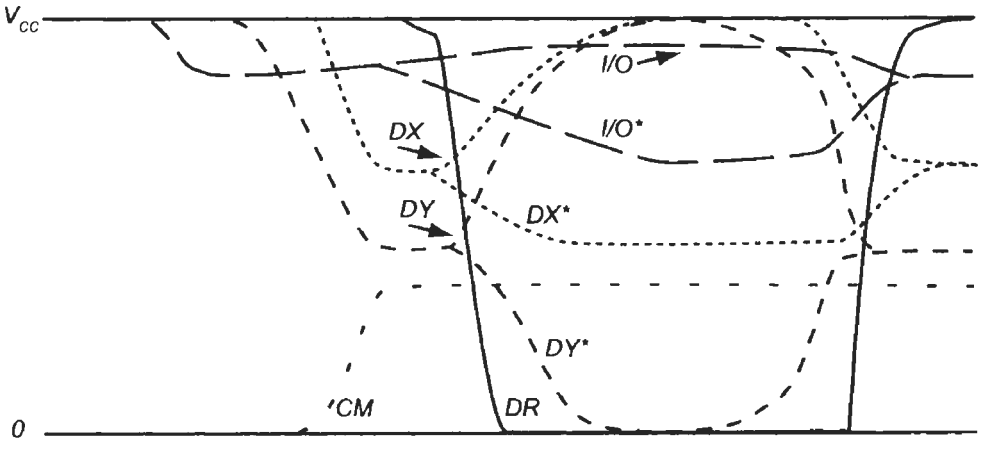


Figure 5.11 DCSA operation waveforms.

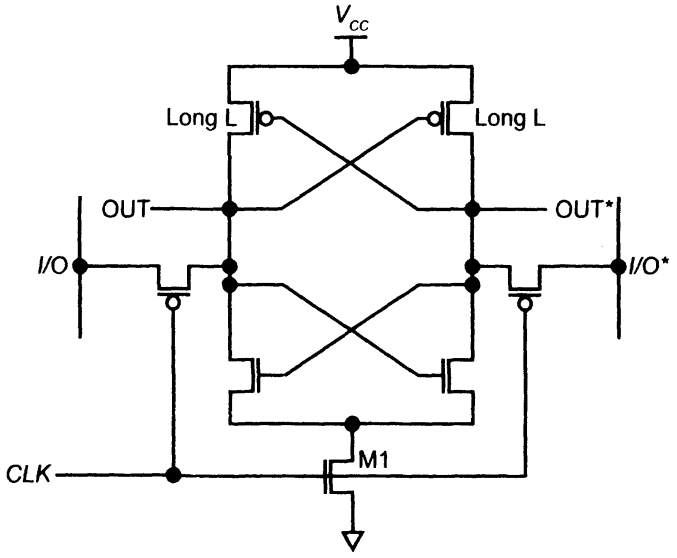
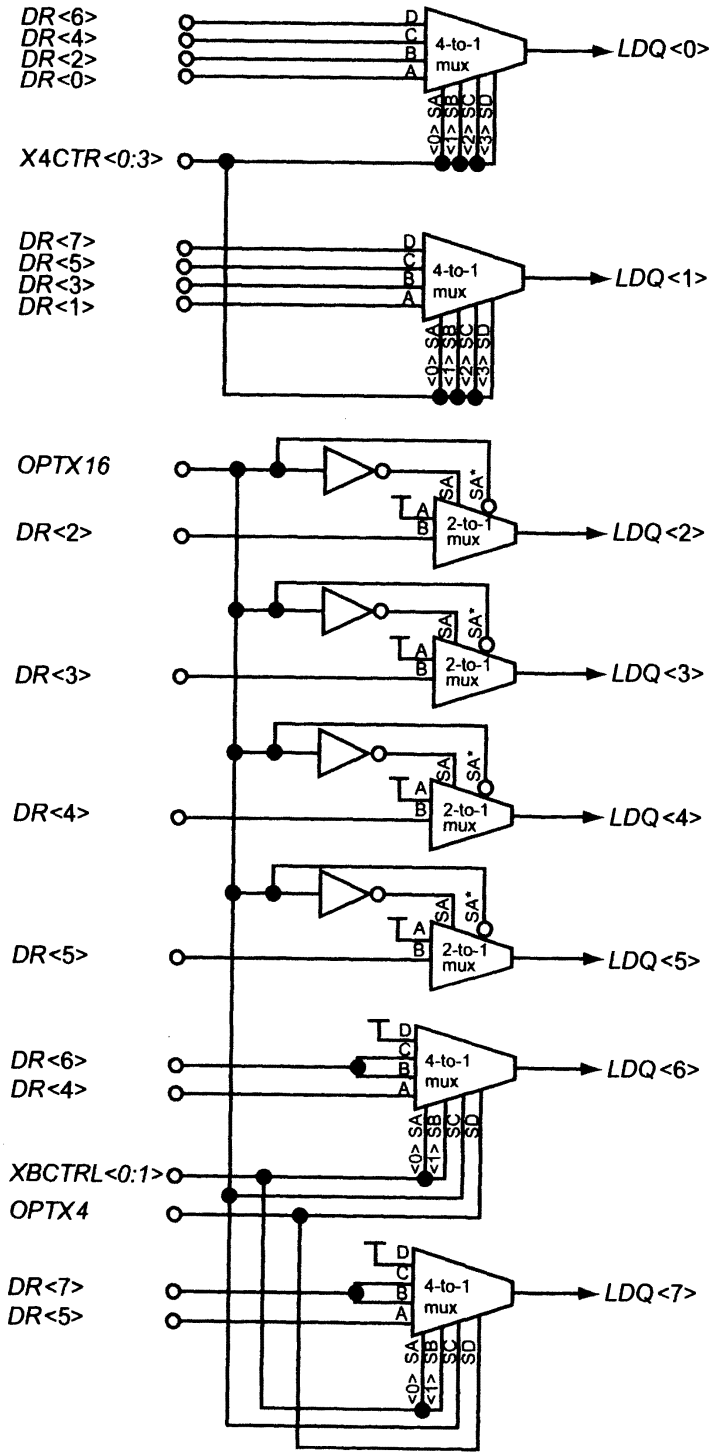


Figure 5.12 A helper flip-flop.

For example, if *I/O* = 1.25 V and *I/O** = 1.23 V, then *I/O* becomes V_{CC}, and *I/O** goes to zero when *CLK* transitions HIGH. Using positive feedback makes the HFF sensitive and fast. Note that *HFF*s can be used at several locations on the *I/O* lines due to the small size of the circuit.

5.1.7 Data Read Muxes

The Read Data path proceeds from the DCSA block to the output buffers. The connection between these elements can either be direct or through Data Read muxes. Similar to Data Write muxes, Data Read muxes are commonly used to accommodate multiple-part configurations with a single design. An example of this is shown in Figure 5.13. This schematic of a Data Read mux block is similar to that found in Figure 5.6 for the Data Write mux block. For x16 operation, each output buffer has access to only one Data Read line pair *(DR*<n> and *DR**<n>*)*. For x8 operation, the eight output buffers each have two pairs of *DR*<n> lines available, doubling the quantity of Mbits accessible by each output. Similarly, for x4 operation, the four output buffers have four pairs of *DR*<n> lines available, again doubling the quantity of Mbits available for each output. For those configurations with multiple pairs available, address lines control which *DR*<n> pair is connected to an output buffer.



LDQ<n>	Read data x16	x8	x4
0	0	0,2	0,2,4,6
1	1	1,3	1,3,5,7
2	2		
3	3		
4	4		
5	5		
6	6	4,6	
7	7	5,7	

Figure 5.13 Data Read mux.

5.1.8 Output Buffer Circuit

The final element in our Read Data path is the output buffer circuit. It consists of an output latch and an output driver circuit. A schematic for an output buffer circuit is shown in . The output driver on the right side of uses three NMOS transistors to drive the output pad to either V_{CCX} or ground. V_{CCX} is the external supply voltage to the DRAM, which may or may not be the same as V_{CC}, depending on whether or not the part is internally regulated. Output drivers using only NMOS transistors are common because they offer better latch-up immunity and ESD protection than CMOS output drivers. Nonetheless, PMOS transistors are still used in CMOS output drivers by several DRAM manufacturers, primarily because they are much easier to drive than full NMOS stages. CMOS outputs are also more prevalent on high-speed synchronous and double-data rate (DDR) designs because they operate at high data rates with less duty-cycle degradation.

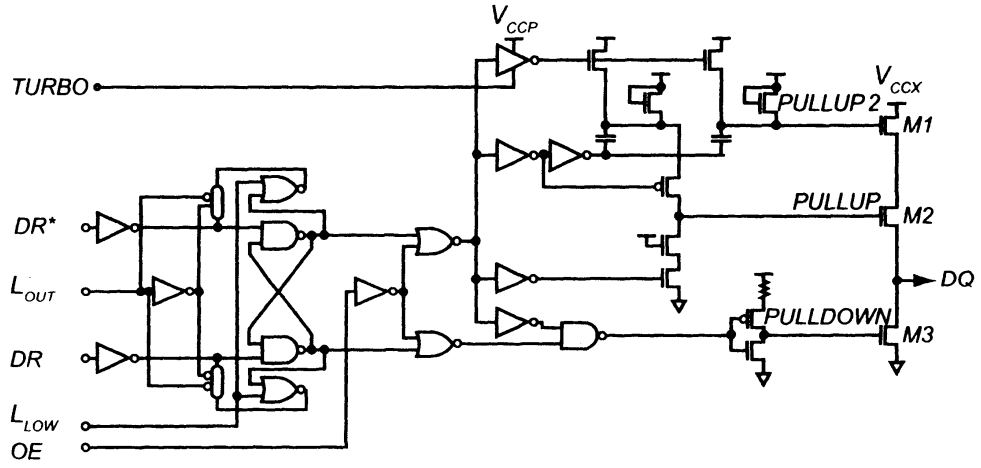


Figure 5.14 Output buffer.

In , two NMOS transistors are placed in series with V_{CCX} to reduce substrate injection currents. Substrate injection currents result from impact ionization, occurring most commonly when high drain-to-source and high gate-to-source voltages exist concurrently. These conditions usually occur when an output driver is firing to V_{CCX}, especially for high-capacitance loads, which slow the output transition. Two transistors in series reduce this effect by lowering the voltages across any single device. The output stage is tristated whenever both signals *PULLUP* and *PULLDOWN* are at ground.

The signal *PULLDOWN* is driven by a simple CMOS inverter, whereas *PULLUP* is driven by a complex circuit that includes voltage charge pumps. The pumps generate a voltage to drive *PULLUP* higher than one V_{TH} above V_{CCX}. This is necessary to ensure that the series output transistors drive the

pad to V_{CCX}. The output driver is enabled by the signal labeled *OE*. Once enabled, it remains tristated until either *DQ* or *DQ** fires LOW. If *DR* fires LOW, *PULLDOWN* fires HIGH, driving the pad to ground through M3. If *DR** fires LOW, *PULLUP* fires HIGH, driving the pad to V_{CCX} through M1 and M2.

The output latch circuit shown in controls the output driver operation. As the name implies, it contains a latch to hold the output data state. The latch frees the DCSA or HFF and other circuits upstream to get subsequent data for the output. It is capable of storing not only one and zero states, but also a high-impedance state *(tristate)*. It offers transparent operation to allow data to quickly propagate to the output driver. The input to this latch is connected to the *DR*<n> signals coming from either the *DCSA*s or Data Read muxes. Output latch circuits appear in a variety of forms, each serving the needs of a specific application or architecture. The data path may contain additional latches or circuits in support of special modes such as burst operation.

5.1.9 Test Modes

Address compression and data compression are two special test modes that are usually supported by the data path design. Test modes are included in a DRAM design to extend test capabilities or speed component testing or to subject a part to conditions that are not seen during normal operation. Compression test modes yield shorter test times by allowing data from multiple array locations to be tested and compressed on-chip, thereby reducing the effective memory size by a factor of 128 or more in some cases. Address compression, usually on the order of 4x to 32x, is accomplished by internally treating certain address bits as "don't care" addresses.

The data from all of the "don't care" address locations, which correspond to specific data input/output pads *(DQ* pins*)*, are compared using special match circuits. Match circuits are usually realized with NAND and NOR logic gates or through P&E-type drivers on the differential *DR*<n> buses. The match circuits determine if the data from each address location is the same, reporting the result on the respective *DQ* pin as a match or a fail. The data path must be designed to support the desired level of address compression. This may necessitate more DCSA circuits, logic, and pathways than are necessary for normal operation.

The second form of test compression is data compression: combining data at the output drivers. Data compression usually reduces the number of *DQ* pins to four. This compression reduces the number of tester pins required for each part and increases the throughput by allowing additional parts to be tested in parallel. In this way, x16 parts accommodate 4x data

compression, and x8 parts accommodate 2x data compression. The cost of any additional circuitry to implement address and data compression must be balanced against the benefits derived from test time reduction. It is also important that operation in test mode correlate 100% with operation in non-test mode. Correlation can be difficult to achieve, however, because additional circuitry must be activated during compression, modifying noise and power characteristics on the die.

5.2 ADDRESS PATH ELEMENTS

DRAMs have used multiplexed addresses since the 4kb generation. Multiplexing is possible because DRAM operation is sequential: column operations follow row operations. Obviously, the column address is not needed until the sense amplifiers have latched, which cannot occur until some time after the wordline has fired. DRAMs operate at higher current levels with multiplexed addressing because an entire page (row address) must be opened with each row access. This disadvantage is overcome by the lower packaging cost associated with multiplexed addresses. In addition, owing to the presence of the *column address strobe (CAS*),* column operation is independent of row operation, enabling a page to remain open for multiple, high-speed column accesses. This page mode type of operation improves system performance because column access time is much shorter than row access time. Page mode operation appears in more advanced forms, such as EDO and synchronous burst mode, providing even better system performance through a reduction in effective column access time.

The address path for a DRAM can be broken into two parts: the row address path and the column address path. The design of each path is dictated by a unique set of requirements. The address path, unlike the data path, is unidirectional, with address information flowing only into the DRAM. The address path must achieve a high level of performance with minimal power and die area just like any other aspect of DRAM design. Both paths are designed to minimize propagation delay and maximize DRAM performance. In this chapter, we discuss various elements of the row and column address paths.

5.2.1 Row Address Path

The row address path encompasses all of the circuits from the address input pad to the wordline driver. These circuits generally include the address buffer, *CAS** before *RAS** *(CBR)* counter, predecode logic, array buffers, redundancy logic, phase drivers, and row decoder blocks. Row decoder blocks are discussed in Section 2.3, while redundancy is addressed in Section 4.2. We will now focus on the remaining elements of the row

address path, namely, the row address buffer, *CBR* counter, predecode logic, array buffers, and phase drivers.

5.2.2 Row Address Buffer

The row address buffer, as shown schematically in Figure 5.15, consists of a standard input buffer and the additional circuits necessary to implement the functions required for the row address path. The address input buffer *(inpBuf)* is the same as that used for the data input buffer (see Figure 5.1) and must meet the same criteria for V_{IH} and V_{IL} as the data input buffer. The row address buffer includes an inverter latch circuit, as shown in Figure 5.15. This latch, consisting of two inverters and any number of input muxes (two in this case), latches the row address after *RAS** falls.

The input buffer drives through a mux, which is controlled by a signal called *row address latch (RAL)*. Whenever *RAL* is LOW, the mux is enabled. The feedback inverter has low drive capability, allowing the latch to be overwritten by either the address input buffer or the *CBR* counter, depending on which mux is enabled. The latch circuit drives into a pair of NAND gates, forcing both *RA*<n> and *RA**<n> to logic LOW states whenever the row address buffer is disabled because *RAEN* is LOW.

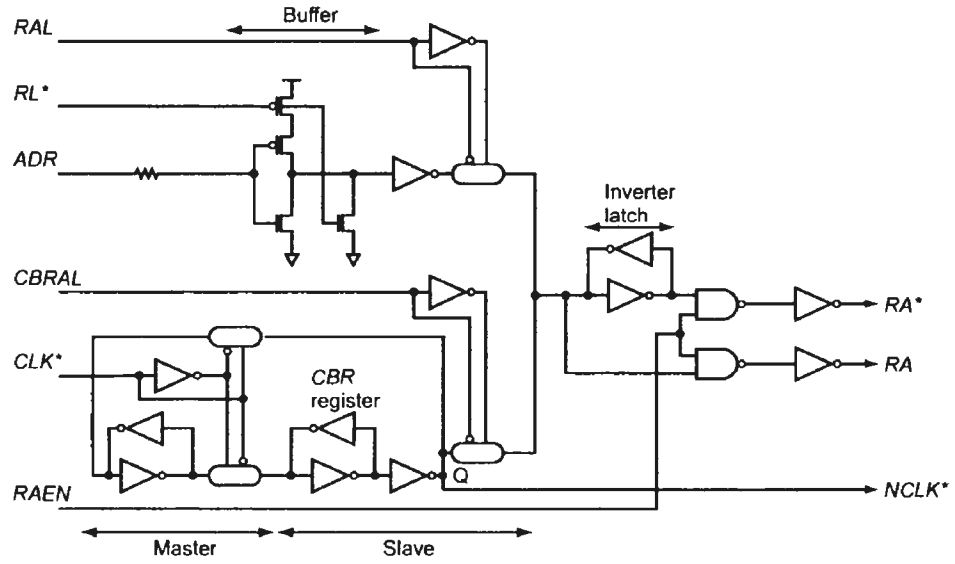


Figure 5.15 Row address buffer.

5.2.3 CBR Counter

As illustrated in Figure 5.15, the *CBR (CAS** before *RAS*)* counter consists of a single inverter and a pair of inverter latches coupled through a pair of complementary muxes to form a one-bit counter. For every HIGH-to-LOW transition of *CLK**, the register output at Q toggles. All of the *CBR*

counters from each row address buffer are cascaded together to form a *CBR* ripple counter. The Q output of one stage feeds the *CLK** input of a subsequent stage. The first register in the counter is clocked whenever *RAS** falls while in a *CBR* Refresh mode. By cycling through all possible row address combinations in a minimum of clocks, the *CBR* ripple counter provides a simple means of internally generating Refresh addresses. The *CBR* counter drives through a mux into the inverter latch of the row address buffer. This mux is enabled whenever *CBR address latch (CBRAL)* is LOW. Note that the signals *RAL* and *CBRAL* are mutually exclusive in that they cannot be LOW at the same time. For each and every DRAM design, the row address buffer and *CBR* counter designs take on various forms. Logic may be inverted, counters may be more or less complex, and muxes may be replaced with static gates. Whatever the differences, however, the function of the input buffer and its *CBR* counter remains essentially the same.

5.2.4 Predecode Logic

As discussed in earlier sections of this book, using predecoded addressing internal to a DRAM has many advantages: lower power, higher efficiency, and a simplified layout. An example of predecode circuits for the row address path is shown in Figure 5.16. This schematic consists of seven sets of predecoders. The first set is used for even and odd row selection and consists only of cascaded inverter stages. *RA*<0> is not combined with other addresses due to the nature of row decoding. In our example, we assume that odd and even will be combined with the predecoded row addresses at a later point in the design, such as at the array interfaces. The next set of predecoders is for addresses *RA*<1> and *RA*<2>, which together form *RA*12<0:3>. Predecoding is accomplished through a set of two-input NAND gates and inverter buffers, as shown. The remaining addresses are identically decoded except for *RA*<12>. As illustrated at the bottom of the schematic, *RA*<12> and *RA**<12> are taken through a NOR gate circuit. This circuit forces both *RA*<12> lines to a HIGH state as they enter the decoder, whenever the DRAM is configured for 4k Refresh. This process essentially makes *RA*<12> a "don't care" address in the predecode circuit, forcing twice as many wordlines to fire at a time.

5.2.5 Refresh Rate

Normally, when Refresh rates change for a DRAM, a higher order address is treated as a "don't care" address, thereby decreasing the row address space but increasing the column address space. For example, a 16Mb DRAM bonded as a 4Mb x4 part could be configured in several Refresh rates: 1k, 2k, and 4k. Table 5.1 shows how row and column

addressing is related to these Refresh rates for the 16Mb example. In this example, the 2k Refresh rate would be more popular because it has an equal number of row and column addresses or *square* addressing.

Refresh rate is also determined by backward compatibility, especially in personal computer designs. Because a 4Mb DRAM has less memory space than a 16Mb DRAM, the 4Mb DRAM should naturally have fewer address pins. To sell 16Mb DRAMs into personal computers that are designed for 4Mb DRAMs, the 16Mb part must be configured with no more address pins than the 4Mb part. If the 4Mb part has eleven address pins, then the 16Mb part should have only eleven address pins, hence 2k Refresh. To trim cost, most PC designs keep the number of DRAM address pins to a minimum. Although this practice holds cost down, it also limits expandability and makes conversion to newer DRAM generations more complicated owing to resultant backward compatibility issues.

Table 5.1 Refresh rate versus row and column addresses.

Refresh rate	Rows	Columns	Row addresses	Column addresses
4K	4,096	1,024	12	10
2K	2,048	2,048	11	11
1K	1,024	4,096	10	12

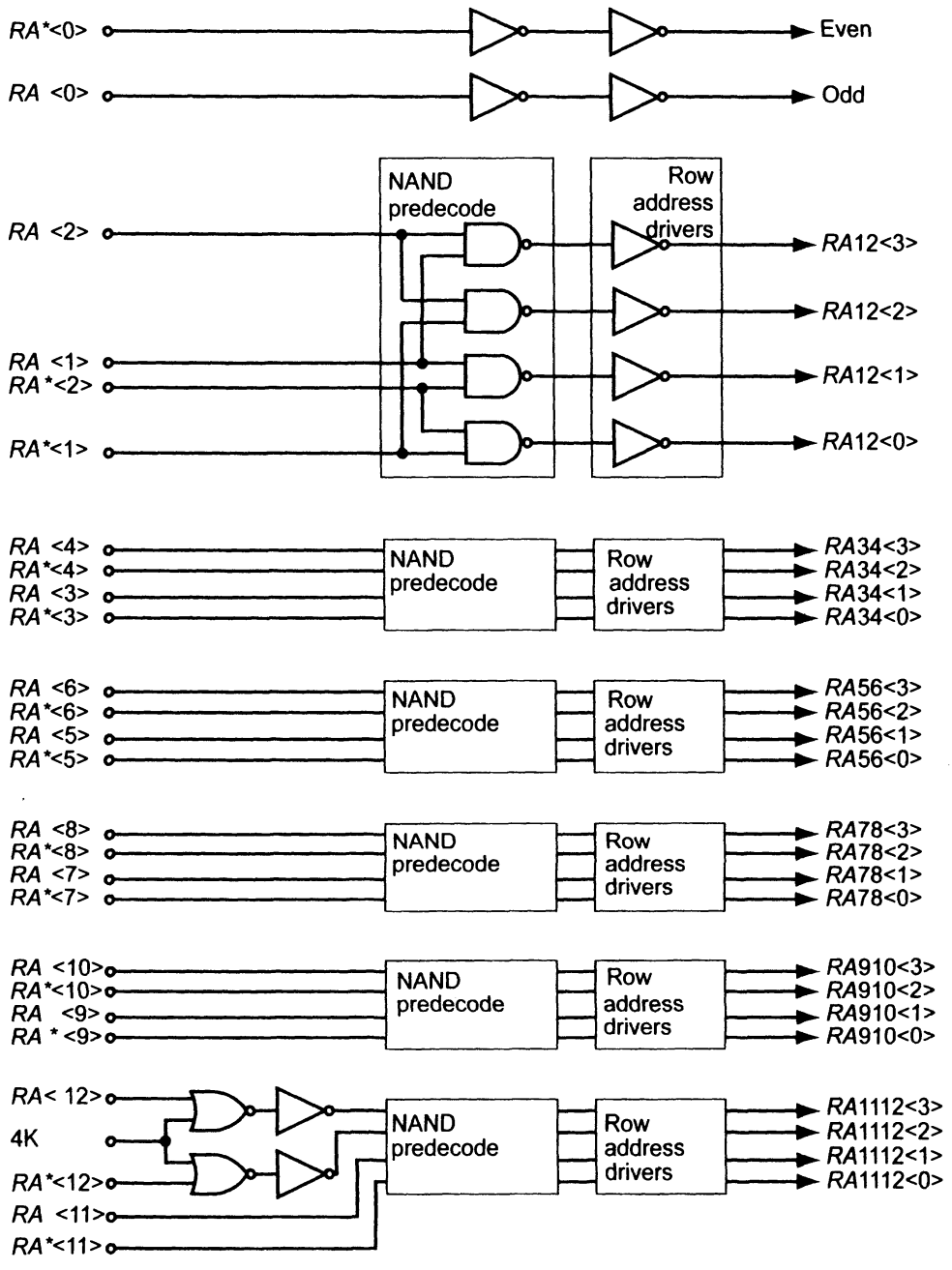


Figure 5.16 Row address predecode circuits.

5.2.6 Array Buffers

The next elements to be discussed in the row address path are array buffers and phase drivers. The array buffers drive the predecoded address signals into the row decoder blocks. In general, the buffers are no more than cascaded inverters, but in some cases they include static logic gates or level translators, depending on row decoder requirements. Additional logic gates

could be included for combining the addresses with enabling signals from the control logic or for making odd/even row selection by combining the addresses with the odd and even address signals. Regardless, the resulting signals ultimately drive the decode trees, making speed an important issue. Buffer size and routing resistance, therefore, become important design parameters in high-speed designs because the wordline cannot be fired until the address tree is decoded and ready for the *PHASE* signal to fire.

5.2.7 Phase Drivers

As the discussion concerning wordline drivers and tree decoders in Section 2.3 showed, the signal that actually fires the wordline is called *PHASE*. Although the signal name may vary from company to company, the purpose of the signal does not. Essentially, this signal is the final address term to arrive at the wordline driver. Its timing is carefully determined by the control logic. *PHASE* cannot fire until the row addresses are set up in the decode tree. Normally, the timing of *PHASE* also includes enough time for the row redundancy circuits to evaluate the current address. If a redundancy match is found, the normal row cannot be fired. In most DRAM designs, this means that the normally decoded *PHASE* signal will not fire but will instead be replaced by some form of redundant *PHASE* signal.

A typical phase decoder/driver is shown in Figure 5.17. Again, like so many other DRAM circuits, it is composed of standard static logic gates. A level translator would be included in the design if the wordline driver required a *PHASE* signal that could drive to a boosted voltage supply. This translator could be included with the phase decoder logic or placed in array gaps to locally drive selected row decoder blocks. Local phase translators are common on double-metal designs with local row decoder blocks.

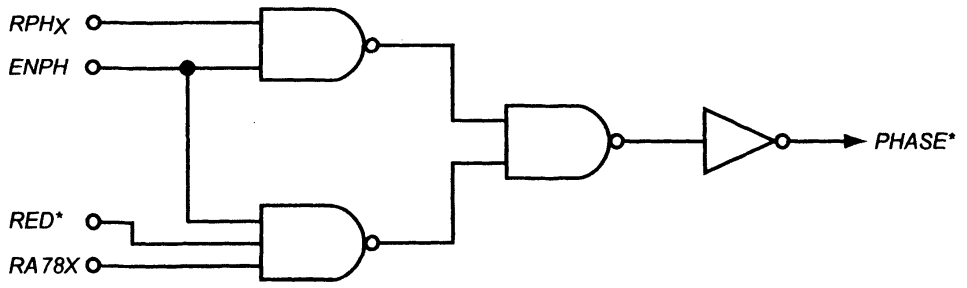


Figure 5.17 Phase decoder/driver.

5.2.8 Column Address Path

With our examination of row address path elements complete, we can turn our attention to the column address path. The column address path consists of input buffers, address transition detection circuits, predecode logic,

redundancy, and column decode circuits. Redundancy and column decode circuits are addressed in Section 4.2 and Section 4.1, respectively.

A column address buffer schematic, as shown in Figure 5.18, consists of an input buffer, a latch, and address transition circuits. The input buffer shown in this figure is again identical to that described in Section 5.1 for the data path, so further description is unnecessary. The column address input buffer, however, is disabled by the signal *power column (PCOL*)* whenever *RAS** is HIGH and the part is inactive. Normally, the column address buffers are enabled by *PCOL** shortly after *RAS** goes LOW. The input buffer feeds into a NAND latch circuit, which traps the column address whenever the *column address latch (CAL*)* fires LOW. At the start of a column cycle, *CAL** is HIGH, making the latch transparent. This transparency permits the column address to propagate through to the predecode circuits. The latch output also feeds into an *address transition detection (ATD)* circuit that is shown on the right side of Figure 5.18.

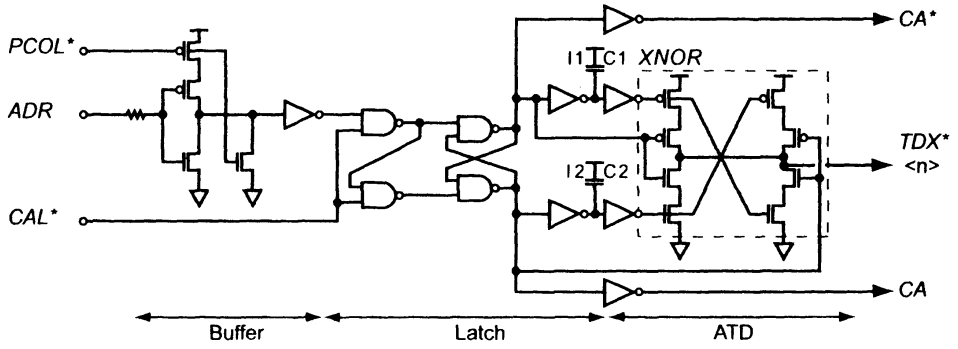


Figure 5.18 Column address buffer.

5.2.9 Address Transition Detection

The address transition detection *(ATD)* circuit is extremely important to page mode operation in an asynchronous DRAM. An *ATD* circuit detects any transition that occurs on a respective address pin. Because it follows the NAND latch circuit, the *CAL** control signal must be HIGH. This signal makes the latch transparent and thus the *ATD* functional. The *ATD* circuit in Figure 5.18 has symmetrical operation: it can sense either a rising or a falling edge. The circuit uses a two-input XNOR gate to generate a pulse of prescribed duration whenever a transition occurs. Pulse duration is dictated by simple delay elements: I1 and C1 or I2 and C2. Both delayed and undelayed signals from the latch are fed into the XNOR gate. The *ATD* output signals from all of the column addresses, labeled *TDX*<n>*, are routed to the equilibration driver circuit shown in Figure 5.19. This circuit generates a set of equilibration signals for the DRAM. The first of these signals is *equilibrate I/O (EQIO*),* which, as the name implies, is used in arrays to force

equilibration of the *I/O* lines. As we learned in Section 5.1.4, the I/O lines need to be equilibrated to $V_{CC}-V_{TH}$ prior to a new column being selected by the *CSEL*<n> lines. *EQIO** is the signal used to accomplish this equilibration.

The second signal generated by the equilibration driver is called *equilibrate sense amp (EQSA)*. This signal is generated from address transitions occurring on all of the column addresses, including the *least significant* addresses. The least significant column addresses are not decoded into the *column select lines (CSEL)*. Rather, they are used to select which set of *I/O* lines is connected to the output buffers. As shown in the schematic, *EQSA* is activated regardless of which address is changed because the *DCSA*s must be equilibrated prior to sensing any new data. *EQIO,* on the other hand, is not affected by the least significant addresses because the *I/O* lines do not need equilibrating unless the *CSEL* lines are changed. The equilibration driver circuit, as shown in Figure 5.19, uses a balanced NAND gate to combine the pulses from each *ATD* circuit. Balanced logic helps ensure that the narrow *ATD* pulses are not distorted as they progress through the circuit.

The column addresses are fed into predecode circuits, which are very similar to the row address predecoders. One major difference, however, is that the column addresses are not allowed to propagate through the part until the wordline has fired. For this reason, the signal *Enable column (ECOL)* is gated into the predecode logic as shown in Figure 5.20. *ECOL* disables the predecoders whenever it is LOW, forcing the outputs all HIGH in our example. Again, the predecode circuits are implemented with simple static logic gates. The address signals emanating from the predecode circuits are buffered and distributed throughout the die to feed the column decoder logic blocks. The column decoder elements are described in Section 4.1.

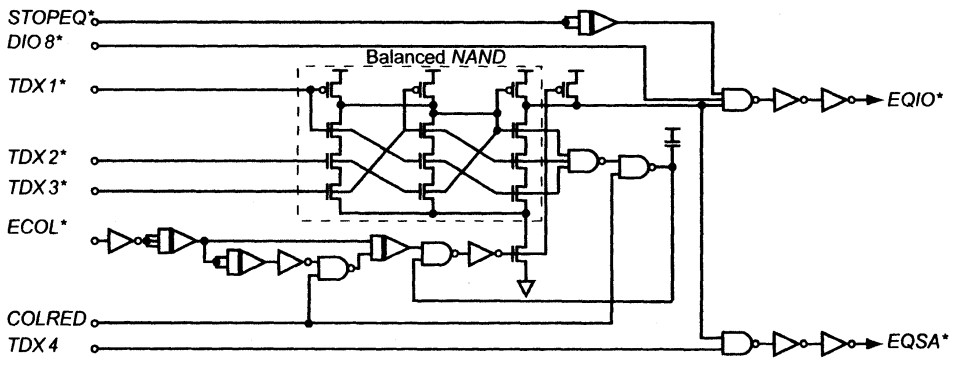


Figure 5.19 Equilibration driver.

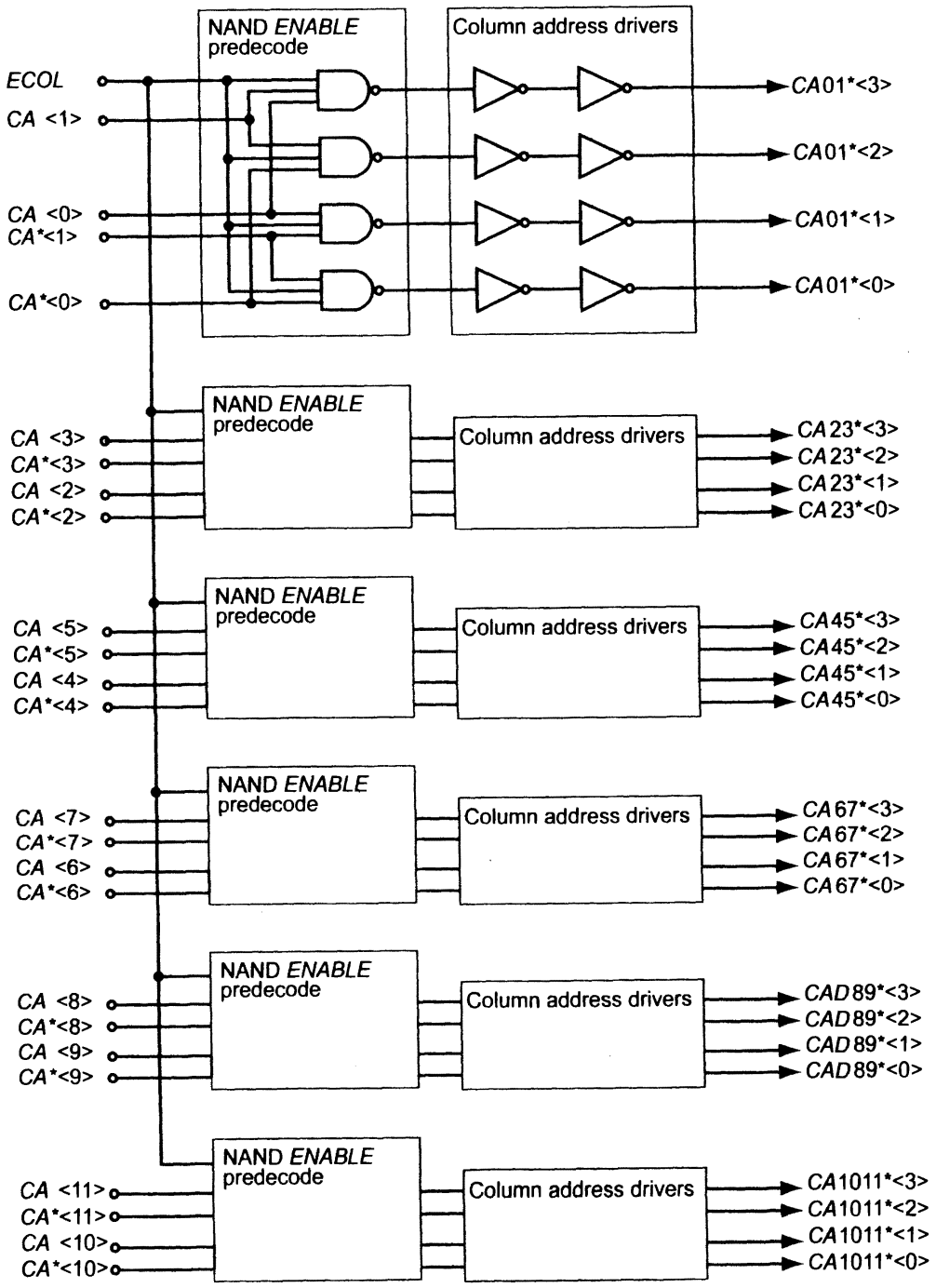


Figure 5.20 Column predecode logic.

5.3 SYNCHRONIZATION IN DRAMS[1]

In a typical SDRAM, the relationship when reading data out of the DRAM between the *CLK* input and the valid data out time or *access time* t_{AC} can vary widely with process shifts, temperature, and operating clock frequency. Figure 5.21 shows a typical relationship between the SDRAM, *CLK* input, and a *DQ* output. The parameter specification t_{AC} is generally specified as being less than some value, for example, $t_{AC} < 5$ns.

As clock frequencies increase, it is desirable to have less uncertainty in the availability of valid data on the output of the DRAM. Towards this goal, double data rate (DDR) SDRAMs use a *delay-locked loop* (DLL) [3] to drive t_{AC} to zero. (A typical specification for t_{AC} in a DDR SDRAM is ± 0.75ns.) Figure 5.22 shows the block diagram for a DLL used in a DDR SDRAM. Note that the data *I/O* in DDR is clocked on both the rising and falling edges of the input *CLK* (actually on the output of the delay line) [4]. Also, note that the input *CLK* should be synchronized with the *DQ strobe (DQS)* clock output.

To optimize and stabilize the clock-access and output-hold times in an SDRAM, an internal *register-controlled delay-locked loop* (RDLL) has been used [4, 5, 6]. The RDLL adjusts the time difference between the output (*DQ*s) and input (*CLK*) clock signals in SDRAM until they are aligned. Because the RDLL is an all-digital design, it provides robust operation over all process corners. Another solution to the timing constraints found in SDRAM was given by using the *synchronous mirror delay* (SMD) in [7]. Compared to RDLL, the SMD does not lock as tightly, but the time to acquire lock between the input and output clocks is only two clock cycles.

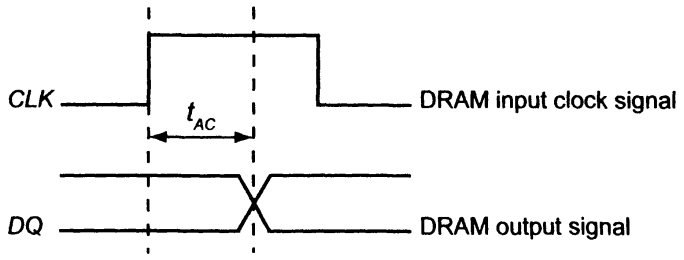


Figure 5.21 SDRAM *CLK* input and *DQ* output.

1.This material is taken directly from [4].

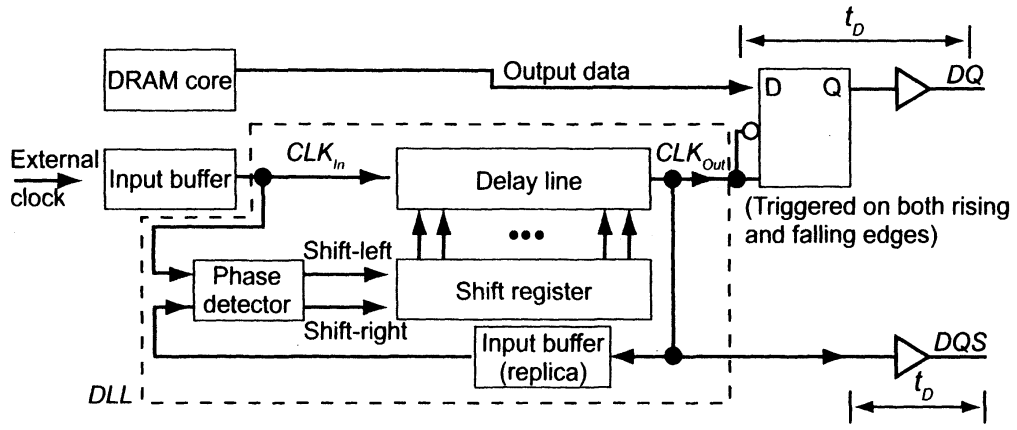


Figure 5.22 Block diagram for DDR SDRAM DLL.

As DRAM clock speeds continue to increase, the skew becomes the dominating concern, outweighing the RDLL disadvantage of longer time to acquire lock.

This section describes an RSDLL (register-controlled symmetrical DLL), which meets the requirements of DDR SDRAM. (Read/Write accesses occur on both rising and falling edges of the clock.) Here, *symmetrical* means that the delay line used in the DLL has the same delay whether a HIGH-to-LOW or a LOW-to-HIGH logic signal is propagating along the line. The data output timing diagram of a DDR SDRAM is shown in Figure 5.23. The RSDLL increases the valid output data window and diminishes DQS to DQ skew by synchronizing both the rising and falling edges of the *DQS* signal with the output data *DQ*.

Figure 5.22 shows the block diagram of the RSDLL. The replica input buffer dummy delay in the feedback path is used to match the delay of the input clock buffer. The *phase detector (PD)* compares the relative timing of the edges of the input clock signal and the feedback clock signal, which comes through the delay line and is controlled by the shift register. The outputs of the *PD,* shift-right and shift-left, control the shift register. In the simplest case, one bit of the shift register is HIGH. This single bit selects a point of entry for *CLKIn* the symmetrical delay line. (More on this later.) When the rising edge of the input clock is within the rising edges of the output clock and one unit delay of the output clock, both outputs of the *PD*, shift-right and shift-left, go LOW and the loop is locked.

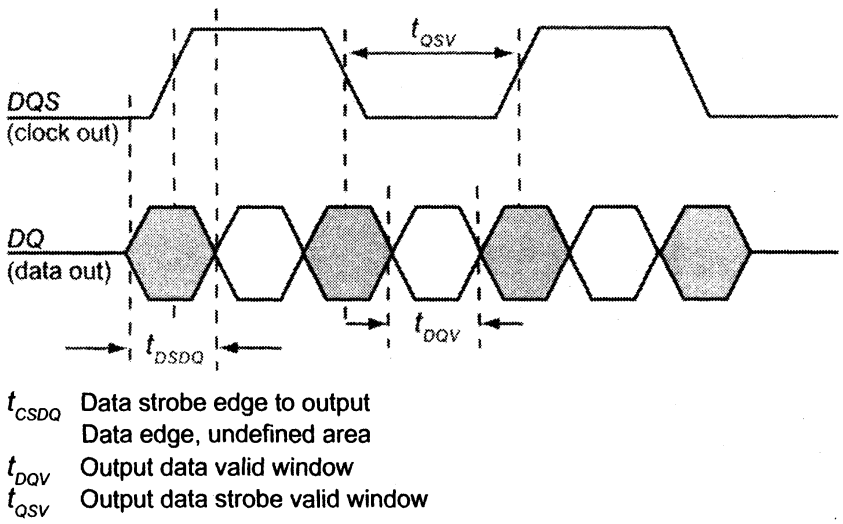


Figure 5.23 Data timing chart for DDR DRAM.

5.3.1 The Phase Detector

The basic operation of the *phase detector (PD)* is shown in Figure 5.24. The resolution of this RSDLL is determined by the size of the unit delay used in the delay line. The locking range is determined by the number of delay stages used in the symmetrical delay line. Because the DLL circuit inserts an optimum delay time between *CLKIn* and *CLKOut,* making the output clock change simultaneously with the next rising edge of the input clock, the minimum operating frequency to which the RSDLL can lock is the reciprocal of the product of the number of stages in the symmetrical delay line with the delay per stage. Adding more delay stages increases the locking range of the RSDLL at the cost of increased layout area.

5.3.2 The Basic Delay Element

Rather than using an AND gate as the unit-delay stage (NAND + inverter), as in [5], a NAND-only-based delay element can be used. The implementation of a three-stage delay line is shown in Figure 5.25. The problem when using a NAND + inverter as the basic delay element is that the propagation delay through the unit delay resulting from a HIGH-to-LOW transition is not equal to the delay of a LOW-to-HIGH transition ($t_{PHL} \neq t_{PLH}$). Furthermore, the delay varies from one run to another. If the skew between t_{PHL} and t_{PLH} is 50 ps, for example, the total skew of the falling edges through ten stages will be 0.5 ns. Because of this skew, the NAND + inverter delay element cannot be used in a DDR DRAM. In our modified symmetrical delay element, another NAND gate is used instead of an inverter (two NAND gates per delay stage). This scheme guarantees that $t_{PHL} = t_{PLH}$ independent of process variations. While one NAND switches from HIGH to

LOW, the other switches from LOW to HIGH. An added benefit of the two-NAND delay element is that two point-of-entry control signals are now available. The shift register uses both to solve the possible problem caused by the POWERUP ambiguity in the shift register.

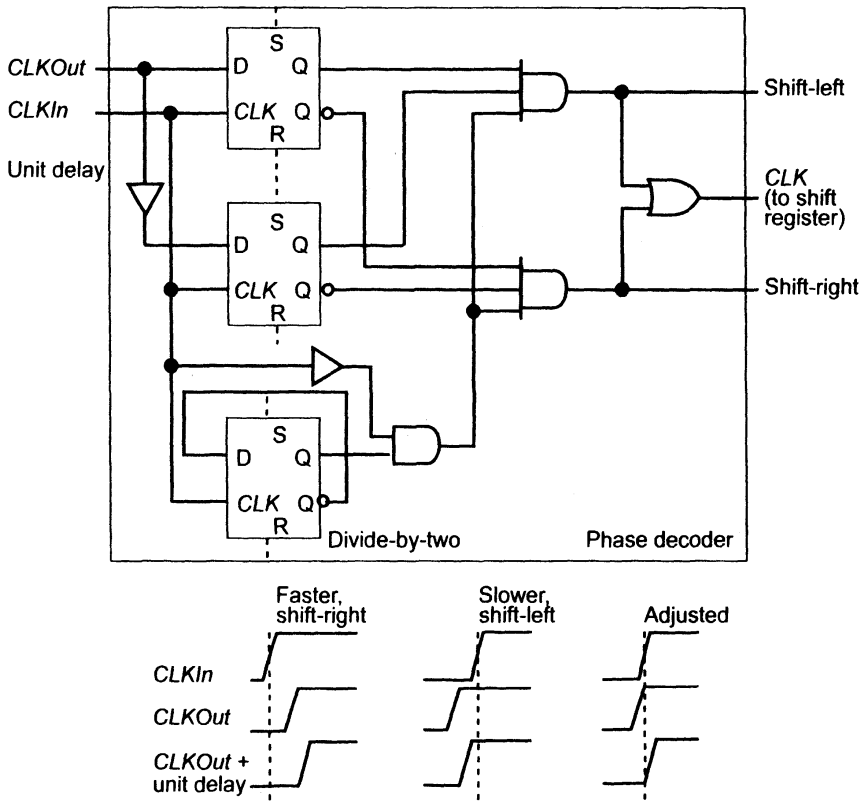


Figure 5.24 Phase detector used in RSDLL.

5.3.3 Control of the Shift Register

As shown in Figures 5.25 and 5.26, the input clock is a common input to every delay stage. The shift register selects a different *tap* of the delay line (the point of entry for the input clock signal into the symmetrical delay line). The complementary outputs of each register cell select the different tap: Q is connected directly to the input A of a delay element, and Q^* is connected to the previous stage of input B.

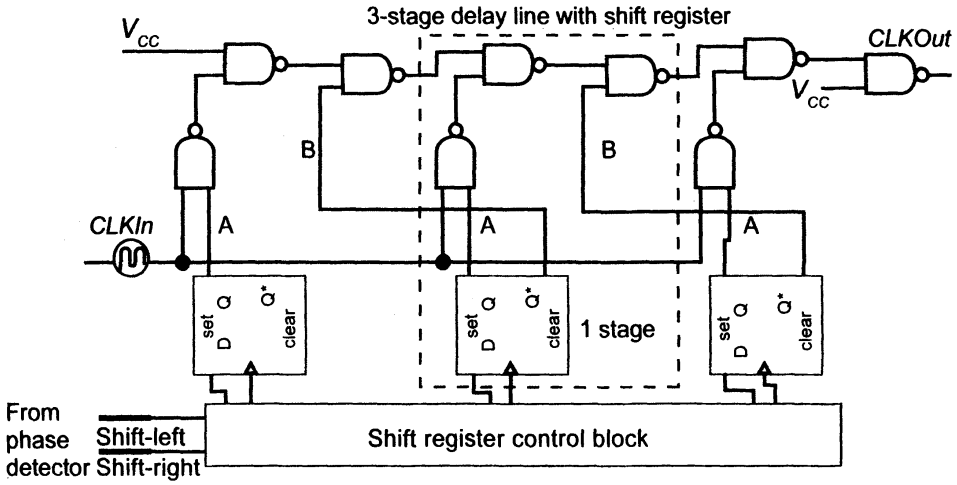


Figure 5.25 Symmetrical delay element used in RSDLL.

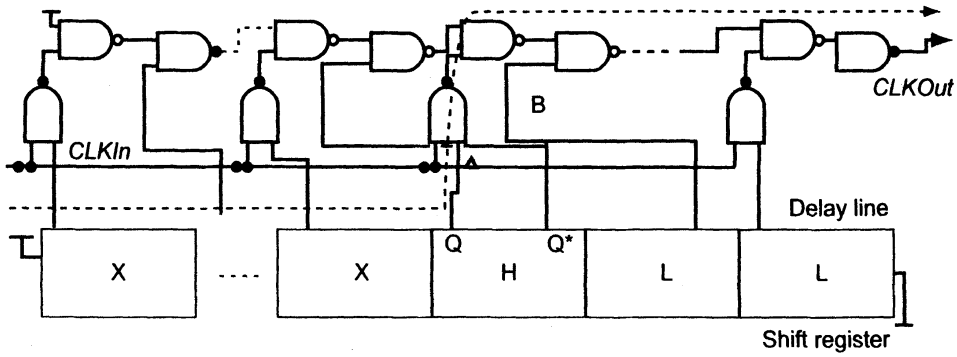


Figure 5.26 Delay line and shift register for RSDLL.

From right to left, the first LOW-to-HIGH transition in the shift register sets the point of entry into the delay line. The input clock passes through the tap with a HIGH logic state in the corresponding position of the shift register. Because the Q^* of this tap is equal to a LOW, it disables the previous stages; therefore, the previous states of the shift register do not matter (shown as "don't care," *X*, in Figure 5.25). This control mechanism guarantees that only one path is selected. This scheme also eliminates POWERUP concerns because the selected tap is simply the first, from the right, LOW-to-HIGH transition in the register.

5.3.4 Phase Detector Operation

To stabilize the movement in the shift register, after making a decision, the phase detector waits at least two clock cycles before making another decision (Figure 5.24). A divide-by-two is included in the phase detector so that every other decision, resulting from comparing the rising edges of the

external clock and the feedback clock, is used. This provides enough time for the shift register to operate and the output waveform to stabilize before another decision by the *PD* is implemented. The unwanted side effect of this delay is an increase in lock time. The shift register is clocked by combining the shift-left and -right signals. The power consumption decreases when there are no shift-left or -right signals and the loop is locked.

Another concern with the phase-detector design is the design of the flip-flops (FFs). To minimize the static phase error, very fast FFs should be used, ideally with zero setup time.

Also, the metastability of the flip-flops becomes a concern as the loop locks. This, together with possible noise contributions and the need to wait, as just discussed, before implementing a shift-right or -left, may make it more desirable to add more filtering in the phase detector. Some possibilities include increasing the divider ratio of the phase detector or using a shift register in the phase detector to determine when a number of—say, four—shift-rights or - lefts have occurred. For the design in Figure 5.26, a divide-by-two was used in the phase detector due to lock-time requirements.

5.3.5 Experimental Results

The RSDLL of Figure 5.22 was fabricated in a 0.21 μm, four-poly, double-metal CMOS technology (a DRAM process). A 48-stage delay line with an operation frequency of 125–250 MHz was used. The maximum operating frequency was limited by delays external to the DLL, such as the input buffer and interconnect. There was no noticeable static phase error on either the rising or falling edges. Figure 5.27 shows the resulting rms jitter versus input frequency. One sigma of jitter over the 125–250 MHz frequency range was below 100 ps. The measured delay per stage versus V_{CC} and temperature is shown in Figure 5.28. Note that the 150 ps typical delay of a unit-delay element was very close to the rise and fall times on-chip of the clock signals and represents a practical minimum resolution of a DLL for use in a DDR DRAM fabricated in a 0.21 μm process.

The power consumption (the current draw of the DLL when $V_{CC} = 2.8$ V) of the prototype RSDLL is illustrated in Figure 5.29. It was found that the power consumption was determined mainly by the dynamic power dissipation of the symmetrical delay line. The NAND delays in this test chip were implemented with 10/0.21 μm NMOS and 20/0.21 μm PMOS. By reducing the widths of both the NMOS and PMOS transistors, the power dissipation is greatly reduced without a speed or resolution penalty (with the added benefit of reduced layout size).

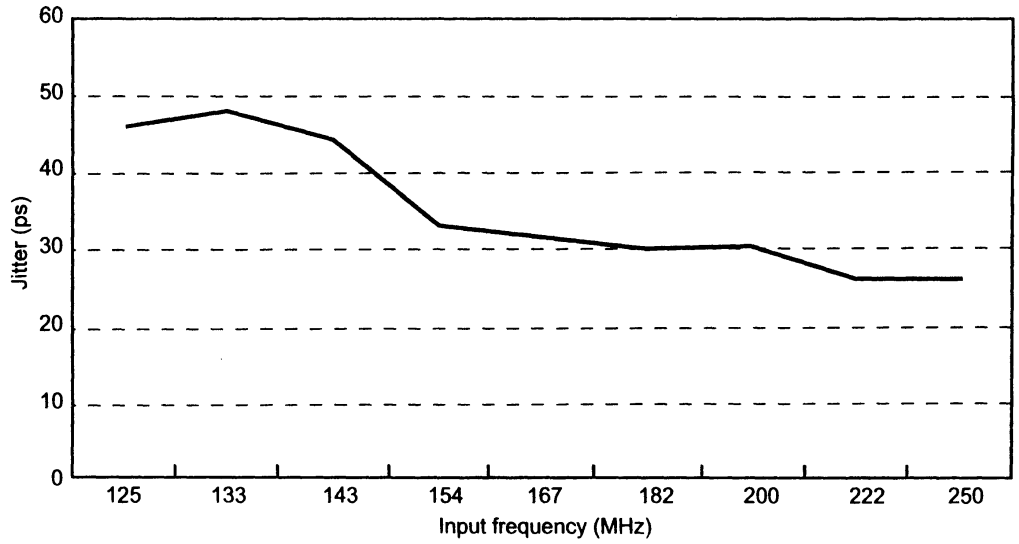


Figure 5.27 Measured rms jitter versus input frequency.

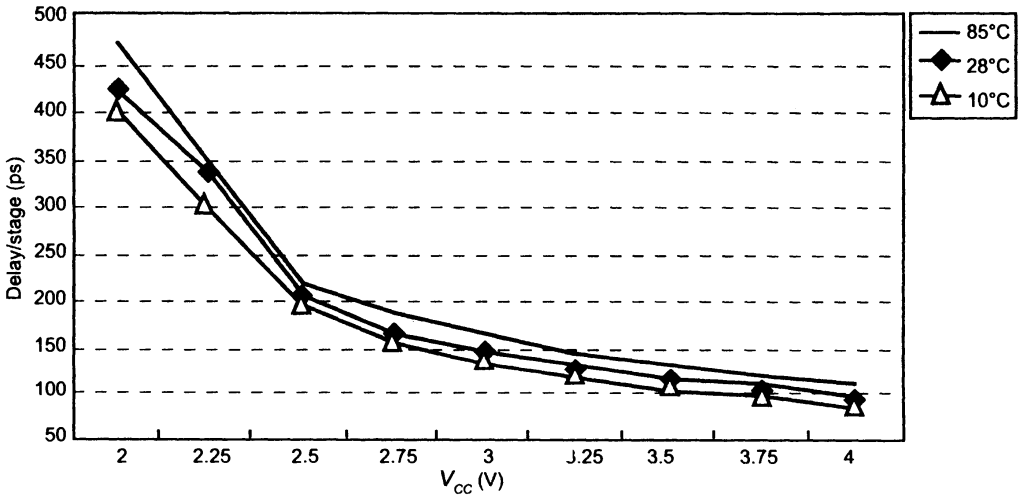


Figure 5.28 Measured delay per stage versus V_{CC} and temperature.

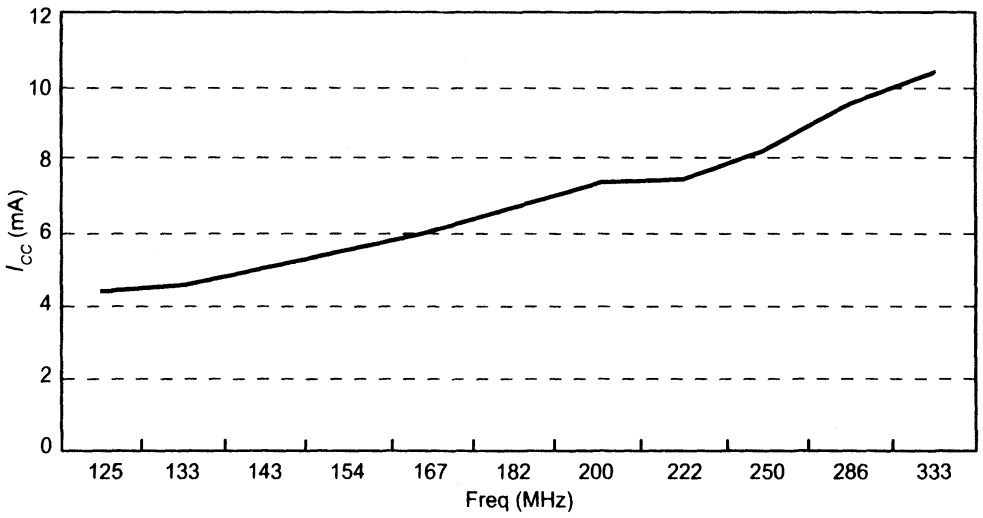


Figure 5.29 Measured *ICC* (DLL current consumption) versus input frequency.

5.3.6 Discussion

In this section we have presented one possibility for the design of a delay-locked loop. While there are others, this design is simple, manufacturable, and scalable.

In many situations the resolution of the phase detector must be decreased. A useful circuit to determine which one of two signals occurs earlier in time is shown in Figure 5.30. This circuit is called an arbiter. If *S*1 occurs slightly before *S*2 then the output *SO*1 will go HIGH, while the output *SO*2 stays LOW. If *S*2 occurs before *S*1, then the output *SO*2 goes HIGH and *SO*1 remains LOW. The fact that the inverters on the outputs are powered from the SR latch (the cross-coupled NAND gates) ensures that *SO*1 and *SO*2 cannot be HIGH at the same time. When designed and laid out correctly, this circuit is capable of discriminating tens of picoseconds of difference between the rising edges of the two input signals.

The arbiter alone cannot be capable of controlling the shift register. A simple logic block to generate shift-right and shift-left signals is shown in Figure 5.31. The rising edge of *SO*1 or *SO*2 is used to clock two D-latches so that the shift-right and shift-left signals may be held HIGH for more than one clock cycle. Figure 5.31 uses a divide-by-two to hold the shift signals valid for two clock cycles. This is important because the output of the arbiter can have glitches coming from the different times when the inputs go back LOW. Note that using an arbiter-based phase detector alone can result in an alternating sequence of shift-right, shift-left. We eliminated this problem in the phase-detector of Figure 5.24 by introducing the dead zone so that a minimum delay spacing of the clocks would result in no shifting.

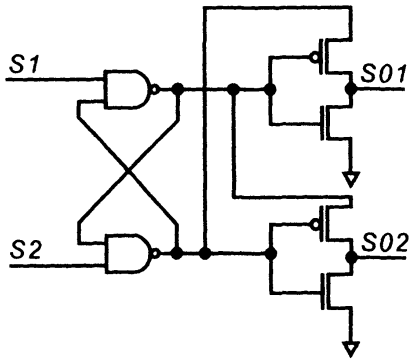


Figure 5.30 Two-way arbiter as a phase detector.

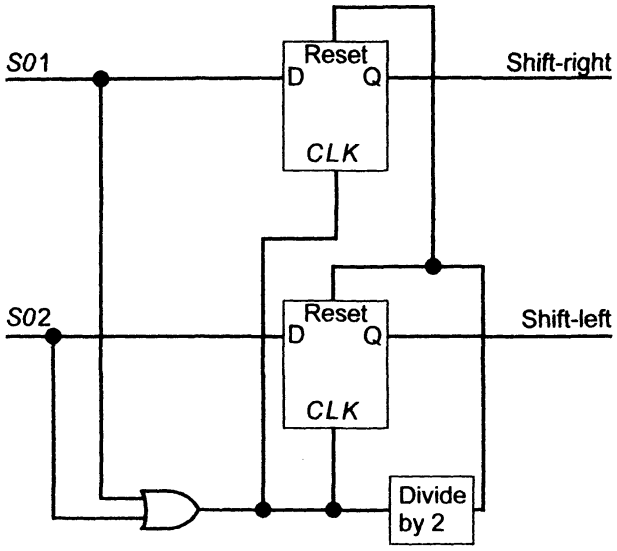


Figure 5.31 Circuit for generating shift register control.

In some situations, the fundamental delay time of an element in the delay line needs reduction. Using the NAND-based delay shown in Figure 5.26, we are limited to delay times much longer than a single inverter delay. However, using a single inverter delay results in an inversion in our basic cell. Figure 5.32 shows that using a double inverter-based delay element can solve this problem. By crisscrossing the single delay element inputs the cell appears to be non-inverting. This scheme results in the least delay because the delay between the cell's input and output is only the delay of a single inverter. The problems with using this type of cell over the NAND-based delay are inserting the feedback clock and the ability of the shift register to control the delay of the resulting delay line.

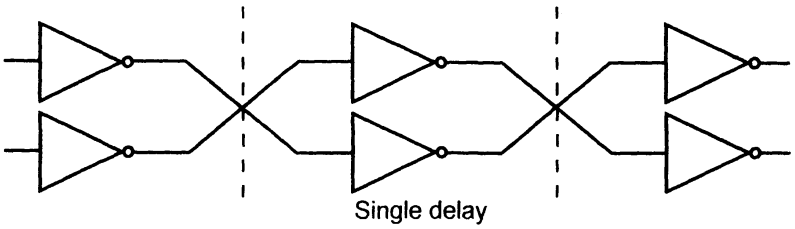


Figure 5.32 A double inverter used as a delay element.

Figure 5.33 shows how inserting transmission gates (TGs) that are controlled by the shift register allows the insertion point to vary along the line. When *C* is HIGH, the feedback clock is inserted into the output of the delay stage. The inverters in the stage are isolated from the feedback clock by an additional set of TGs. We might think, at first glance, that adding the TGs in Figure 5.33 would increase the delay significantly; however, there is only a single set of TGs in series with the feedback before the signal enters the line. The other TGs can be implemented as part of the inverter to minimize their impact on the overall cell delay. Figure 5.34 shows a possible inverter implementation.

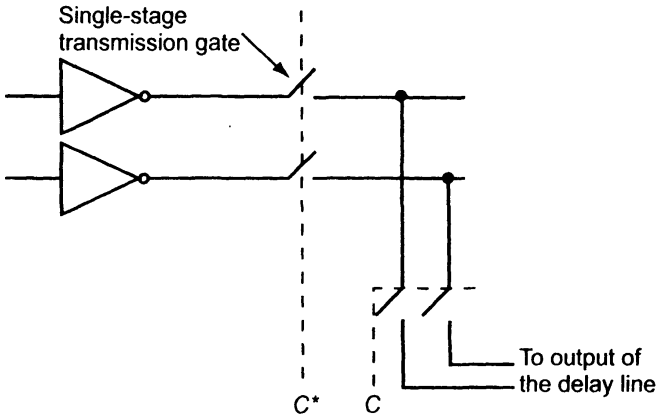


Figure 5.33 Transmission gates added to delay line.

Finally, in many situations other phases of an input clock need generating. This is especially useful in minimizing the requirements on the setup and hold times in a synchronous system. Figure 5.35 shows a method of segmenting the delays in a DLL delay line. A single control register can be used for all four delay segments. The challenge becomes reducing the amount of delay in a single delay element. A shift in this segmented configuration results in a change in the overall delay of four delays rather than a single delay. Note that with a little thought, and realizing that the delay elements of Figure 5.32 can be used in inverting or non-inverting configurations, the delay line of Figure 5.35 can be implemented with only two segments and still provide taps of 90°, 180°, 270°, and 360°.

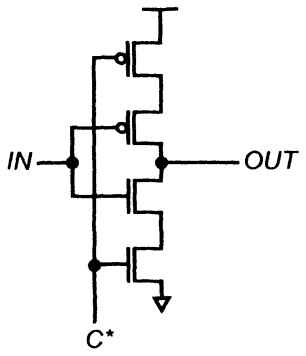


Figure 5.34 Inverter implementation.

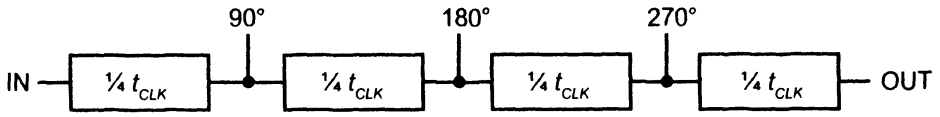


Figure 5.35 Segmenting delays for additional clocking taps.

REFERENCES

[1] H. Ikeda and H. Inukai, "High-speed DRAM architecture development," *IEEE Journal of Solid-State Circuits,* vol. 34, pp. 685–692, May 1999.

[2] M. Bazes, "Two novel full complementary self-biased CMOS differential amplifiers," *IEEE Journal of Solid-State Circuits,* vol. 26, pp. 165–168, February 1991.

[3] R. J. Baker, *CMOS: Circuit Design, Layout, and Simulation,* 2nd ed. Hoboken, NJ: IEEE Press, 2005.

[4] F. Lin, J. Miller, A. Schoenfeld, M. Ma, and R. J. Baker, "A register-controlled symmetrical DLL for double-data-rate DRAM," *IEEE Journal of Solid-State Circuits,* vol. 34, no. 4, 1999.

[5] A. Hatakeyama, H. Mochizuki, T. Aikawa, M. Takita, Y. Ishii, H. Tsuboi, S. Y. Fujioka, S. Yamaguchi, M. Koga, Y. Serizawa, K. Nishimura, K. Kawabata, Y. Okajima, M. Kawano, H. Kojima, K. Mizutani, T. Anezaki, M. Hasegawa, and M. Taguchi, "A 256-Mb SDRAM using a register-controlled digital DLL," *IEEE Journal of Solid-State Circuits,* vol. 32, pp. 1728–1732, November 1997.

[6] S. Eto, M. Matsumiya, M. Takita, Y. Ishii, T. Nakamura, K. Kawabata, H. Kano, A. Kitamoto, T. Ikeda, T. Koga, M. Higashiro, Y. Serizawa, K. Itabashi, O. Tsuboi, Y. Yokoyama, and M. Taguchi, "A 1Gb SDRAM with ground level precharged bitline and non-boosted 2.1V word line," in *ISSCC Digest of Technical Papers*, pp. 82–83, February 1998.

[7] T. Saeki, Y. Nakaoka, M. Fujita, A. Tanaka, K. Nagata, K. Sakakibara, T. Matano, Y. Hoshino, K. Miyano, S. Isa, S. Nakazawa, E. Kakehashi, J. M. Drynan, M. Komuro, T. Fukase, H. Iwasaki, M. Takenaka, J. Sekine, M. Igeta, N. Nakanishi, T. Itani, K. Yoshida, H. Yoshino, S. Hashimoto, T. Yoshii, M. Ichinose, T. Imura, M. Uziie, S. Kikuchi, K. Koyama, Y. Fukuzo, and T. Okuda, "A 2.5-ns clock access 250-MHz, 256-Mb SDRAM with synchronous mirror delay," *IEEE Journal of Solid-State Circuits,* vol. 31, pp. 1656–1665, November 1996.

Chapter
6

Voltage Converters

In this chapter, we discuss the circuitry for generating the on-chip voltages that lie outside the supply range. In particular, we look at the wordline pump voltage and the substrate pumps. We also discuss voltage regulators that generate the internal power supply voltages.

6.1 INTERNAL VOLTAGE REGULATORS

6.1.1 Voltage Converters

DRAMs depend on a variety of internally generated voltages to operate and to optimize their performance. These voltages generally include the boosted wordline voltage V_{CCP}, the internally regulated supply voltage V_{CC}, the $V_{CC}/2$ cellplate and digitline bias voltage $DVC2$, and the pumped substrate voltage V_{BB}. Each of these voltages is regulated with a different kind of voltage generator. A linear voltage converter generates V_{CC}, while a modified CMOS inverter creates $DVC2$. Generating the boosted supply voltages V_{CCP} and V_{BB} requires sophisticated circuits that employ charge or voltage pumps (a.k.a., charge pumps). In this chapter, we discuss each of these circuits and how they are used in modern DRAM designs to generate the required supply voltages.

Most modern DRAM designs rely on some form of internal voltage regulation to convert the external supply voltage V_{CCX} into an internal supply voltage V_{CC}. We say most, not all, because the need for internal regulation is dictated by the external voltage range and the process in which the DRAM is based. The process determines gate oxide thickness, field device characteristics, and diffused junction properties. Each of these properties, in turn, affects breakdown voltages and leakage parameters, which limit the

maximum operating voltage that the process can reliably tolerate. For example, a 1 Gb DRAM built in a 45 nm CMOS process with a 14Å thick gate oxide can operate reliably with an internal supply voltage not exceeding 1.2 V. If this design had to operate in a 3.3 V system, an internal voltage regulator would be needed to convert the external 3.3 V supply to an internal 1.2 V supply. For the same design operating in a 1.2 V system, an internal voltage regulator would not be required. Although the actual operating voltage is determined by process considerations and reliability studies, the internal supply voltage is generally proportional to the minimum feature size. Table 6.1 summarizes this relationship.

Table 6.1 DRAM process versus supply voltage.

Process	V_{CC} Internal
90 nm	1.5 V
72 nm	1.35 V
45 nm	1.2 V
32 nm	1.0 V

All DRAM voltage regulators are built from the same basic elements: a voltage reference, one or more output power stages, and some form of control circuit. How each of these elements is realized and combined into the overall design is the product of process and design limitations and the design engineer's preferences. In the paragraphs that follow, we discuss each element, overall design objectives, and one or more circuit implementations.

6.1.2 Voltage References

The sole purpose of a voltage reference circuit is to establish a nominal operating point for the voltage regulator circuit. However, this nominal voltage is not a constant; rather, it is a function of the external voltage V_{CCX} and temperature. Figure 6.1 is a graph of V_{CC} versus V_{CCX} for a typical DRAM voltage regulator. This figure shows three regions of operation. The first region occurs during a POWERUP or POWERDOWN cycle, in which V_{CCX} is below the desired V_{CC} operating voltage range. In this region, V_{CC} is set equal to V_{CCX}, providing the maximum operating voltage allowable in the part. A maximum voltage is desirable in this region to extend the

DRAM's operating range and ensure data retention during low-voltage conditions.

The second region exists whenever V_{CCX} is in the nominal operating range. In this range, V_{CC} flattens out and establishes a relatively constant supply voltage to the DRAM. Various manufacturers strive to make this region absolutely flat, eliminating any dependence on V_{CCX}. We have found, however, that a moderate amount of slope in this range for characterizing performance is advantageous. It is critically important in a manufacturing environment that each DRAM meet the advertised specifications, with some margin for error. A simple way to ensure these margins is to exceed the operating range by a fixed amount during component testing. The voltage slope depicted in Figure 6.1 allows this margin testing to occur by establishing a moderate degree of dependence between V_{CCX} and V_{CC}.

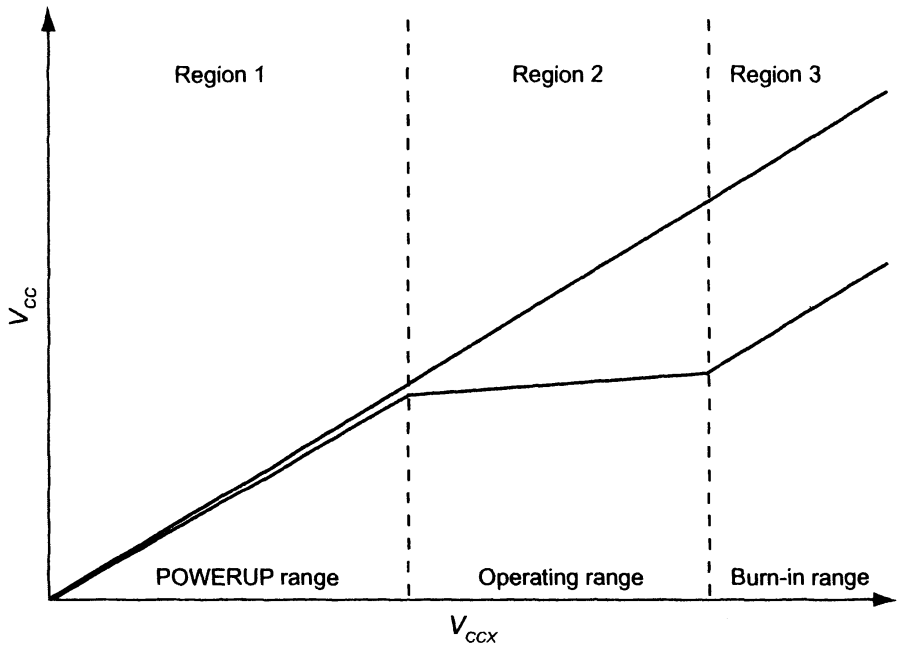


Figure 6.1 Ideal regulator characteristics.

The third region shown in Figure 6.1 is used for component burn-in. During burn-in, both the temperature and voltage are elevated above the normal operating range to stress the DRAMs and weed out infant failures. Again, if there were no V_{CCX} and V_{CC} dependency, the internal voltage could not be elevated. A variety of manufacturers do not use the monotonic curve shown in Figure 6.1. Some designs break the curve as shown in Figure 6.2, producing a step in the voltage characteristics. This step creates a region in which the DRAM cannot be operated. We will focus on the more desirable circuits that produce the curve shown in Figure 6.1.

To design a voltage reference, we need to make some assumptions about the power stages. First, we will assume that they are built as unbuf-

fered, two-stage, CMOS operational amplifiers and that the gain of the first stage is sufficiently large to regulate the output voltage to the desired accuracy. Second, we will assume that they have a closed loop gain of Av. The value of Av influences not only the reference design, but also the operating characteristics of the power stage (to be discussed shortly). For this design example, assume Av = 1.5. The voltage reference circuit shown in Figure 6.3 can realize the desired V_{CC} characteristics shown in Figure 6.4. This circuit uses a simple resistor and a PMOS diode reference stack that is buffered and amplified by an unbuffered CMOS op-amp. The resistor and diode are sized to provide the desired output voltage and temperature characteristics and the minimum bias current. Note that the diode stack is programmed through the series of PMOS switch transistors that are shunting the stack. A fuse element is connected to each PMOS switch gate. Unfortunately, this programmability is necessary to accommodate process variations and design changes.

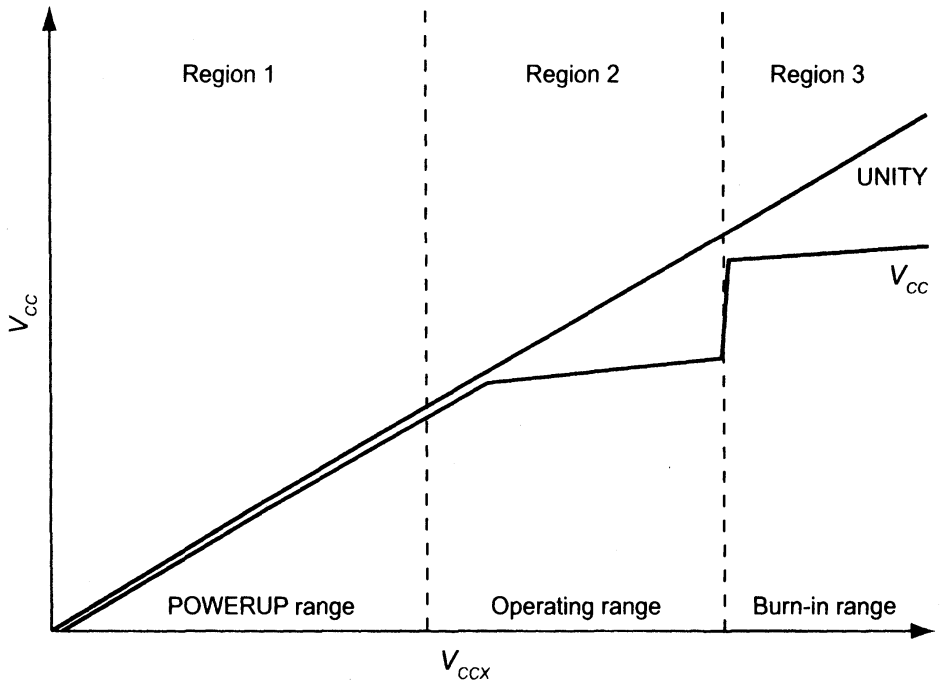


Figure 6.2 Alternative regulator characteristics.

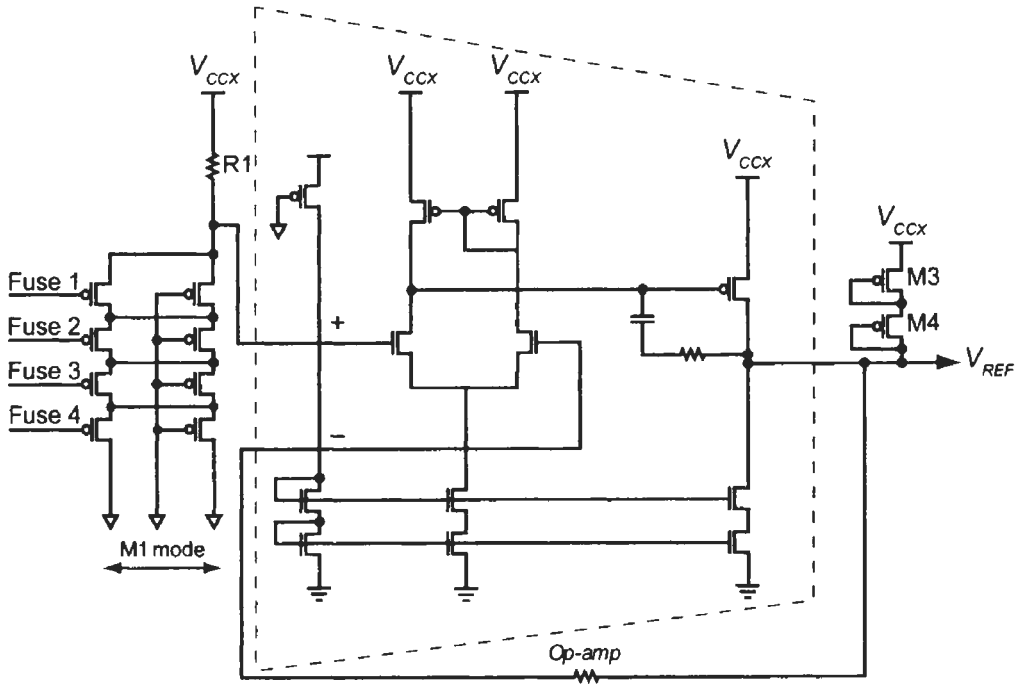


Figure 6.3 Resistor/diode voltage reference.

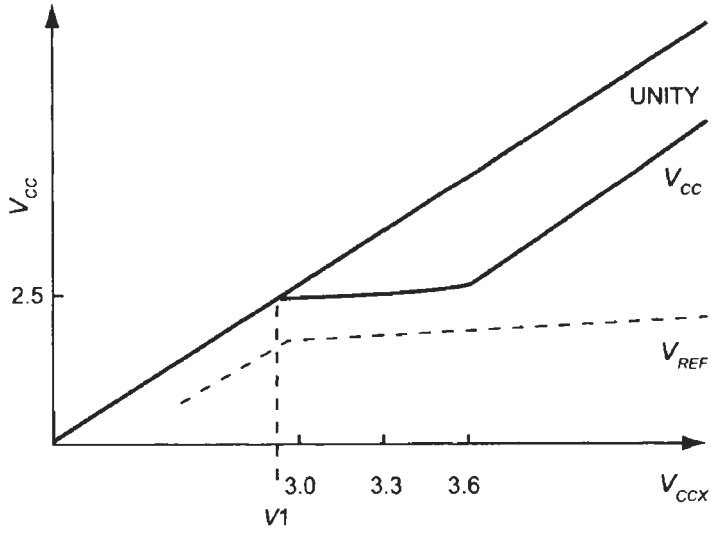


Figure 6.4 Voltage regulator characteristics.

The reference temperature characteristics rely on establishing a proper balance between V_{TH}, mobility, and resistance variations with temperature. An ideal temperature coefficient for this circuit would be positive such that the voltage rises with temperature, somewhat compensating for the CMOS gate delay speed loss associated with increasing temperature.

Two PMOS diodes (M3–M4) connected in series with the op-amp output terminal provide the necessary burn-in characteristics in Region 3. In normal operation, the diodes are OFF. As V_{CCX} is increased into the burn-in range, V_{CCX} will eventually exceed V_{REF} by two diode drops, turning ON the PMOS diodes and clamping V_{REF} to V_{TH} below V_{CCX}. The clamping action will establish the desired burn-in characteristics, keeping the regulator monotonic in nature.

In Region 2, voltage slope over the operating range is determined by the resistance ratio of the PMOS reference diode M1 and the bias resistor R1. Slope reduction is accomplished by either increasing the effective PMOS diode resistance or replacing the bias resistor with a more elaborate current source as shown in Figure 6.5. This current source is based on a V_{TH} referenced source to provide a reference current that is only slightly dependent on V_{CCX} [1]. A slight dependence is still necessary to generate the desired voltage slope.

The voltage reference does not actually generate Region 1 characteristics for the voltage regulator. Rather, the reference ensures a monotonic transition from Region 1 to Region 2. To accomplish this task, the reference must approximate the ideal characteristics for Region 1, in which $V_{CC} = V_{CCX}$. The regulator actually implements Region 1 by shorting the V_{CC} and V_{CCX} buses together through the PMOS output transistors found in each power stage op-amp. Whenever V_{CCX} is below a predetermined voltage V1, the PMOS gates are driven to ground, actively shorting the buses together. As V_{CCX} exceeds the voltage level V1, the PMOS gates are released and normal regulator operation commences. To ensure proper DRAM operation, this transition needs to be as seamless as possible.

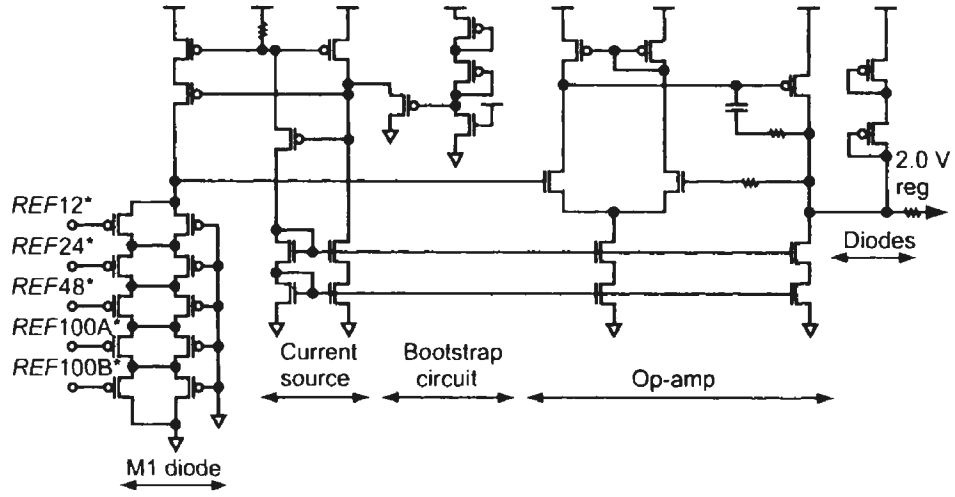


Figure 6.5 Improved voltage reference.

6.1.3 Bandgap Reference

Another type of voltage reference that is popular among DRAM manufacturers is the bandgap reference. The bandgap reference is traditionally built from vertical pnp transistors. A representative bandgap reference circuit is shown in Figure 6.6. As shown, it uses two bipolar transistors with an emitter size ratio of 10:1. Because they are both biased with the same current and owing to the different emitter sizes, a differential voltage will exist between the two transistors. The differential voltage appearing across resistor R1 will be amplified by the op-amp. Resistors R2 and R1 establish the closed loop gain for this amplifier and determine nominal output voltage and bias currents for the transistors [1].

The almost ideal temperature characteristics of a bandgap reference are what make them attractive to regulator designers. Through careful selection of emitter ratios and bias currents, the temperature coefficient can be set to approximately zero. Also, because the reference voltage is determined by the bandgap characteristics of silicon rather than a PMOS V_{TH}, this circuit is less sensitive to process variations than the circuit in Figure 6.3.

Three problems with the bandgap reference shown in Figure 6.6, however, make it much less suitable for DRAM applications. First, the bipolar transistors need moderate current to ensure that they operate beyond the knee on their I-V curves. This bias current is approximately 10–20 μA per transistor, which puts the total bias current for the circuit above 25 μA. The voltage reference shown in Figure 6.3, on the other hand, consumes less than 10 μA of the total bias current. Second, the vertical pnp transistors inject a significant amount of current into the substrate—as high as 50% of the total bias current in some cases. For a pumped substrate DRAM, the resulting charge from this injected current must be removed by the substrate pump, which raises standby current for the part. Third, the voltage slope for a bandgap reference is almost zero because the feedback configuration in Figure 6.6 has no dependence on V_{CCX}. Seemingly ideal, this performance is unacceptable because to perform margin testing on the DRAM a finite voltage slope is needed.

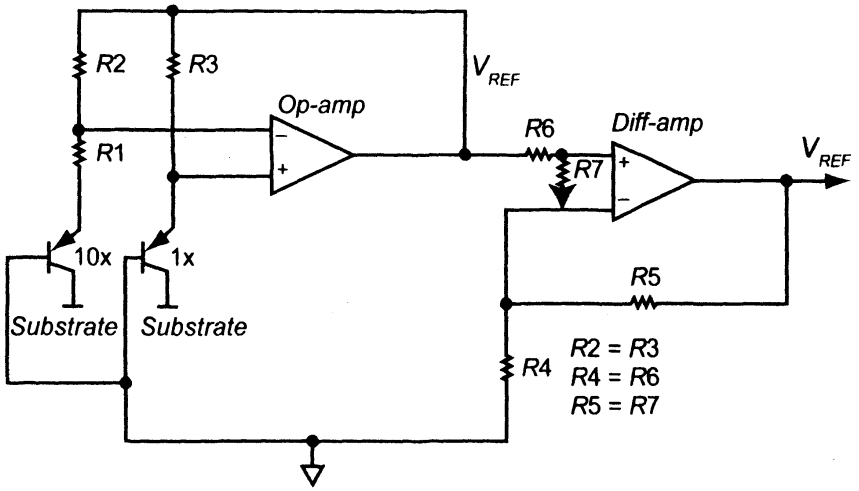


Figure 6.6 Bandgap reference circuit.

6.1.4 The Power Stage

Although static voltage characteristics of the DRAM regulator are determined by the voltage reference circuit, dynamic voltage characteristics are dictated by the power stages. The power stage is therefore a critical element in overall DRAM performance. To date, the most prevalent type of power stage among DRAM designers is a simple, unbuffered op-amp. Unbuffered op-amps, while providing high open loop gain, fast response, and low offset, allow design engineers to use feedback in the overall regulator design. Feedback reduces temperature and process sensitivity and ensures better load regulation than any type of open loop system. Design of the op-amps, however, is anything but simple.

The ideal power stage would have high bandwidth, high open-loop gain, high slew rate, low systematic offset, low operating current, high drive, and inherent stability. Unfortunately, several of these parameters are contradictory, which compromises certain aspects of the design and necessitates trade-offs. While it seems that many DRAM manufacturers use a single op-amp for the regulator's power stage, we have found that it is better to use a multitude of smaller op-amps. These smaller op-amps have wider bandwidth, greater design flexibility, and an easier layout than a single, large op-amp.

The power op-amp is shown in Figure 6.7. The schematic diagram for a voltage regulator power stage is shown in Figure 6.8. This design is used on a 256Mb DRAM and consists of 18 power op-amps, one boost amp, and one small standby op-amp. The V_{CC} power buses for the array and peripheral circuits are isolated except for the 20-ohm resistor that bridges the two together. Isolating the buses is important to prevent high-current spikes that occur in the array circuits from affecting the peripheral circuits. Failure to

isolate these buses can result in speed degradation for the DRAM because high-current spikes in the array cause voltage cratering and a corresponding slowdown in logic transitions.

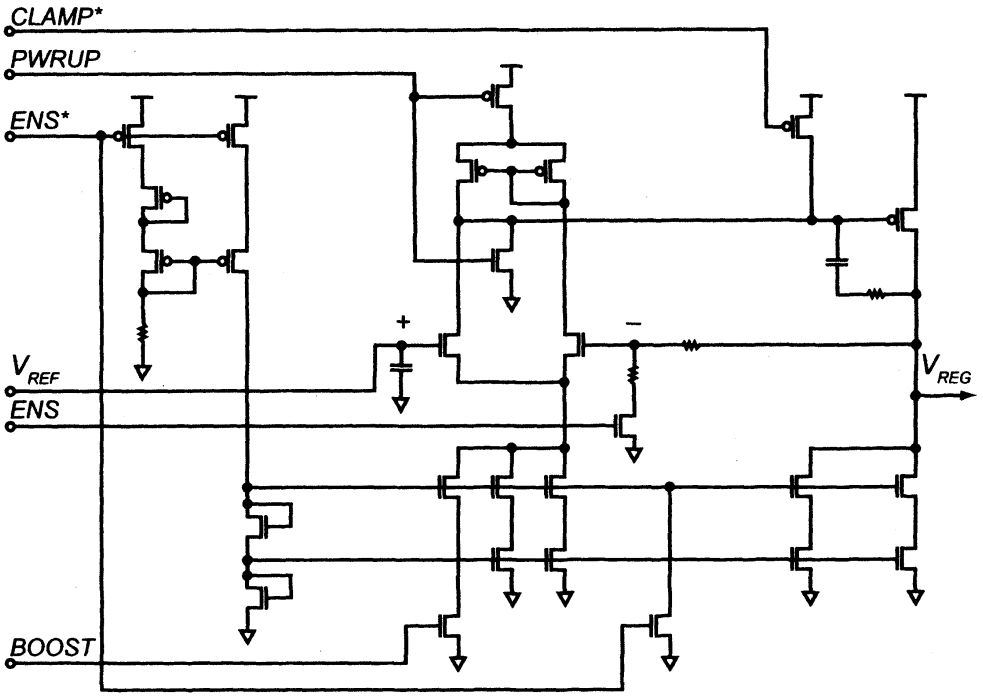


Figure 6.7 Power op-amp.

With isolation, the peripheral V_{CC} is almost immune to array noise. To improve slew rate, each of the power op-amp stages shown in Figure 6.8 features a boost circuit that raises the differential pair bias current and slew rate during expected periods of large current spikes. Large spikes are normally associated with Psense-amp activation. To reduce active current consumption, the boost current is disabled a short time after Psense-amp activation by the signal labeled *BOOST*. The power stages themselves are enabled by the signals labeled *ENS* and *ENS** only when *RAS** is LOW and the part is active. When *RAS** is HIGH, all of the power stages are disabled.

The signal labeled *CLAMP** ensures that the PMOS output transistor is OFF whenever the amplifier is disabled to prevent unwanted charging of the V_{CC} bus. When forced to ground, however, the signal labeled *PWRUP* shorts the V_{CCX} and V_{CC} buses together through the PMOS output transistor. (The need for this function was described earlier in this section.) Basically, these two buses are shorted together whenever the DRAM operates in Region 1, as depicted in Figure 6.1. Obviously, to prevent a short circuit between V_{CCX} and ground, *CLAMP** and *PWRUP* are mutually exclusive.

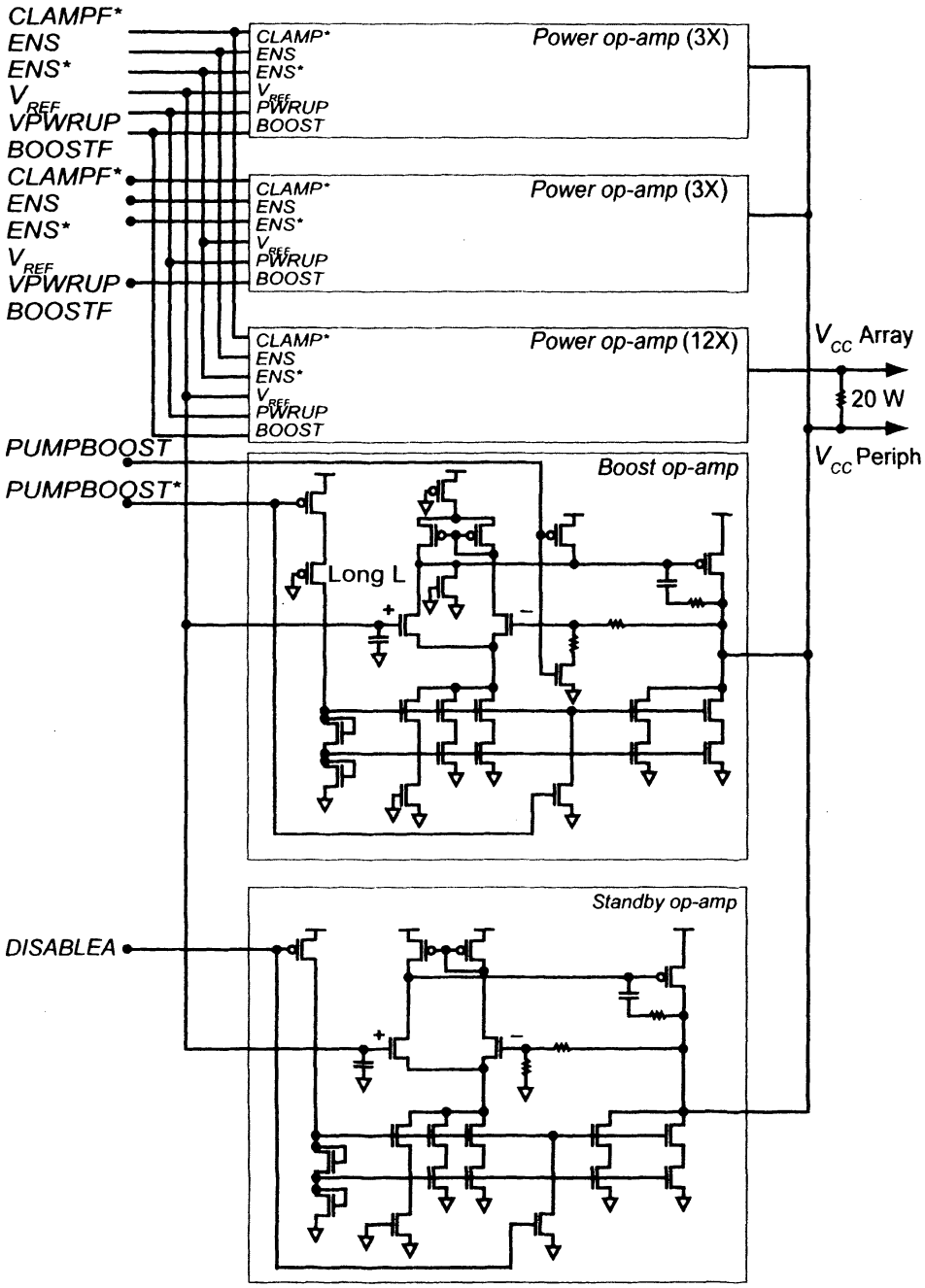


Figure 6.8 Power stage.

The smaller standby amplifier is included in this design to sustain the V_{CC} supply whenever the part is inactive, as determined by *RAS**. This amplifier has a very low operating current and a correspondingly low slew rate. Accordingly, the standby amplifier cannot sustain any type of active load. For this reason, the third and final type of amplifier is included. The boost amplifier shown in Figure 6.8 is identical to the power stage amplifi-

ers except that the bias current is greatly reduced. This amplifier provides the necessary supply current to operate the V_{CCP} and V_{BB} voltage pumps.

The final element in the voltage regulator is the control logic. An example of this logic is shown in Figure 6.9. It consists primarily of static CMOS logic gates and level translators. The logic gates are referenced to V_{CC}. Level translators are necessary to drive the power stages, which are referenced to V_{CCX} levels. A series of delay elements tune the control circuit relative to *Psense-amp activation (ACT)* and *RAS* (RL*)* timing. Included in the control circuit is the block labeled V_{CCX} *level detector* [1]. The *reference generator* generates two reference signals, which are fed into the comparator, to determine the transition point V1 between Region 1 and Region 2 operation for the regulator. In addition, the *boost amp control* logic block is shown in Figure 6.9. This circuit examines the V_{BB} and V_{CCP} control signals to enable the boost amplifier whenever either voltage pump is active.

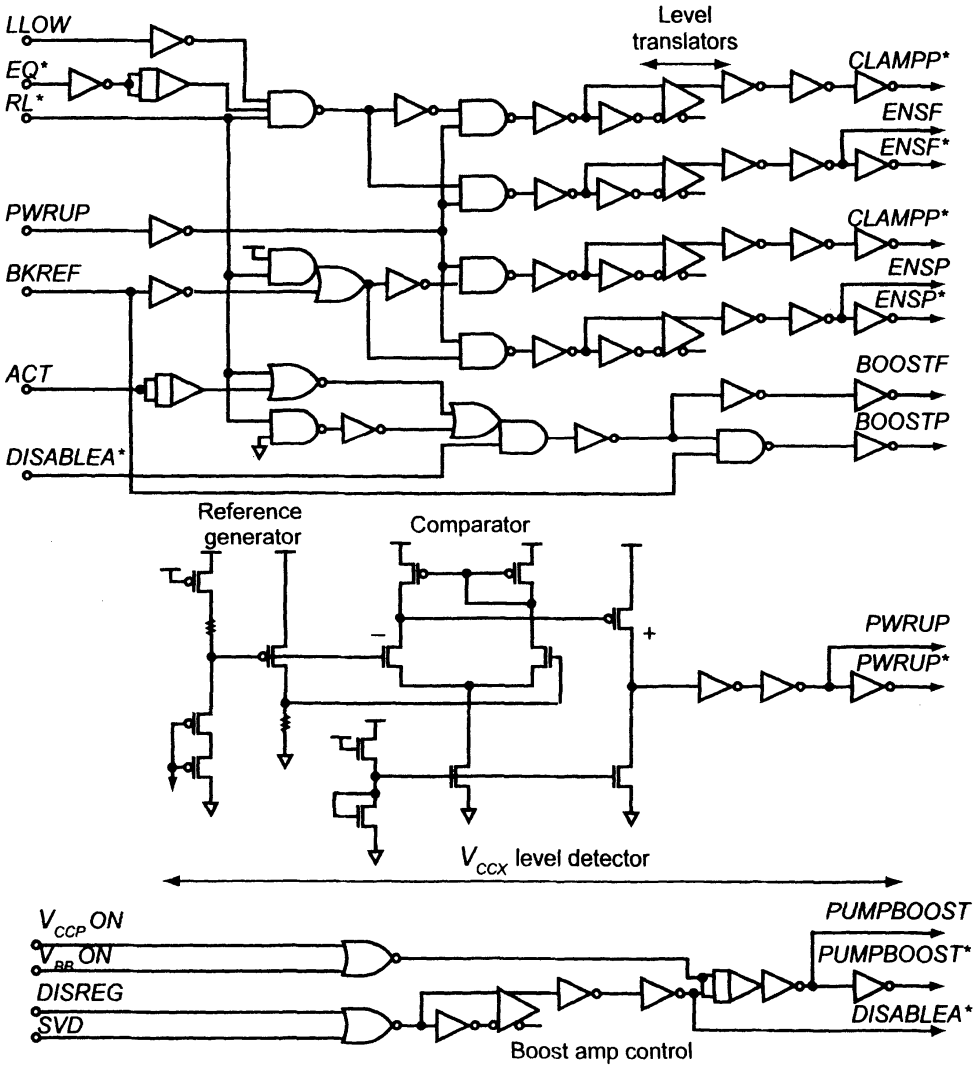


Figure 6.9 Regulator control logic.

6.2 PUMPS AND GENERATORS

Generation of the boosted wordline voltage and negative substrate bias voltage requires voltage pump (or charge pump) circuits. Pump circuits are commonly used to create voltages that are more positive or negative than available supply voltages. Two voltage pumps are commonly used in a DRAM today. The first is a V_{CCP} pump, which generates the boosted wordline voltage and is built primarily from NMOS transistors. The second is a V_{BB} pump, which generates the negative substrate bias voltage and is built from PMOS transistors. The exclusive use of NMOS or PMOS transistors in each pump is required to prevent latch-up and current injection into the Mbit arrays. NMOS transistors are required in the V_{CCP} pump because various active nodes would swing negative with respect to the substrate voltage V_{CCP}. Any n-diffusion regions connected to these active nodes would forward bias and cause latch-up and injection. Similar conditions mandate the use of PMOS transistors in the V_{BB} pump.

6.2.1 Pumps

Voltage pump operation can be understood with the assistance of the simple voltage pump circuit depicted in Figure 6.10. For this positive pump circuit, imagine, for one phase of a pump cycle, that the clock *CLK* is HIGH. During this phase, node A is at ground and node B is clamped to $V_{CC}-V_{TH}$ by transistor M1. The charge stored in capacitor C1 is then

$$Q1 = C1 \cdot (V_{CC} - V_{TH}) \text{ coulombs} \tag{6.1}$$

During the second phase, the clock *CLK* will transition LOW, which brings node A HIGH. As node A rises to V_{CC}, node B begins to rise to $V_{CC} + (V_{CC}-V_{TH})$, shutting OFF transistor M1. At the same time, as node B rises one V_{TH} above V_{LOAD}, transistor M2 begins to conduct. The charge from capacitor C1 is transferred through M2 and shared with the capacitor C_{LOAD}. This action effectively pumps charge into C_{LOAD} and ultimately raises the voltage V_{OUT}. During subsequent clock cycles, the voltage pump continues to deliver charge to C_{LOAD} until the voltage V_{OUT} equals $2V_{CC}-V_{TH1}-V_{TH2}$, one V_{TH} below the peak voltage occurring at node B. A simple, negative voltage pump could be built from the circuit of Figure 6.10 by substituting PMOS transistors for the two NMOS transistors shown and moving their respective gate connections.

Schematics for actual V_{CCP} and V_{BB} pumps are shown in Figures 6.11 and 6.12, respectively. Both of these circuits are identical except for the changes associated with the NMOS and PMOS transistors. These pump cir-

cuits operate as two phase pumps because two identical pumps are working in tandem. As discussed in the previous paragraph, note that transistors M1 and M2 are configured as switches rather than as diodes. The drive signals for these gates are derived from secondary pump stages and the tandem pump circuit. Using switches rather than diodes improves pumping efficiency and operating range by eliminating the V_{TH} drops associated with diodes.

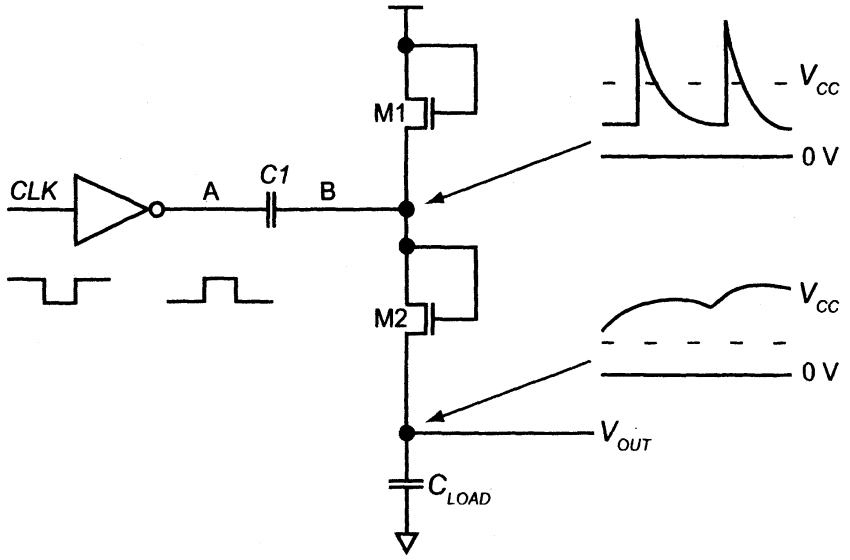


Figure 6.10 Simple voltage pump circuit.

Two important characteristics of a voltage pump are capacity and efficiency. Capacity is a measure of how much current a pump can continue to supply, and it is determined primarily by the capacitor's size and its operating frequency. The operating frequency is limited by the rate at which the pump capacitor C1 can be charged and discharged. Efficiency, on the other hand, is a measure of how much charge or current is wasted during each pump cycle. A typical DRAM voltage pump might be 30–50% efficient. This translates into 2–3 milliamps of supply current for every milliamp of pump output current.

In addition to the pump circuits just described, regulator and oscillator circuits are needed to complete a voltage pump design. The most common oscillator used in voltage pumps is the standard CMOS ring oscillator. A typical voltage pump ring oscillator is shown in Figure 6.13. A unique feature of this oscillator is the multifrequency operation permitted by including mux circuits connected to various oscillator tap points. These muxes, controlled by signals such as *PWRUP*, enable higher frequency operation by reducing the number of inverter stages in the ring oscillator.

Typically, the oscillator is operated at a higher frequency when the DRAM is in a *PWRUP* state, since this will assist the pump in initially

charging the load capacitors. The oscillator is enabled and disabled through the signal labeled *REGDIS**. This signal is controlled by the voltage regulator circuit shown in Figure 6.14. Whenever *REGDIS** is HIGH, the oscillator is functional, and the pump is operative. Examples of V_{CCP} and V_{BB} pump regulators are shown in Figure 6.14 and 6.15, respectively.

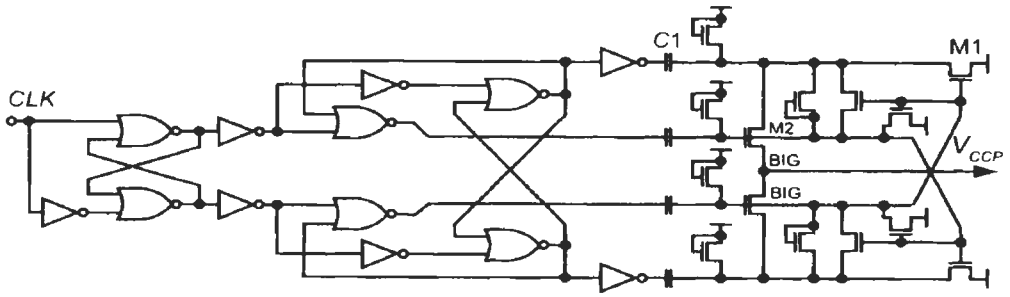


Figure 6.11 V_{CCP} pump.

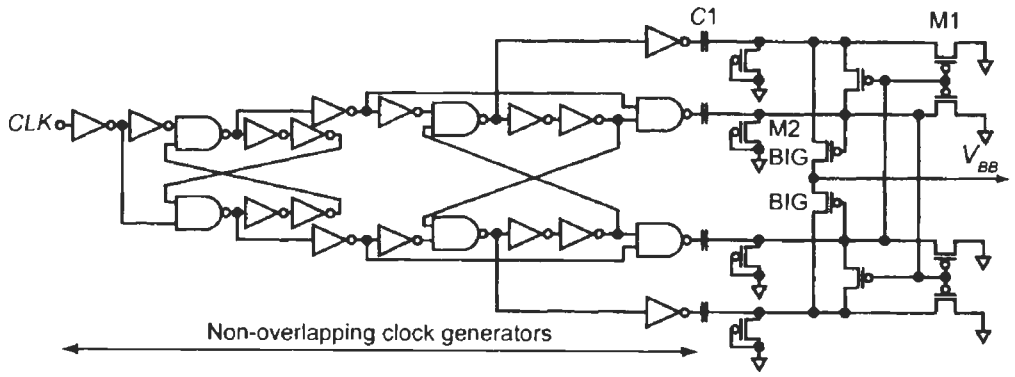


Figure 6.12 V_{BB} pump.

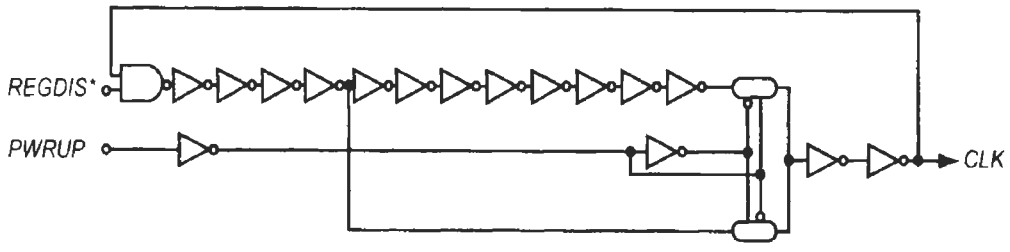


Figure 6.13 Ring oscillator

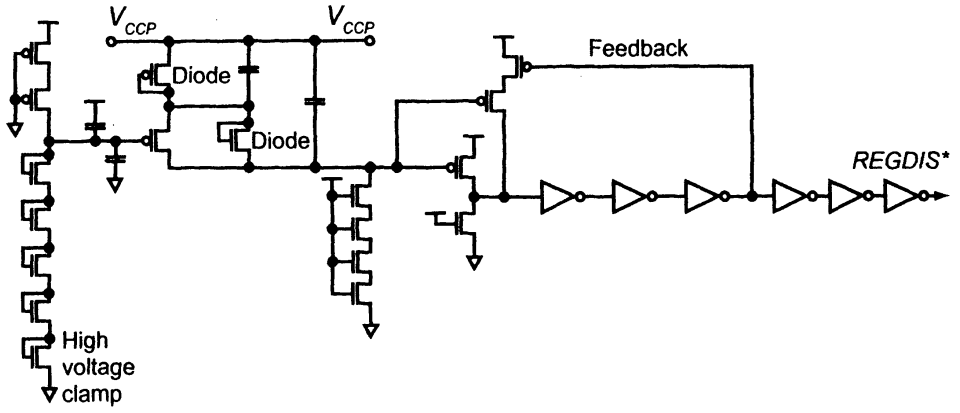


Figure 6.14 V_{CCP} regulator.

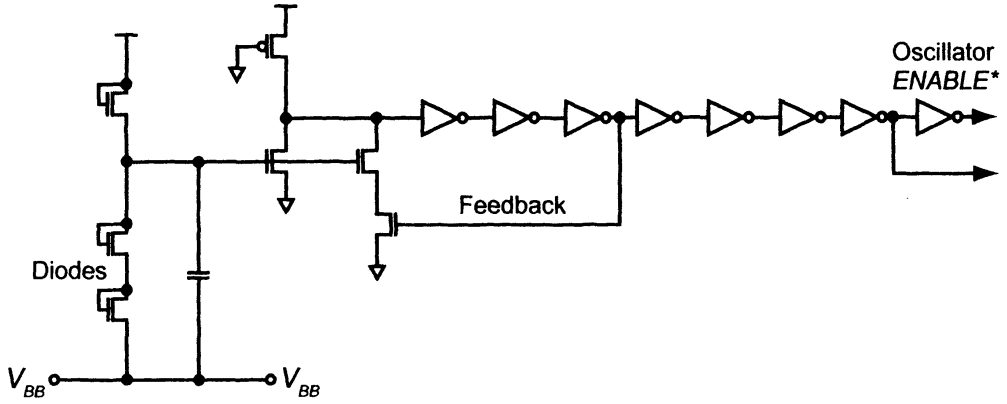


Figure 6.15 V_{BB} regulator.

These circuits use low-bias current sources and NMOS/PMOS diodes to translate the pumped voltage levels, V_{CCP} and V_{BB}, to normal voltage levels. This shifted voltage level is fed into a modified inverter stage that has an output-dependent trip point (the inverter has hysteresis). The trip point is modified with feedback to provide hysteresis for the circuit. Subsequent inverter stages provide additional gain for the regulator and boost the signal to the full CMOS level necessary to drive the oscillator. Minimum and maximum operating voltages for the pump are controlled by the first inverter stage trip point, hysteresis, and the NMOS/PMOS diode voltages. The clamp circuit shown in Figure 6.14 is included in the regulator design to limit the pump voltage when V_{CC} is elevated, such as during burn-in.

A second style of voltage pump regulator for controlling V_{CCP} and V_{BB} voltages is shown in Figure 6.16 and 6.17, respectively. This type of regulator uses a high-gain comparator coupled to a voltage translator stage. The translator stage of Figure 6.16 translates the pumped voltage V_{CCP} and the reference voltage V_{DD} down within the input common-mode range of the comparator circuit. The translator accomplishes this with a reference current

source and MOS diodes. The reference voltage supply V_{DD} is translated down by one threshold voltage (V_{TH}) by sinking reference current from a current mirror through a PMOS diode. The pumped voltage supply V_{CCP} is similarly translated down by sinking the same reference current with a matching current mirror through a diode stack. The diode stack consists of a PMOS diode, matching that in the V_{DD} reference translator, and a pseudo-NMOS diode. The pseudo-NMOS diode is actually a series of NMOS transistors with a common gate connection.

Transistor quantity and sizes included in the pseudo-NMOS diode are mask-programmable. The voltage drop across this pseudo-NMOS diode determines the regulated voltage for V_{CCP}. Accordingly,

$$V_{CCP} = V_{DD} + Vndiode \tag{6.2}$$

The voltage dropped across the PMOS diode does not affect the regulated voltage because the reference voltage supply V_{DD} is translated through a matching PMOS diode. Both translated voltages are fed into a comparator stage, enabling the pump oscillator whenever the translated V_{CCP} voltage falls below the translated V_{DD} reference voltage. The comparator hysteresis dictates the amount of ripple present on the regulated V_{CCP} supply.

The V_{BB} regulator in Figure 6.17 operates in a similar fashion to the V_{CCP} regulator of Figure 6.16. The primary difference lies in the voltage translator stage. For the V_{BB} regulator, this stage translates the pumped voltage V_{BB} and the reference voltage V_{SS} up within the input common mode range of the comparator. The reference voltage V_{SS} is translated up by one threshold voltage (V_{TH}) by sourcing a reference current from a current mirror through an NMOS diode. The regulated voltage V_{BB} is similarly translated up by sourcing the same reference current with a matching current mirror through a diode stack. This diode stack contains an NMOS diode that matches that used in translating the reference voltage V_{SS}. The stack also contains a mask-adjustable, pseudo-NMOS diode. The voltage across the pseudo-NMOS diode determines the regulated voltage for V_{BB} such that

$$V_{BB} = -Vndiode \tag{6.3}$$

The amount of ripple present on the regulated *VBB* supply is dictated by comparator hysteresis.

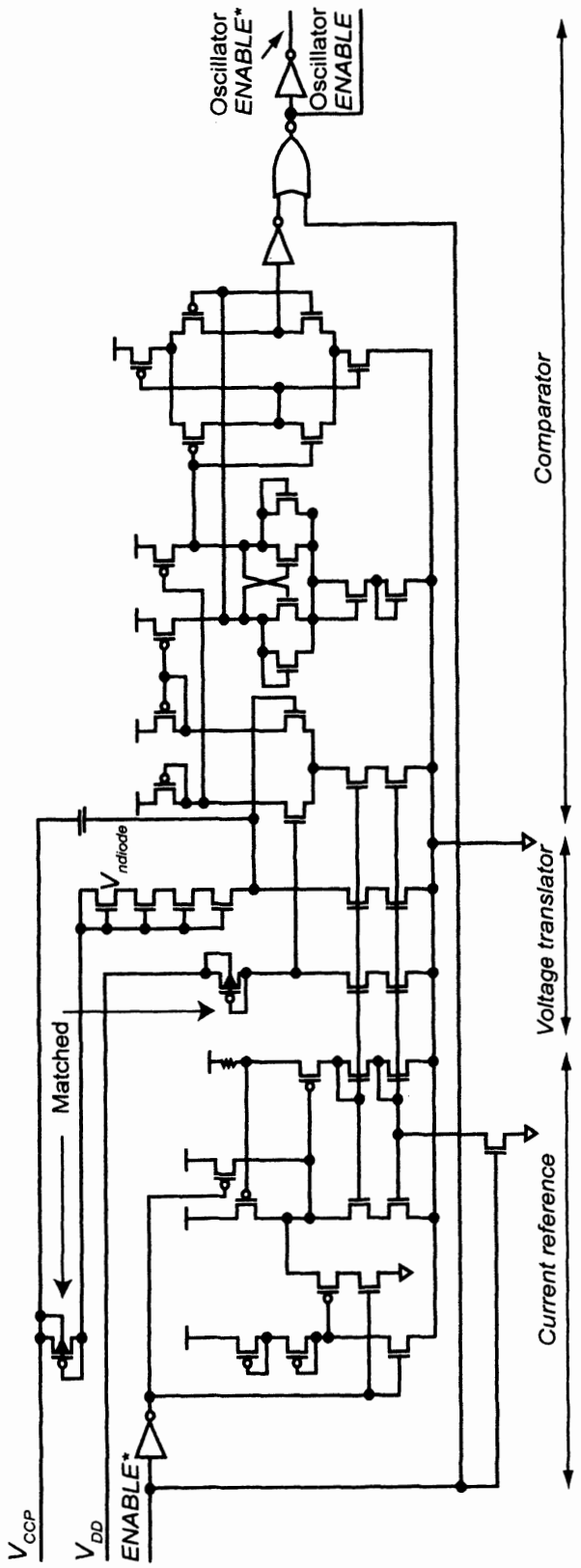


Figure 6.16 V_{CCP} differential regulator.

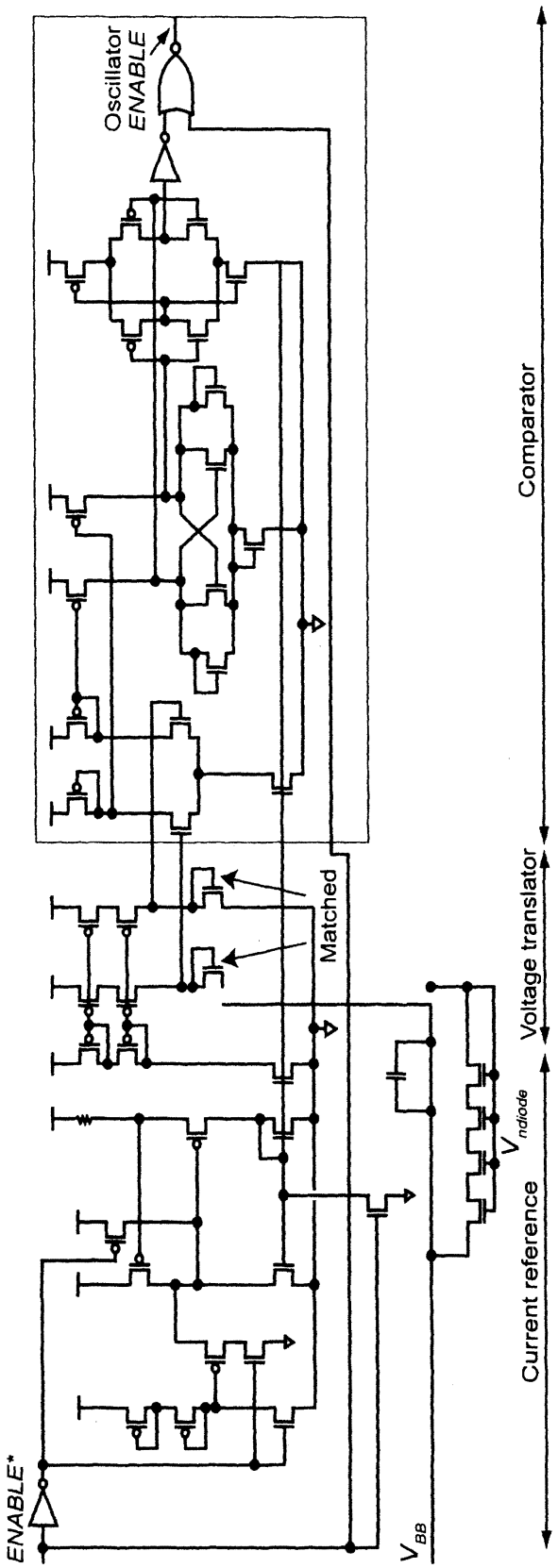


Figure 6.17 V_{BB} differential regulator.

6.2.2 *DVC2* Generator

With our discussion of voltage regulator circuits concluded, we can briefly turn our attention to the *DVC2* generation. As discussed in Section 1.2, the memory capacitor cellplate is biased to $V_{CC}/2$. Furthermore, the digitlines are always equilibrated and biased to $V_{CC}/2$ between array accesses. In most DRAM designs, the cellplate and digitline bias voltages are derived from the same generator circuit. A simple circuit for generating $V_{CC}/2$ (*DVC2*) voltage is shown in Figure 6.18. It consists of a standard CMOS inverter with the input and output terminals shorted together. With correct transistor sizing, the output voltage of this circuit can be set precisely to $V_{CC}/2$ V. Taking this simple design one step further results in the actual DRAM *DVC2* generator found in Figure 6.19. This generator contains additional transistors to improve both its stability and drive capability.

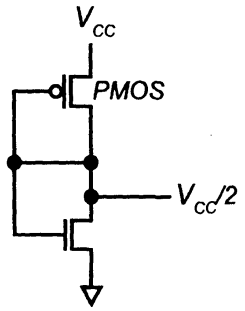


Figure 6.18 Simple *DVC2* generator.

6.3 DISCUSSION

In this chapter, we introduced the popular circuits used on a DRAM for voltage generation and regulation. Because this introduction is far from exhaustive, we include a list of relevant readings and references in the Appendix for those readers interested in greater detail.

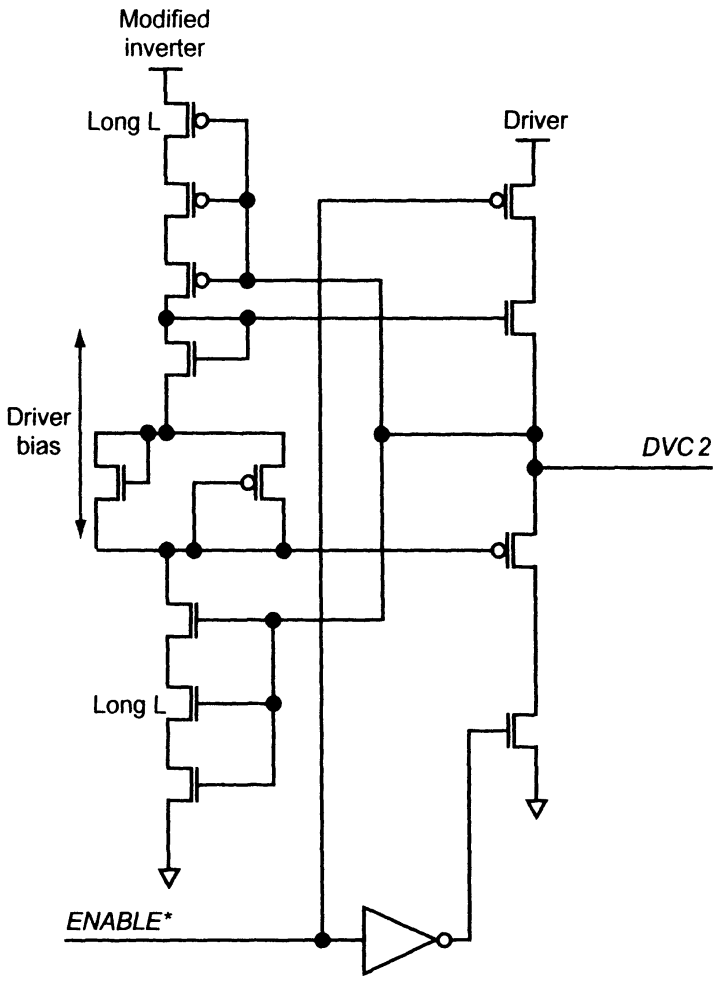


Figure 6.19 *DVC2* generator.

REFERENCES

[1] R. Baker, *CMOS: Circuit Design, Layout, and Simulation,* 2nd ed. Hoboken, NJ: IEEE Press, 2005.

[2] B. Keeth. 1994. Control circuit responsive to its supply voltage level. US Patent 5,373,227, issued December 13, 1994.

Chapter
7

An Introduction to High-Speed DRAM

Historically, the semiconductor industry has been the slave to two masters: cost and performance. Cost, or more specifically cost reduction, is a matter of pure economic survival for most companies in this industry. It is cost after all that drives the industry to aggressively shrink process technology year after year. Memory chips, due to their status as a commodity product, are highly sensitive to cost. This fact fosters additional incentives for memory companies to push their process technology development faster than most other segments of the semiconductor industry. The historical record is littered with the names of companies that produced memory products at one time or another, but subsequently dropped out of the marketplace. A key contributor to this attrition is the ability or inability of any given company to remain cost competitive. Market forces, not supplier controls, determine the basic selling price for commodity memory—most notably, the balance or imbalance in supply and demand. This leaves cost as the only component of a balance sheet that can be controlled. Control costs and survive. Lose control and fade away.

7.1 THE PERFORMANCE PARADIGM

While a complete analysis of the cost issues facing memory companies would be interesting, the remainder of this book deals with a topic more fascinating and dear to engineering professionals: performance. The word *performance*, much like the words *quality* or *value*, is rather ambiguous. Yet, when we talk about performance, most people assume we are referring to *high* performance. This is a natural and appropriate assumption, as we rarely strive for low performance, let alone write books about it. Just to set

the record straight, the remaining chapters deal entirely with high-performance DRAM.

Now, DRAM may not be what first comes to mind when you hear the phrase *high performance*. We agree with that sentiment as images of race cars and fighter aircraft flash through our minds. But, when we come back to reality and refocus our attention on the semiconductor industry, DRAM memory devices still don't come to mind. Rather, we are more likely to consider microprocessors. Clearly, these devices are on the leading edge of high performance, and they benefit from the use of the latest transistor, metal, and packaging technology in the quest for speed and processing muscle.

Memory, on the other hand, tends to limp along with slower transistors, a minimum of metal layers, and inexpensive packaging. The goal at DRAM companies has always been to meet the necessary speed targets while keeping manufacturing costs to an absolute minimum. This doesn't mean that the DRAM processes are not highly advanced. They clearly *are* advanced. The real difference between microprocessor and DRAM process technology has been the focus of process development. While the focus for microprocessors has been performance, the focus in the memory sector has always been cost. For DRAM, this focus has become a myopic quest for smaller and smaller process geometries, with the ultimate goal of yielding more die per wafer and, hence, at a lower cost than the competition.

Only recently has the need for higher performance DRAM become a pressing concern. There are two sectors of the memory industry that are driving this issue. The first is high-performance desktop systems; the second is high-end graphics cards. Both of these sectors are extremely competitive, and they are highly prized for the pricing margins and marketing mileage available to the best in class. With processor speeds racing into the multi-gigahertz range there are greater incentives for the memory subsystem to try and keep pace. Pricing pressures are lower in this sector, which gives DRAM manufacturers more latitude with their process technology, design architectures, and packaging in order to realize desired performance gains. These performance gains and the underlying technology eventually trickle down, over time, to the higher volume desktop market—increasing performance levels for the mainstream market, but ultimately raising the bar again in the high-performance desktop market.

DRAM devices developed for the graphics card markets follow a similar path, except at a more accelerated pace. High-end graphics cards create a need for very high-performance DRAM devices. Given lower pricing pressure, DRAM manufacturers can improve their process technology, designs, and packaging to significantly increase performance. Ultimately, these improvements trickle down to other portions of the graphics market and

even into those main memory devices being developed for the high-performance desktop market.

7.2 PERFORMANCE FOR DRAM MEMORY DEVICES

So, how do we define high performance for DRAM memory devices? Historically, asynchronous, fast page mode-style DRAM device performance was dictated by timing specifications, such as t_{AA} (access time from column access), t_{CAC} (access time from *CAS**), t_{RAC} (access time from *RAS**), and t_{PC} (Read or Write cycle time). Note that these specs, save the final one, are all measures of access time, which is the amount of time it takes for the device to drive out data following the transition of the column address, the column address strobe (*CAS**), or the row address strobe (*RAS**). Only the final spec, t_{PC}, relates to the cycle time, that is, the ability of the device to deliver data at a sustained rate from one column address to another. There were many other specifications on an asynchronous DRAM data sheet, but these four, far and away, defined performance.

What was so important about these particular specifications? With regard to t_{AA}, t_{CAC}, and t_{RAC}, these specs all defined some sort of delay: delay that the memory controller must endure every time that new data is needed from the DRAM. The shorter the delay, the sooner the memory controller could ultimately deliver data to the processor. Whenever the processor waits for data, its performance, and that of the system, suffers. Any improvement in these critical DRAM timing specifications ultimately leads to improvements in system performance. As a result, the memory industry naturally migrates towards the development of devices with faster access and shorter cycle times.

Following the advent of synchronous operation, DRAM device specifications began a slow migration towards performance metrics related to clock frequency. The most important timing parameters became t_{AC} (access time from clock), t_{CK} (clock cycle time), *CL* (*CAS* latency), and t_{RCD} (active to read or write delay). Most of these new parameters are specified in terms of nanoseconds, except for *CAS* latency, which is specified in number of clock cycles. Again, all, save t_{CK}, are a measure of delay. While the t_{AC} parameter is somewhat important, because it defines data availability relative to clock transitions, real performance is measured by *CAS* latency and t_{CK}. These two parameters define how fast the DRAM device can cycle (t_{CK}) and how many clock cycles the memory controller, and ultimately the system, must wait for data following a request for data (a Read operation).

Of course, system designers want their DRAM devices to have the lowest *CAS* latency possible at any given operating frequency. This combina-

tion yields the best system performance, but reality being what it is, won't result in acceptable device yields for the memory manufacturer. Generally, both t_{CK} and *CAS* latency incrementally improve with successive generations. Both memory manufacturers and system builders manage device performance, and the inevitable improvements, through a system known as *speed grading*. Basically, a speed grade is a set of timing parameters associated with device operation at a specific data bus frequency. At higher bus frequencies, these timing parameters become more demanding and harder to meet. Fortunately for the memory industry, devices that are unable to meet all of the timing parameters for a specific speed grade can be down-binned to a slower speed grade and sold into other market segments. Speed grades are not only important to ensure that components from different suppliers are compatible, but also to support down-binning strategies that improve overall production yields.

Synchronous DRAM (SDRAM), despite notable improvements in performance, represents only an evolution in technology. When SDRAM first appeared, manufacturers essentially converted existing asynchronous designs by merely changing *I/O* circuits to accommodate the synchronous interface. Even the device specifications remained rooted in asynchronous behavior. The real advantage gained by synchronous operation occurred in the output timing specs. Asynchronous designs had very limited data rates due to the fact that the memory controller could not issue a new data request (Read operation) until it successfully captured data from the previous Read. The controller drove *CAS** LOW to initiate the Read operation and held *CAS** LOW until it was able to capture the Read data. This meant that column cycle times included not just the DRAM access delay, but also the *I/O* transfer delays, including flight time. Accordingly, column cycle time was not optimized, because the controller was precluded from issuing a subsequent read command until it had captured data from the current read command.

Extended Data Out (EDO) asynchronous DRAMs reduced column cycle time by hiding some of the *I/O* transfer delay. While traditional asynchronous devices, such as Page Mode (PM) or Fast Page Mode (FPM), only drive Read data while *CAS** is LOW, EDO devices continue to fire Read data after *CAS** transitions HIGH. This allows the memory controller to begin to cycle the column for a subsequent Read while concurrently capturing data from the current Read. This seemingly small change allowed a reasonable gain in system performance.

SDRAM represents another step in the steady evolution of DRAM technology towards higher performance. The synchronous interface specification is important, as it essentially divides the Read operation into two separate components: the column access phase and the data output phase.

Essentially, the data output operation was pushed into a separate clock cycle from that of the array access. This meant that device speed and the associated clock frequency were now only limited by how fast the memory array could perform column accesses. Synchronization ultimately increased performance because the data output operation became a scheduled event, isolated from the array access.

Double data rate (DDR)-style DRAM devices and their faster cousins, graphics double data rate (GDDR) devices, represent an evolution in technology compared to synchronous DRAM (SDRAM). For each Read command, they drive two bits of data for every clock cycle: one bit on the rising edge and a second bit on the falling edge. This simple change, dating from technology developed in the early 1980s, effectively doubles data transfer rates to and from the DRAM [1]. The various permutations of DDR and GDDR, such as DDR2, DDR3, GDDR2, GDDR3, and GDDR4, encompass evolutionary advances in technology to achieve higher overall data transfer rates. These advances are enabled by changes in burst length, improvements in signaling technology and packaging, and significant advances in circuit design.

7.3 UNDERLYING TECHNOLOGY IMPROVEMENTS

In the balance of the book we discuss the underlying technology improvements that enable high-speed DRAM. Many of these improvements don't show up all at once, but begin to appear, in various forms, at each step of the high-speed staircase. Some of the advances are somewhat delayed, if not outright restrained, by legacy baggage that gets carried along from one generation to the next. We discuss the importance of this issue as well in upcoming chapters.

REFERENCE

[1] M. F. Novak, K. M. Guttag, and D. J. Redwine. 1987. Control of data access to memory for improved video system. US Patent 4,688,197, issued Aug. 18, 1987.

Chapter
8

High-Speed Die Architectures

8.1 INTRODUCTION: OPTIMIZING DRAM ARCHITECTURE FOR HIGH PERFORMANCE

Given the fact that the DRAM market is a commodity market, manufacturers struggle to squeeze out profit in the only way available to them, by both controlling and reducing cost, whenever and wherever possible. Because die size is a leading indicator of cost, designers have always placed significant emphasis on producing highly efficient and minimized die architectures. As the saying goes, less is more. Standard die architectures are highly optimized as a result of this quest for lower cost. At higher clock and data rates, die architectures must undergo a transformation or re-optimization, which places somewhat greater value on performance. While cost remains an overriding concern, the quest for performance places new demands on the design. When comparing die photographs of standard and high-speed DRAM devices, as shown in Figure 8.1, the difference in relative size may be easy to spot, but detailed changes in architecture may be harder to detect. In point of fact, the differences can be subtle, but nevertheless necessary to permit sustained operation at higher clock and data rates. In this chapter, we explore high-speed die architectures and, in the process, gain insight into many high-speed design issues.

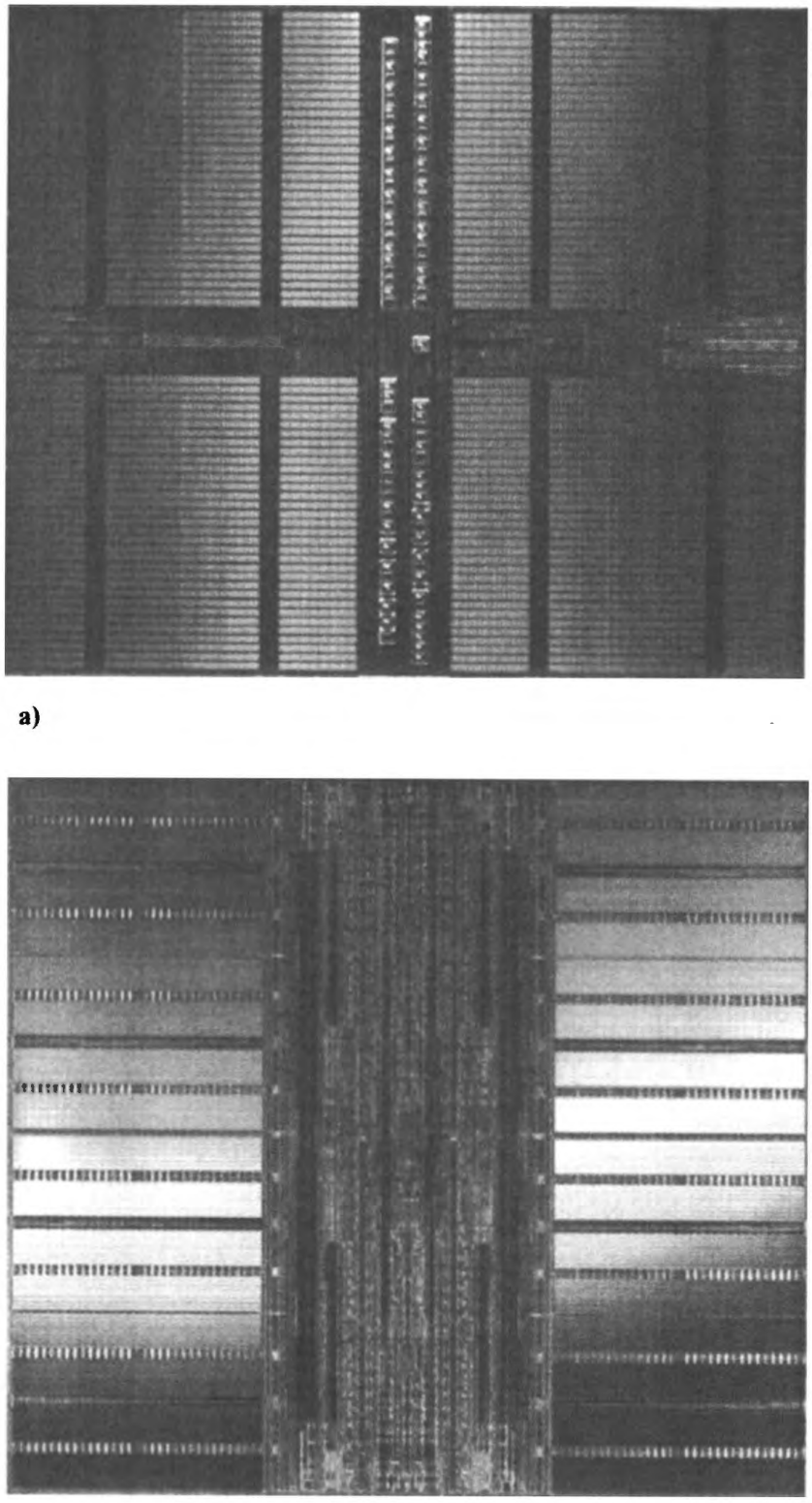


Figure 8.1 a) Standard DDR2 die photo and b) high-speed GDDR3 die photo.

8.2 ARCHITECTURAL FEATURES: BANDWIDTH, LATENCY, AND CYCLE TIME

To gain an understanding of features specific to a high-speed die architecture, let's first examine the limitations of standard die architectures. Exactly what improvements are needed to reach the next performance level on a standard die? The three device characteristics most closely associated with high-speed performance are bandwidth, latency, and cycle time. Bandwidth is the rate at which data can be written to or read from the memory device. Typically it is measured in bits per second, per pin. A standard DDR2 device, for instance, is designed to go as high as 800Mbps/pin. GDDR3 memory is designed to reach speeds of 1.6Gbps/pin and GDDR4 upwards of 2.5Gbps/pin. Latency is the delay between a request and a response. Read latency, accordingly, is the delay from the time that a Read command is issued by the host to the time that the Read data is made available to the host. Cycle time, in the context of memory, is the rate at which array activities can be repeated. Column cycle time, specifically, is the rate at which back-to-back, column-specific commands, such as Reads or Writes, can be supported within a single memory block. So, as expected, a high-performance device should be significantly better in one or more of these performance metrics than what we might call a standard memory device. However, try to keep in mind that performance and speed are in a state of perpetual migration. Today's high-performance parts eventually become tomorrow's standard-performance parts. Everything is relative.

While die size and cost are of paramount importance, most DRAM designs start with specific targets for bandwidth, latency, and cycle time. Let's take a look at each of these metrics to see how they can be impacted by die architecture.

Typical synchronous DRAM devices achieve higher data rates and bandwidth by fetching multiple bits of data per data pin (*DQ*) for every Read operation. For instance, a DDR2 device with eight data pins (X8) fetches or pre-fetches 32 bits of data during each Read operation. These 32 bits are divided amongst the eight DQ pins so that a total of four bits of data are driven out serially over two full clock cycles: one data bit per clock edge. So what limits a DDR2 device to only 800Mbps/pin data rates for Read operations? Basically, any circuit that is called into action during a Read operation is capable of limiting performance, including command and address receiver circuits, capture circuits, command decoders, column address path, column decoders, array data path, secondary sense amplifiers, Read data path, clock generation circuits, clock trees, serialization circuits, and output drivers. Let's assume however, that there are no limitations resulting from circuit shortcomings and focus only on architectural limiters.

This leaves us with the array data path, Read data path, and any limitations caused by the locality of circuit blocks.

8.2.1 Architectural Limiters: The Array Data Path

The array data path is the heart of any DRAM. It consists of the sense amplifier *I/O* transistors, local *I/O* lines, global *I/O* lines, and any DC sense amplifiers or helper flip-flops (HFF) along with Write drivers for Write operations. So, in order to increase DRAM bandwidth, the array data path must cycle faster. Essentially, bandwidth is equal to the burst length divided by the column cycle time as shown.

$$\text{Bandwidth} = \text{Burst length} / \text{Column cycle time} \tag{8.1}$$

Hence, a shorter column cycle time yields a higher data rate (bandwidth). Note that the column cycle time is expressed as t_{CCD} in DRAM device data sheets. This relationship leads to the conclusion that higher bandwidths require faster column cycle times, which brings us right back to the array data path. Historically, column cycle time hasn't ventured below the 5 ns mark for the majority of designs. In fact, burst length typically increases from one interface technology to the next to maintain the column cycle time above 5 ns. Table 8.1 shows the relationship between column cycle time, burst length, and bandwidth for a variety of DRAM technologies.

Table 8.1 DRAM technology and speed relationships.

Technology	Burst Length	Min Column Cycle Time (t_{CCD})	Peak Bandwidth
SDRAM	1	6 ns	167Mbps/pin
DDR	2	5 ns	400Mbps/pin
DDR2	4	5 ns	800Mbps/pin
DDR3	8	5 ns	1.6Gbps/pin
GDDR3	4	2.5 ns	1.6Gbps/pin
GDDR4	8	3.2 ns	2.5Gbps/pin

The apparent exceptions to a 5 ns limit are GDDR3 and GDDR4 graphics DRAM. So, what allows a graphics device to cycle faster than other technologies? Let's explore a GDDR3 array data path and find out.

To begin, we'll review what happens within a DRAM array data path during a typical column cycle. As shown in Figure 8.2, the array data path consists of a column decode circuit; column select lines; *I/O* transistors, which connect each sense amplifier to local *I/O* lines; global *I/O* lines; and helper flip-flop (HFF) circuits. In a column cycle, as shown in Figure 8.3, the column select line fires HIGH, and the *I/O* and HFF equilibration circuits are released. This permits the sense amplifier to drive data onto the local and global *I/O* lines, thus forcing a voltage differential to develop on the lines. This differential signal drives into the HFF, shown in Figure 8.4, which is fired to both amplify and latch the data for subsequent handling by the Read data path. Finally, the *I/O* equilibration circuit fires, and the column select line drives LOW. In a GDDR3 device, all of this action takes place in as little as 2.5 ns.

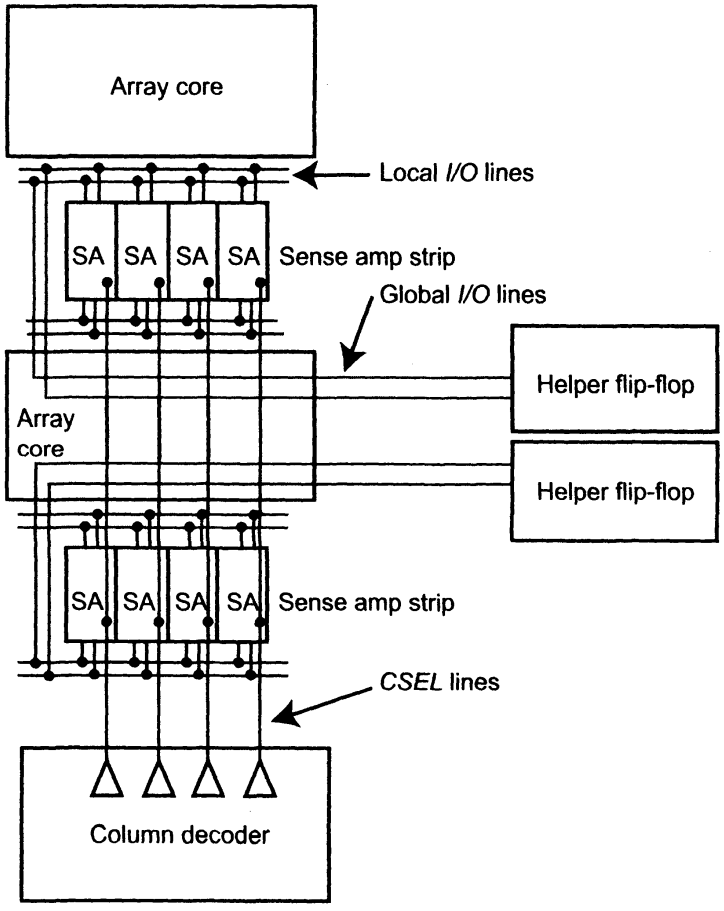


Figure 8.2 Array data path block diagram.

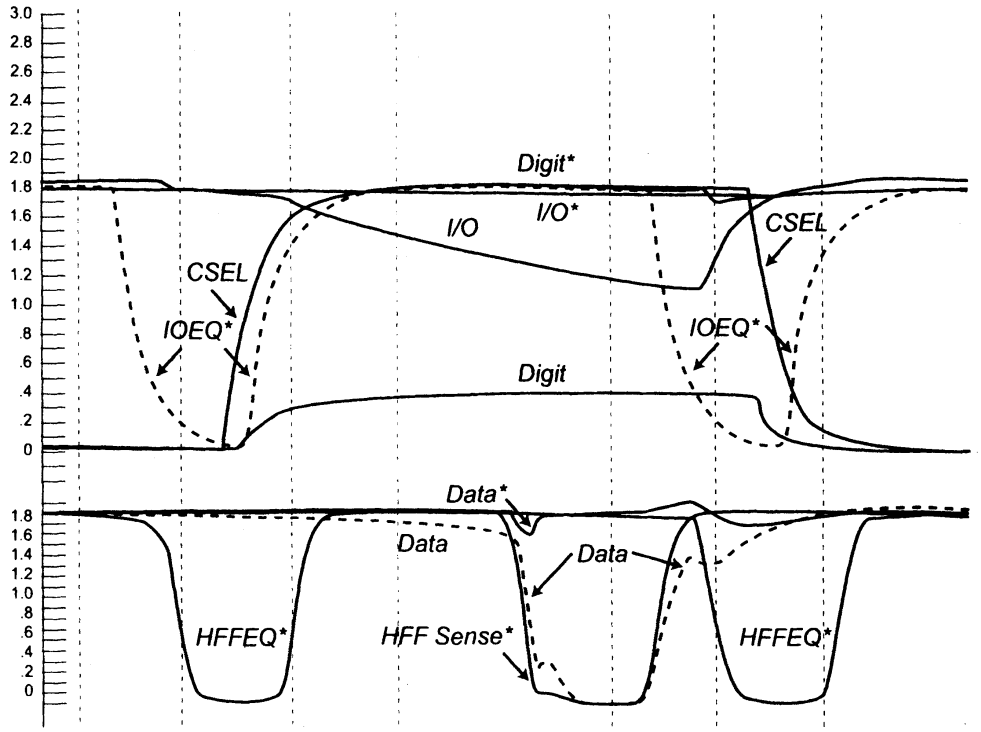


Figure 8.3 Array data path 2.5 ns column cycle waveforms (0.5 ns/division).

To maximize the performance of the array data path, all of its components must be carefully designed for high-speed operation. The column decoder circuit and the associated redundancy logic must be designed to cycle very quickly. This may require adapting different types of logic styles into the design and modifying how redundant columns are handled. Traditional column redundancy techniques push out column cycle time to allow for address matching and decoder delays. Alternative schemes, such as column steering or data suppression, may need to be considered in place of the more popular address suppression technique. Also, the global column select lines, which connect the column decoder to the sense amplifier *I/O* gates, must be carefully constructed to minimize the *RC* time constant. Finally, the *I/O* transistors must be properly sized to strike an optimal balance between minimum loading on the column select line (via smaller gate area) and maximum Read/Write performance (via low source/drain resistance). Necessary compromises such as this are common in the high-speed design realm.

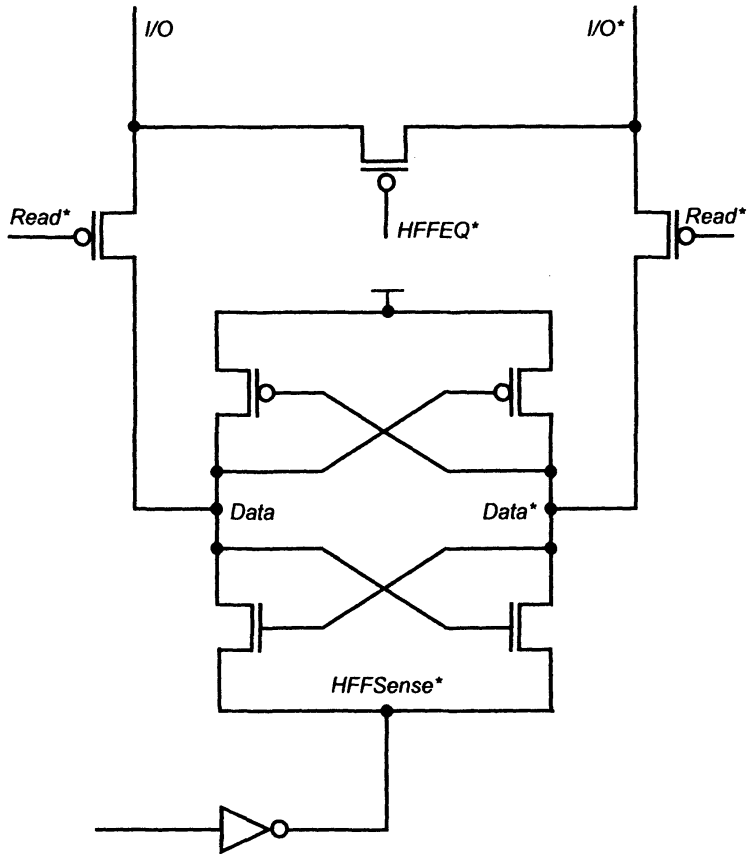


Figure 8.4 Helper flip flop schematic.

The 2.5 ns column operation shown in Figure 8.3 is a good example of finely tuned column cycle timing. Note that the *I/O* equalization signal *IOEQ** is active LOW for no more than 600 ps. This timing is tightly synchronized with HFF equalization as indicated by the HFF*EQ** signal in Figure 8.3. The column select (*CSEL*) transition is carefully placed within the equalization envelope, while exhibiting very similar rise and fall times to *IOEQ** itself. Given the short timing intervals involved, the *I/O* lines must split quickly to adequately separate the *Data* lines before the flip-flop fires, as indicated by the HFF*Sense** driving LOW. Flip-flop firing is held off to develop as much separation (signal) as possible for correct sensing. However, adequate time must be left in the cycle to transfer data into a secondary latch before the *I/O* lines and HFF equalize in preparation for a subsequent column cycle.

A major consideration, which is not apparent in the timing waveforms shown, is that the various circuits involved in column cycle operation can be distributed over large distances. Also, there is generally a multitude of distributed helper flip-flops and sense amplifier strips involved, creating an even more challenging timing problem, especially for tight column cycle

times. One good technique for managing the spatial nature of this problem is to design the signal nets with identical flight times. For instance, referring back to Figure 8.2, the flight times of the column select lines *(CSEL)* must match the flight times of the various signals controlling the helper flip-flops. This requires us to carefully manage loading, wire widths, and drivers so as to match both the propagation delay and the rise/fall times of these critically timed signals. Any carelessness or shortcut taken in this regard pushes out column cycle times. Likewise, all of the timing control circuits that drive the column decoder and array data path must be similar in structure and design to ensure that they track over temperature and voltage variations. This attention to detail is necessary so that the timing relationships, necessary for fast cycle times, are consistent despite anticipated changes in the environment.

Column cycle time can be significantly affected by architectural choices and by the construction of the array data path. These choices include array block size, array partitioning, column decoder configuration and location, global column select design, HFF configuration and location, routing metal utilization, local *I/O* line length, global *I/O* line configuration, and any decision to share resources between array banks. Let's walk through a simple set of examples to see the impact that each of these choices may have on speed.

Start by examining Figure 8.5, which shows a simplified block diagram of a memory array and an array data path. This example is typical of that found in a DDR or DDR2 SDRAM device. It consists of a pair of array blocks that are each subdivided by sense amplifier strips into four array cores. Each array block adjoins column and row decoder blocks. At each wordline and column select intersection we gain access to four bits of data, two bits per sense amplifier strip. In this example, the HFF block is shared between two array blocks. The shared HFF block necessitates the addition of multiplexers onto each HFF so that the *I/O* lines can be selected from one array block or the other. These multiplexers increase loading on the *I/O* lines and invariably impact cycle time. The sizing of these multiplexers as well as of the *I/O* equilibration devices is critical in order to keep column cycle times minimized.

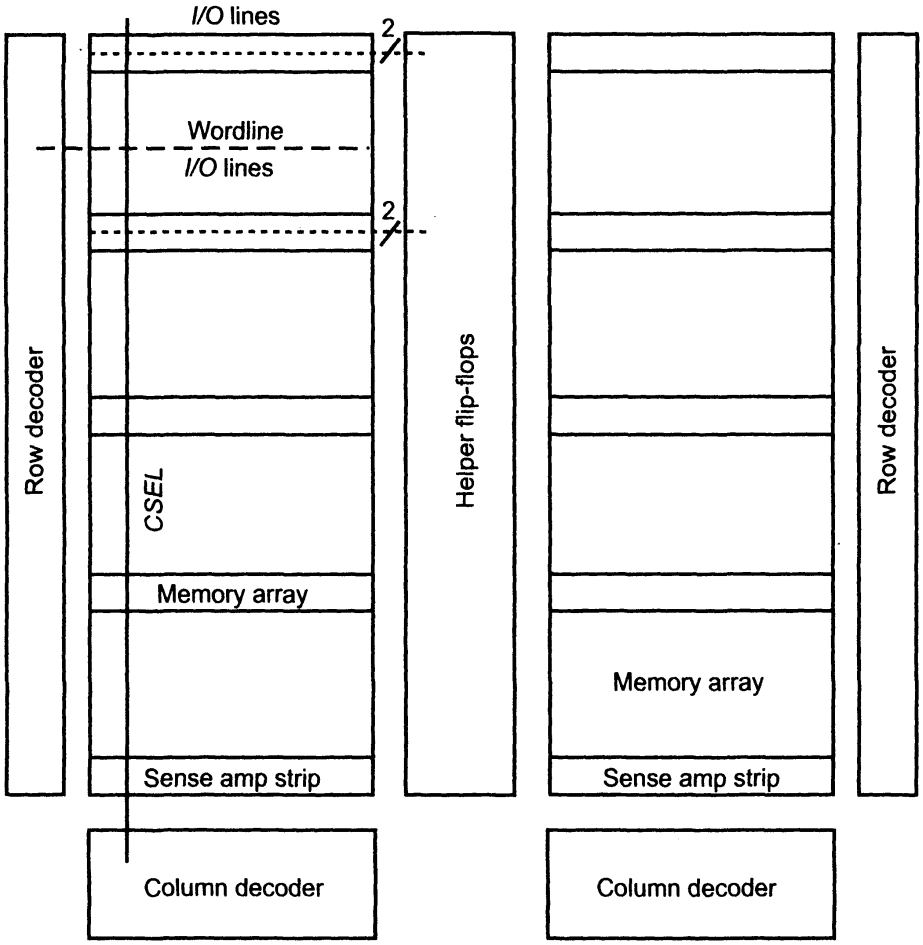


Figure 8.5 Array block diagram.

The array design as shown in Figure 8.5 is capable of accessing only four bits of data at a time. As a result, the HFF block needs to contain only four helper flip-flops. Because there are actually five sense amplifier strips that can feed data into the helper flip-flops, an additional level of multiplexing would be needed to fully implement the array data path. Once again, these multiplexers will impact column cycle time if their design is not carefully implemented. To access more than four data bits per column cycle would require basic architectural changes to the design.

The simplified array block diagram shown in Figure 8.6 is modified to permit two wordline segments to fire rather than one. With this change, eight bits of data are available from every column access, which doubles the number of helper flip-flops needed in the HFF block. Because any given HFF services only half of the array block, as compared to the configuration shown in Figure 8.5, its *I/O* lines can be shortened by almost fifty percent. Also, the *I/O* line multiplexers do not need to connect with as many sense

amplifier strips, which simplifies their design and reduces their effective loading on the *I/O* lines. These changes to the *I/O* lines and multiplexers support reduced column cycle time and higher bandwidth performance. All of this results from a simple change to the array architecture.

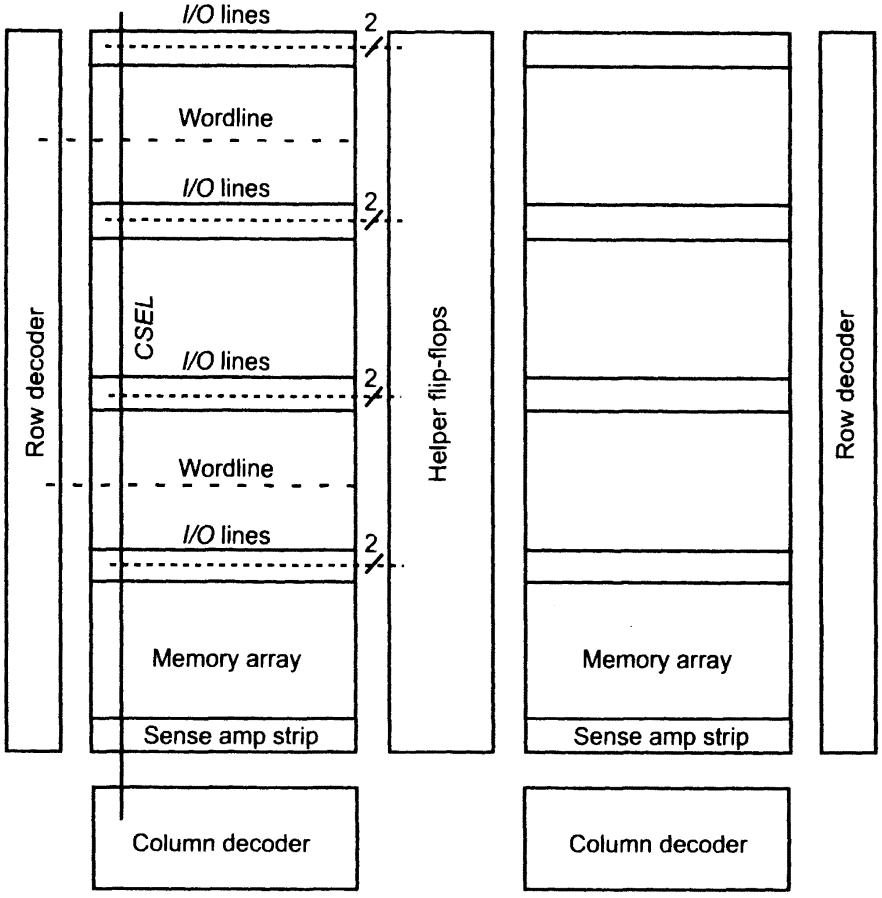


Figure 8.6 Array block diagram, double wordline activation.

Another possible architectural change to the array architecture is shown in Figure 8.7. In this embodiment, two column select lines are fired across a single wordline segment for every array access. This configuration also delivers eight bits of data for every access, requiring a like number of helper flip-flops in the HFF block. However in this design, the number of *I/O* lines running through the sense amplifier strips is doubled from that shown in Figure 8.5 and Figure 8.6. There is no relief with regard to the length of the *I/O* lines or the level of multiplexing needed to support the entire array block. So, while this configuration delivers the amount of data required by our specification, its speed is comparable to the array shown in Figure 8.5, and it is larger because the number of *I/O* lines has doubled. While this approach may not seem preferable to the double wordline activation approach in Figure 8.6, it still finds application since many device designs

have wordline activation restrictions imposed upon them by either the specification or by simple power constraints. Typically, most designs use some combination of multiple wordline and multiple column activation to satisfy data access requirements.

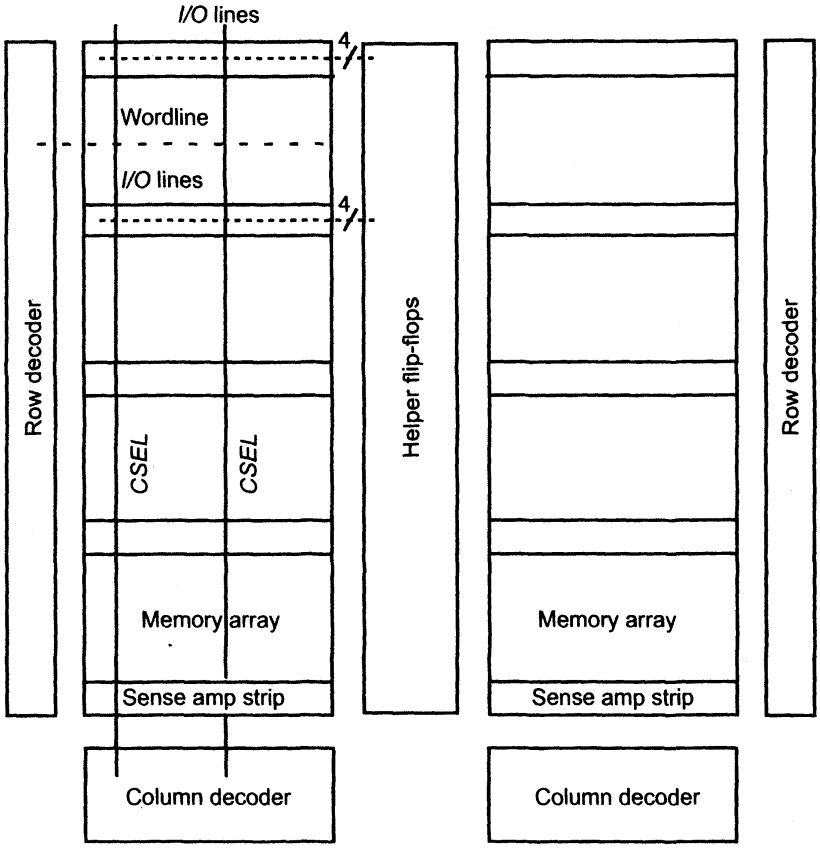


Figure 8.7 Array block diagram, double column activation.

8.2.2 Architectural Limiters: The Read Data Path

The next architectural limiter to higher bandwidth is the Read data path. The Read data path essentially starts off where the array data path ends: at the helper flip-flops. The Read data path can also limit device bandwidth if it suffers from cycle time limitations. A block diagram of a typical Read data path is shown in Figure 8.8. It consists of the helper flip-flops, one or more secondary latches, the data pipeline, the data serializer, and the output driver circuits. The Read data path design is generally constrained by both the array architecture and the locations of the bond pads. Let's take a look at how these constraints affect the Read data path and its performance.

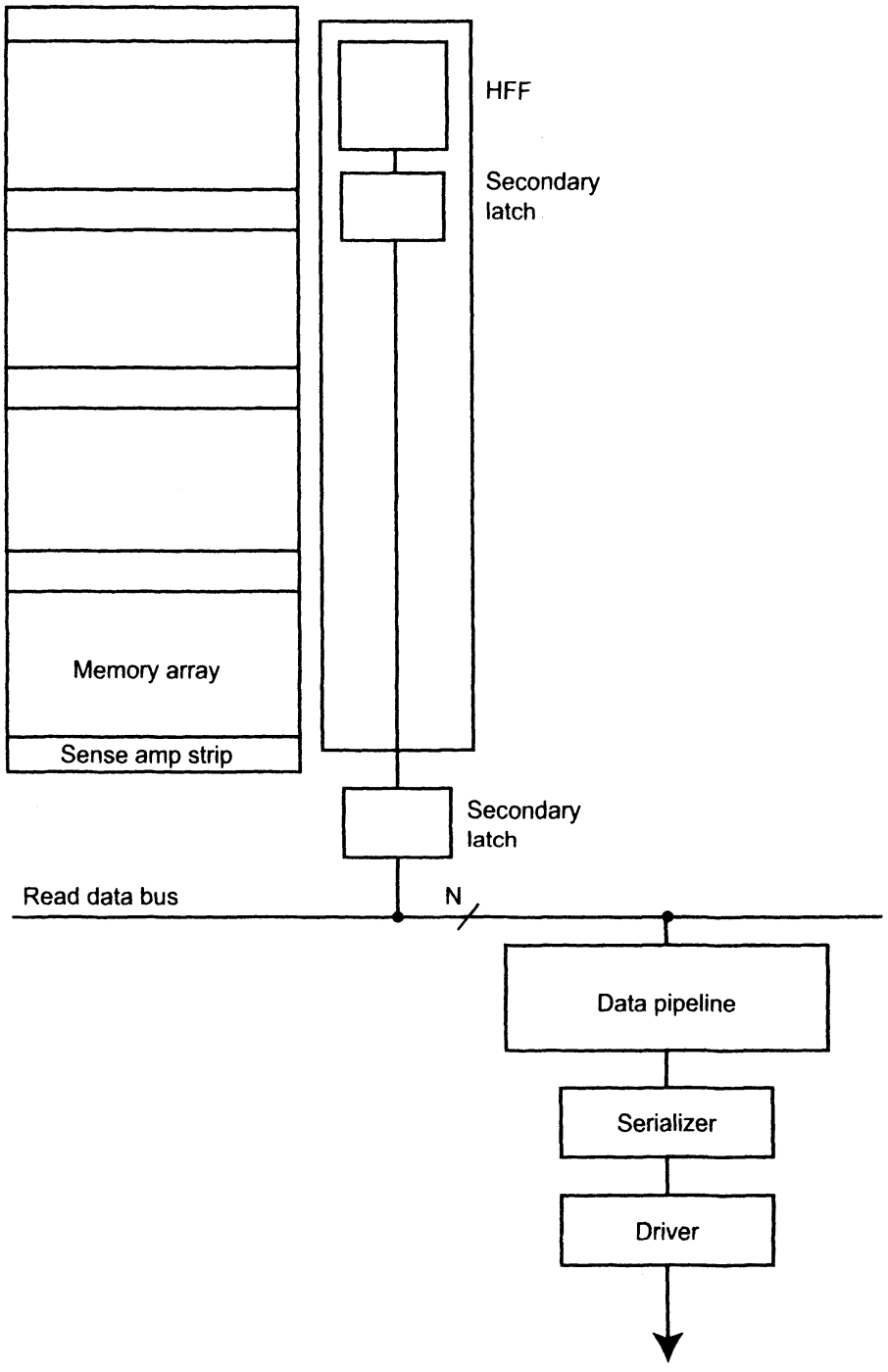


Figure 8.8 Read data path block diagram.

Figure 8.9 shows a simplified floor plan for a high-speed GDDR3 DRAM device. Graphics memory devices, typified by GDDR3, typically have 32 *DQ* pins (X32). GDDR3 and GDDR4 standards, as defined by JEDEC (Joint Electron Device Engineering Council), divide these 32 *DQ* pins into four byte groups with eight *DQ* pins in each group. As shown in

Figure 8.10, the GDDR3 package is pinned out so that there is a byte group located in each of the four corners, with the command and address pins *(C/A)* located in the center. This unique feature permits the floor plan of Figure 8.9 to be divided into four quadrants, where each quadrant services a separate byte group.

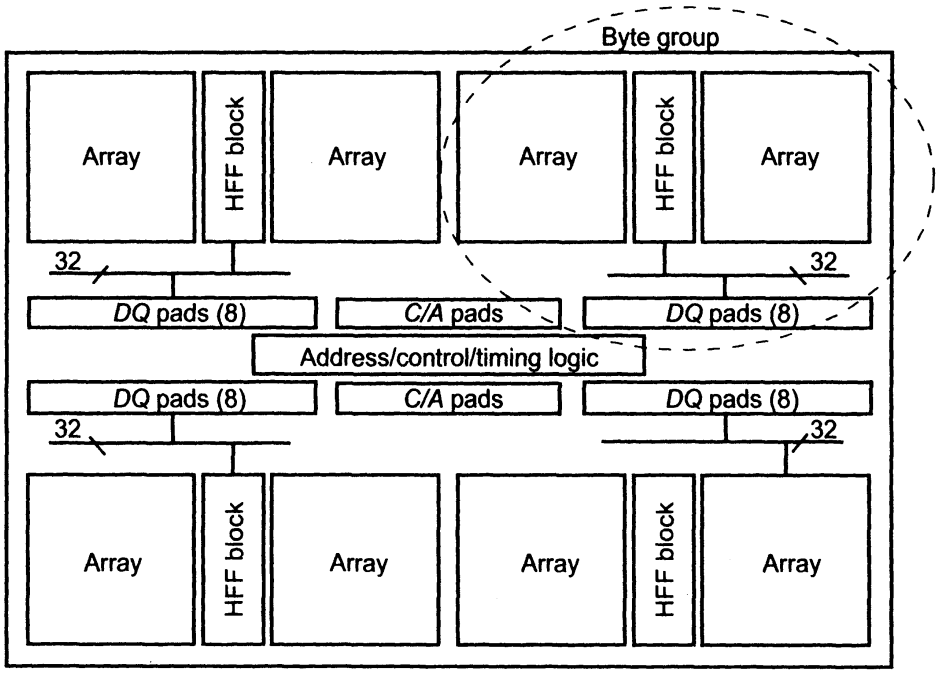


Figure 8.9 GDDR3 floor plan diagram.

To help facilitate discussion, one of the four identical byte groups is circled in Figure 8.9. Each byte group consists of a 32-bit data path that operates on four bits of data for each of the eight *DQ* pins associated with the group. The byte groups are fairly compact owing to the very pinout that gives rise to them. Package pinouts for SDRAM, DDR, DDR2, and DDR3, on the other hand, locate the *DQ* pins and *C/A* pins on opposite ends of the package. While this arrangement simplifies module routing, it penalizes the on-die data path by stretching it out along the entire length of the die. The compact GDDR3 byte group of Figure 8.9 produces shorter routing lengths and better timing control to enable faster cycle times and higher overall performance. The GDDR3 device implements two parallel pad rows. This feature permits all address, control, and timing circuits to exist between the pads. This in turn allows these circuits to be centered between all the byte groups, while keeping address and control timing tight and signal routing short.

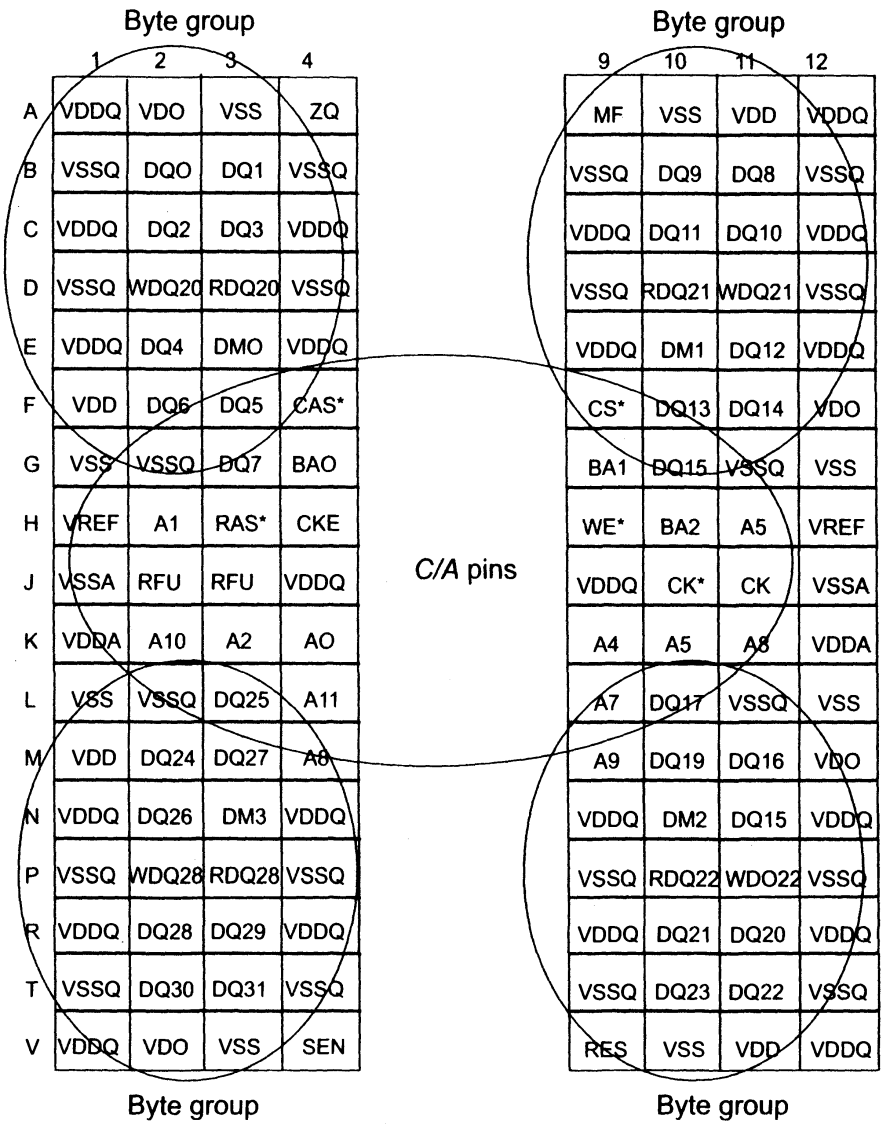


Figure 8.10 GDDR3 pinout diagram.

Additionally, Read data path components sit outside the two pad rows allowing the overall byte group to be even more compact since all of the circuits associated with that byte group are in close proximity to the array and the array data path. It should be apparent by now that cycle time and speed are improved by keeping the layout very compact and the associated routes as short as possible. For a GDDR3 device, as shown in Figure 8.9, the highly optimized placement of bond pads, array, array data path, and Read data path produce very short cycle times and excellent overall performance.

8.2.3 Architectural Limiters: Latency

The remaining performance characteristic to be discussed in this chapter is latency. Read latency is fairly sensitive to changes in die architecture.

Most of these changes can be sorted into one of two cost classifications: minor and major. Minor changes or low-hanging fruit are simple design changes and optimizations that result in minor improvements to Read latency. Major changes basically trade off die size for significant improvements in Read latency. Let's discuss each of these cost classifications in turn.

Latency is a by-product of delay accumulation that can be broadly classified as gate delay, analog delay, or synchronization delay. While a large number of circuits are called into action by a DRAM Read operation, only a portion of these circuits actually impact Read latency. These particular circuits form the Read latency critical timing path. All three delay types are usually present in this critical timing path. However, the critical timing path can actually change to a new path as individual circuits are tuned to minimize delay. This result stems from the fact that Read latency is determined by the longest—not the shortest—electrical path involved in a Read operation.

For the sake of discussion, we will define the Read latency critical circuit path for an exemplary device as comprised of C/A receivers; C/A capture circuits; a command decoder; a column address path; column redundancy; a column decoder; an array data path, including HFF; a Read data path, including pipeline and serialization circuits; and an output driver. This list of circuits is fairly representative of a critical timing path for most SDRAM designs in production and is characterized by gate delays, synchronization points, and various wire and analog delays. Of these delays, analog delays and the synchronization points are the most important.

Synchronization (sync) points are unavoidable in an SDRAM Read latency control path. This is due to the fact that latency is defined in clock cycles, not in nanoseconds. So, at the very least, the final stage must be synchronized to the clock. Additional sync points can help overcome timing uncertainties and assist with latency management. Sync points can hide significant timing variability. However these results come at a cost. Each sync point adds a clock cycle to Read latency. At shorter t_{CK} (higher frequency), this may be a small concern, but as t_{CK} pushes out at lower frequencies, these clock cycles can translate into large latencies.

So, let's return to the topic of Read latency and minor design changes. The first consideration when designing the critical timing path is striking the proper balance of delay types. Better performance is normally achieved by minimizing the number of sync points to only those required to ensure deterministic latency. Additional sync points generally contribute to higher t_{CK} dependency. The real goal is to produce the lowest Read latency possible, independent of frequency. The memory array and array data path are highly asynchronous. The timing of events within the array data path is gen-

erally controlled by either timing models or delay circuits. Timing models attempt to mimic the behavior of actual circuits or networks to safely predict when to trigger subsequent events. Their timing is designed to track any timing changes inherent in the circuit that is being modeled. While these models are important for accurately timing events within asynchronous blocks, they nevertheless introduce timing variability into the overall Read latency critical path. A pipeline circuit is placed downstream of these circuits to absorb this variability and ensure deterministic latency. The output of the pipeline feeds into a parallel-to-serial converter to ultimately synchronize the data to the Read clock.

In addition to balancing delay types within the critical timing path, it is important to keep the entire critical timing path tracking well over PVT. For instance, if two different signals are designed for a specific relationship at one operating corner, then they should maintain that relationship, as much as possible, at all other operating corners. If this timing relationship changes across corners, then the timing path sacrifices both timing margin and performance. As stated before, the best way to maintain timing relationships across corners is through careful design, with an emphasis placed on matching topology, loading, and circuitry.

Larger, more significant improvements to latency are still possible. However, these improvements are not free, being typically paid for with increased die size. An example of this is evident in reduced latency DRAMs (RLDRAMs). Figure 8.11 shows a typical Read operation for an RLDRAM. A row activation cycle with Read and Precharge takes less than 15 ns from start to finish, as shown. These devices are built to serve applications that previously resorted to using SRAM. RLDRAMs have a traditional SRAM broadside address interface but are otherwise DRAMs built for low cycle times and low Read latency. Given the broadside addressing, however, RLDRAM Read latency is essentially the concatenation of both row and column access delays. To achieve low latency, both pieces of this must be thoughtfully constructed with minimal delay in mind.

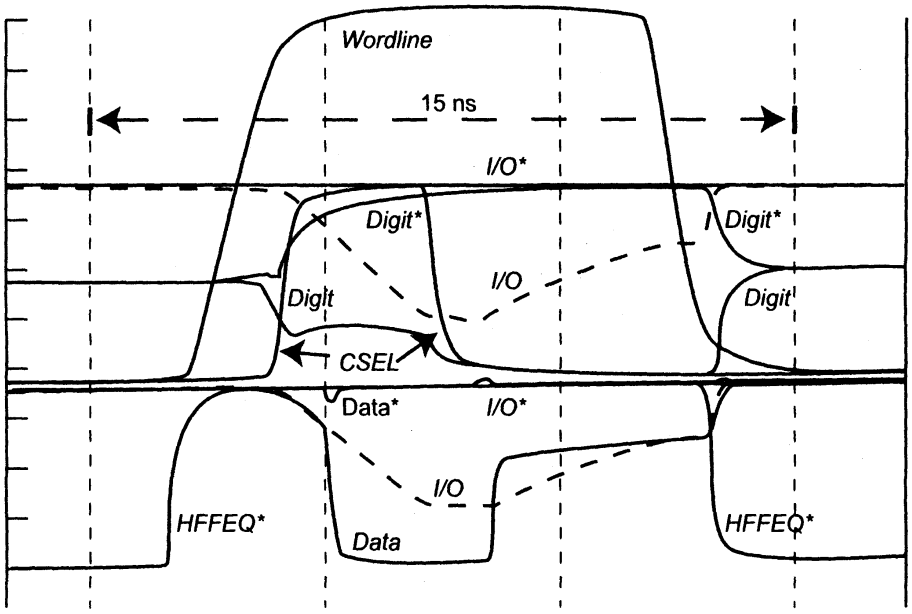


Figure 8.11 RLDRAM Read operation waveforms.

Figure 8.12 is an RLDRAM die photo exemplifying a memory architecture that is capable of both low cycle times and low latency. The most important feature of this architecture is the use of very small memory subarrays within larger memory mattes. A small block of subarrays is serviced along one edge by a local row decoder and along an orthogonal edge by a local column decoder. Both the wordlines and column select lines are kept short to keep rise and fall times short. For the device shown in Figure 8.12, each wordline segment crosses only 512 addressable digit pairs, while each digitline crosses only 256 addressable wordlines. Thus each subarray is only 128Kbits (131.072Kbits actual). The global column select lines span eight subarrays or 2,048 wordlines. This is significantly fewer than the 8,192 wordlines crossed by a column select line in the GDDR3 part of Figure 8.1b. Therefore, the length of the column select line is much shorter, which helps reduce RLDRAM cycle times and latency.

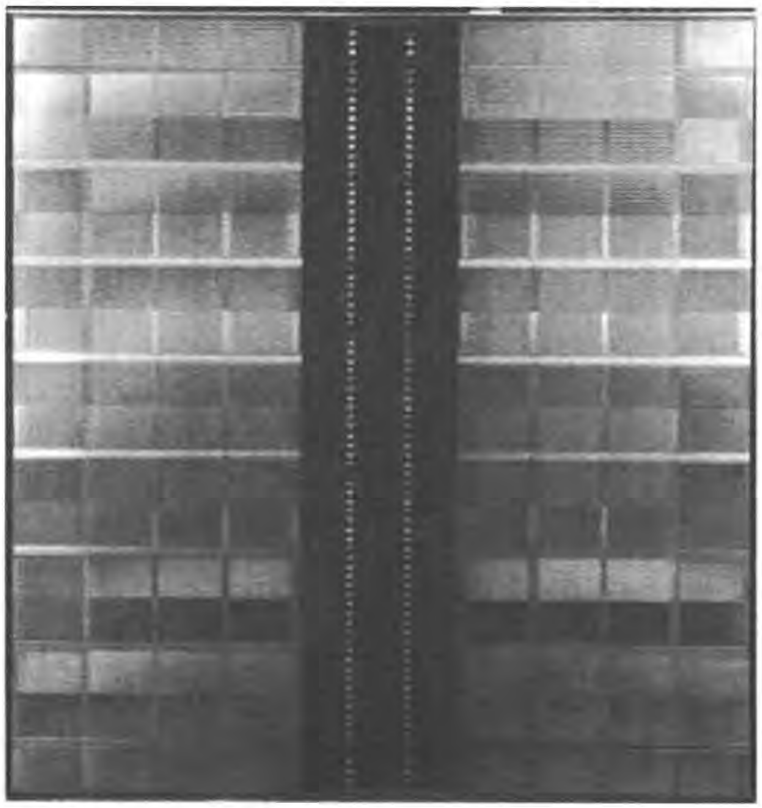


Figure 8.12 RLDRAM die photo.

It should be readily apparent that smaller subarrays are achieved by subdividing the memory array to a higher degree. Additional subdivision, in turn, requires more sense amplifier strips, larger row decoder strips, and more die area for additional column decoder blocks. So the benefit of better performance is paid for with added die size and its corresponding higher cost. In fact, the memory array and associated circuits are approximately 40–80% larger on RLDRAM than on a comparable DDR2 device. This is a necessary and reasonable trade-off for RLDRAM. In this instance, new market opportunities were created. However, if these opportunities aren't accompanied by a higher average selling price (ASP), then the device and its place in the market will evaporate.

8.3 CONCLUSION: DESIGNING FOR HIGH PERFORMANCE

In this chapter, we briefly explored how DRAM architecture impacts device performance. Architectural features, both large and small, can affect device performance for better or worse. A handful of these features can be modified or tuned late in the design cycle to squeeze out additional performance. Optimum performance cannot be achieved, however, without considering

performance/cost architectural trade-offs at the beginning of a design project. Only then, before the schematics and layout take shape, can major architectural changes be fully woven into the design fabric. After all, high performance is no accident.

Chapter
9

Input Circuit Paths

As part of the *I/O* interface, input paths manage incoming data and command and address (*C/A*) and store them for subsequent processing. Input paths may include receivers, buffers, and latches for data capture and distribution. In this chapter, we explore various receiver circuits.

For higher frequency operation, input paths also need clocks to latch and retime received data and *C/A*. After retiming, any distortion and noise introduced by the memory channels can be substantially removed before propagating further down the path. We discuss two commonly used methods for retiming in high-speed DRAM: matched routing and input timing adjustment. Both techniques involve clock distribution for multiple *I/O*s, which are spread out over a long distance. Finally, to meet stringent timing specifications and tight setup and hold at capture latches, current-mode logic (CML) is introduced trading off better timing performance with power and design complexity.

9.1 INTRODUCTION

For synchronous DRAM (SDRAM), clocks and data are transmitted together with a fixed relationship: they are source synchronous (SS). This essentially means that the clock edges used to launch transmitted data are available at the receive end for capture. While double data rate (DDR) SDRAM uses both clock edges to latch data, *C/A* uses only a single edge (single data rate or SDR). The relationships between the clock, data, *C/A*, and reference voltage (V_{REF}) for DDR SDRAM are shown in Figure 9.1 [1–3]. Because data is running twice as fast as *C/A*, additional clocks called *data strobes* can be included in the *I/O* interface and dedicated for high-speed data capture. In other words, strobes can be treated as a special type of data as they are bundled and tightly linked (source synchronous) with

data. Strobes save power because they are active only in the presence of data, unlike clocks, which are active continuously. Furthermore, strobes, either single-ended or differential, provide predictable timing relationships when synchronized with system clocks (*CLK/CLK**). High-speed data capture is more reliable with dedicated strobes than with non-source-synchronous system clocks.

Clock and data alignment are also critical for data capture. Clocks and strobes (*DQS*) are usually center aligned with incoming *C/A* and data, respectively. Center alignment creates the largest timing margin for the capture latches. Timing parameters, such as data setup (t_S) and hold (t_H) are illustrated in Figure 9.1 along with input voltage margins (V_{IH}/V_{IL}). As data rates continue to increase (from 133 Mb/s for SDR to 1,600 Mb/s for DDR3) and supply voltages scale down (from 3.3 V for SDR to 1.5 V for DDR3), the valid data windows (or data "eyes") shrink due to clock jitter, duty-cycle distortion, inter-symbol interference (ISI), simultaneous switching noise (SSN), impedance mismatch, reference noise, and supply noise, to name a few. To address these signal integrity (SI) issues, we investigate several circuit techniques in Chapter 10, such as on-die termination (ODT) and data bus inversion (DBI).

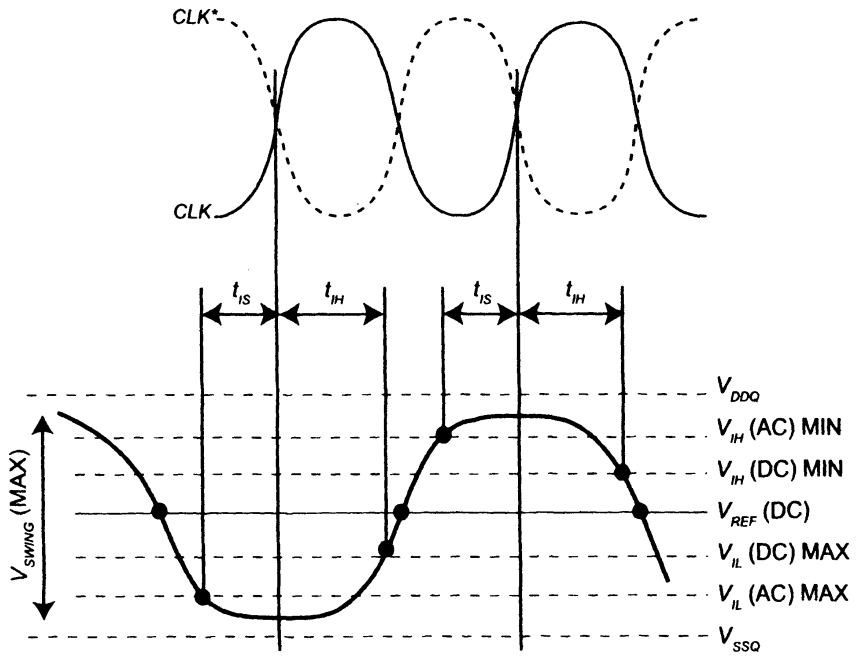


Figure 9.1 a) Setup and hold timing diagram for *C/A*.

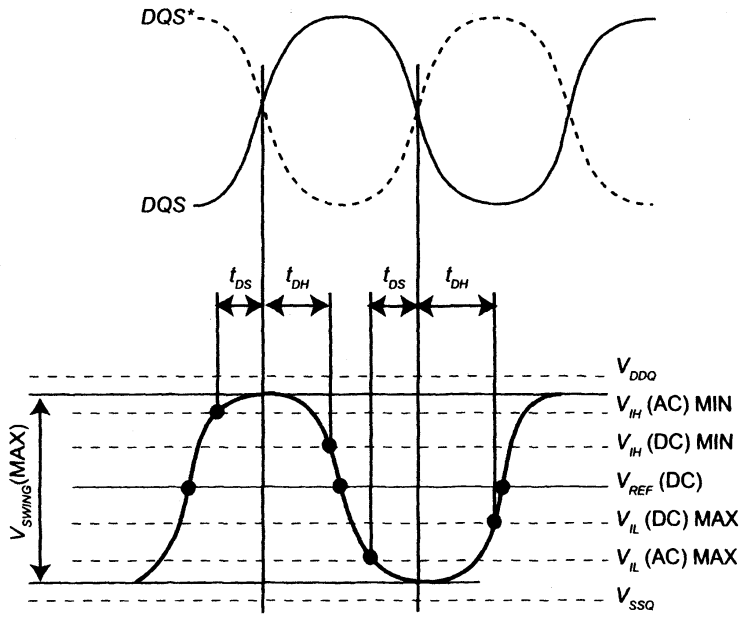


Figure 9.1 b) Setup and hold timing diagram for data.

Timing uncertainty is another challenge for data capture and processing. A clock path that runs between the clock (or strobe) pads and capture latches may include a clock receiver, timing adjust circuits, and a clock distribution network (or clock tree). Data and *C/A* paths, on the other hand, can be much shorter with only input buffers. With different electrical lengths, the input paths are no longer source synchronous internally. Furthermore, process, voltage, and temperature (PVT) variations introduce excessive jitter, and even post-production trimming may not be sufficient as the timing margin shrinks. One way to retain the clock and data relationship is by matched routing all the way from the pads to the capture points. On-chip, closed-loop synchronization or input path "de-skewing" is another approach. For instance, data training, either slaved to the host memory controller or conducted inside the DRAM itself, can greatly improve the capture timing and ensure reliable data capture over PVT variations. Timing alignment methods are addressed in Section 9.3 and Section 9.5.

Besides increasing *I/O* speed, prefetching multiple bits of data can improve memory throughput significantly without sacrificing internal array cycle time. A 4n- or 8n-Prefetch widens the internal bus compared to the external data bus. SIPO (series in parallel out) circuits and pipelines can cooperate with input circuits to tailor the incoming data into the internal buses based on the decoded commands and programmable Write latency. Control logic for the internal data path is discussed in Chapter 12. This chapter starts with input receiver design.

9.2 INPUT RECEIVERS

Due to pin limitations in DRAM, data buses are usually configured bi-directionally, while command and address (*C/A*) buses are unidirectional. Data and *C/A* signals are single-ended with reference to a common voltage (V_{REF}). Clocks and/or strobes, on the other hand, come in pairs for differential signaling and are more immune to common mode and coupling noise. Input signals like clock, data, and *C/A* are first captured by receivers or buffers. Two primary criteria should be met when designing the input receivers: 1) input signals must convert from external to internal signaling levels and 2) information presented in the original signals must be maintained.

Input signal imperfections make signal conversion difficult. Figure 9.2 shows delay variations based on Figure 9.2a signaling levels, Figure 9.2b slew rate, and Figure 9.2c common-mode range (e.g., center terminated versus V_{DD} terminated) and reference voltage. As a consequence, timing jitter can occur, which greatly impacts data capture. One way to minimize timing variation is to retain the same input receiver characteristics for data, *C/A*, and the clocks, which is implied in the second design criterion. The other way to minimize delay variations and extend the effective data eye is by using V_{REF} calibration.

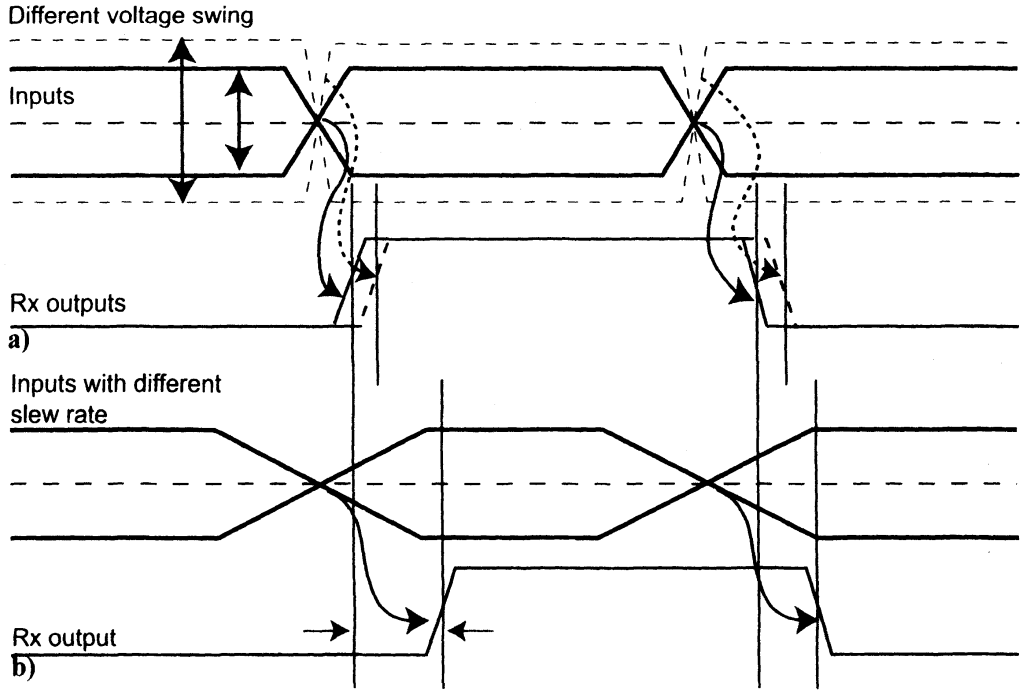


Figure 9.2 Input receiver delay variations based on a) signaling levels and b) slew rate.

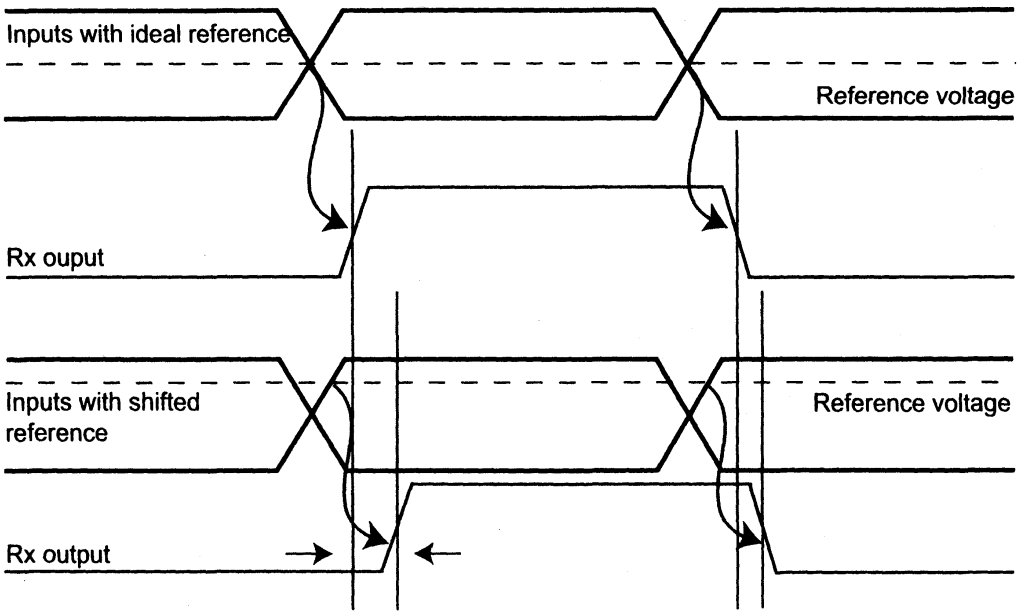
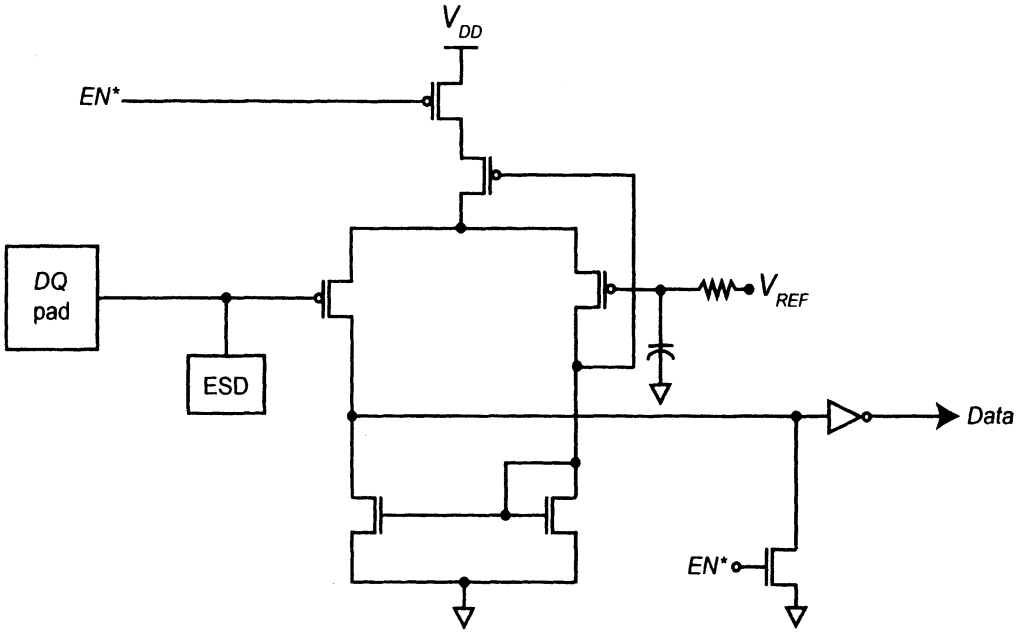
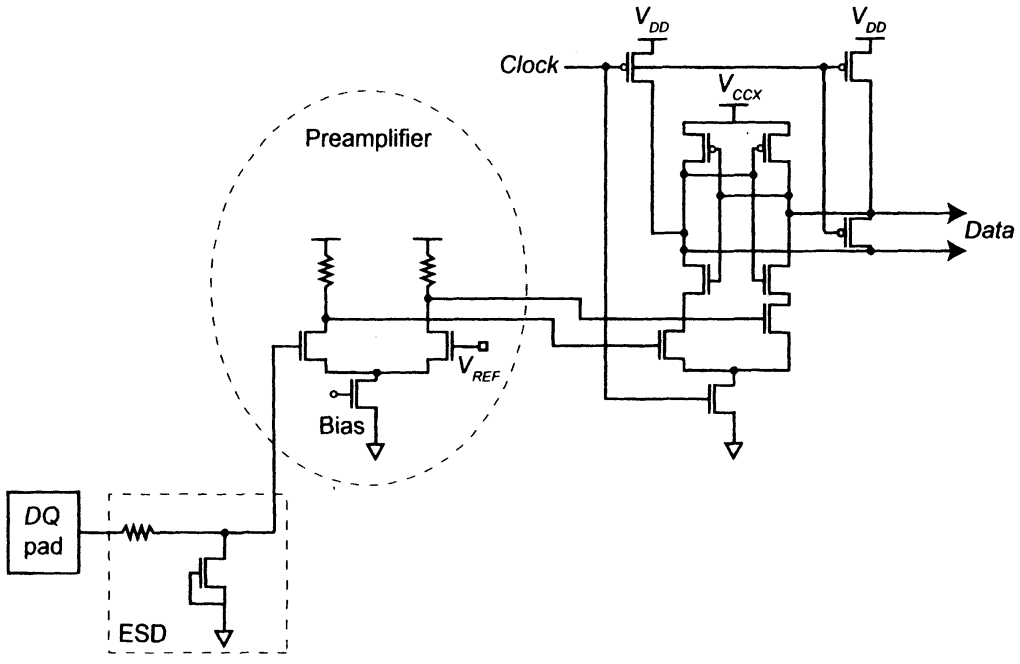


Figure 9.2 c) Input receiver delay variations under different common-mode range or reference voltage.

Several input buffers or receivers based upon differential amplifiers (diff-amps) and self-bias techniques have been shown in Figures 5.3, 5.4, and 5.5. Other examples of data receivers employing different current sources, voltage clamps, and input pairs are given in Figure 9.3. The trip-point of the second-stage amplifier (inverters in Figure 9.3a) becomes critical because it needs to complete the conversion of the input signal to internal CMOS levels. Different rise and fall times reduce the valid data window. The high-speed comparator shown in Figure 9.3b can be used as a receiver having a pre-amplifier (current-mode logic or CML) and decision circuit. This receiver has less propagation delay and is less sensitive to PVT variations. Although designed for fully differential signaling, one of the two inputs for this differential receiver comes from a reference voltage (V_{REF}). This is commonly referred to as *pseudo-differential* signaling. To achieve the best input timing, V_{REF} can be trained on-die or externally by the memory controller.



a)



b)

Figure 9.3 a) Differential receivers and
b) comparator with differential CML as input receivers.

An important concern with low-voltage design is the need for an improved common-mode range. Two complementary diff-amp stages can be placed in parallel to extend the usable operating range for the composite

receiver, as shown in Figure 9.4. Notice that in order to get a better voltage margin, not all of the devices are biased in the saturation region. For fully differential clock receivers, the reference input *(V_{REF})* is replaced by the complementary clock input *(CLK*)*.

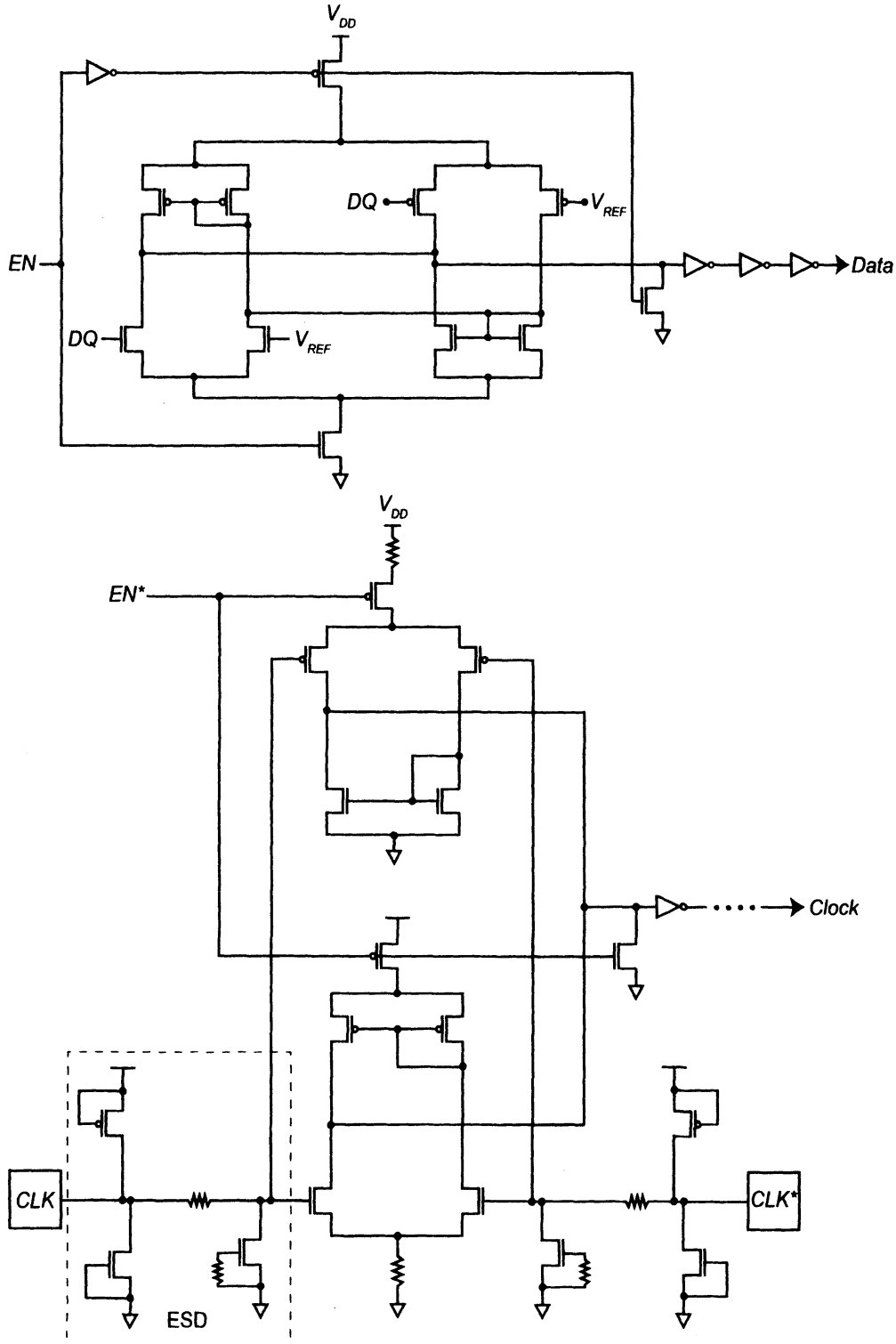


Figure 9.4 Wide-swing diff-amps for input receiver.

Other considerations for input receiver design include electrostatic discharge (ESD), PVT sensitivity, capacitive loading, power, and layout area. Between input devices and pads, a thick oxide pass-gate with a higher gate voltage (V_{CCP} in Figure 10.23) can provide protection for the thin gate oxide present in the receiver transistors. For device protection, voltage clamps are also used. Low-pin capacitance is crucial to achieving higher data rate. Unfortunately, higher gain and lower offset generally require large input devices. Power consumption, slew rate, and the overall complexity of the receiver design must also be considered.

9.3 MATCHED ROUTING

As mentioned, maintaining the original timing relationship (i.e., source-synchronous between clock and data) is important for data capture. Matched routing is the most effective solution for establishing and preserving data capture timing. The quality or property most important for matched routing is delay. Delay mismatch produces timing skew that can negatively impact input or output timing specifications. Sensitivities introduced by process, voltage, and temperature (PVT) variations are highly undesirable, too, and can impact yield and potential t_{CK} migration. Sensitivity to PVT can be reduced or mitigated by eliminating all sources of potential mismatch—in essence, every single element of the matched routes must be "exactly identical." Being exactly identical necessitates that the following elements match:

- Conductor width, spacing, and length for every segment of the routes. Each route, including the conductor layer, must match segment-for-segment.
- Driver topology and size.
- Load topology and size.
- Shielding.
- Route topology.
- Quantity of bends and their location.

A simple point-to-point routing topology is shown in Figure 9.5. A path with two identical inverters and metal1 (M1) and metal2 (M2) routes is drawn in both schematic and layout views. A "matched" path is placed side-by-side for comparison.

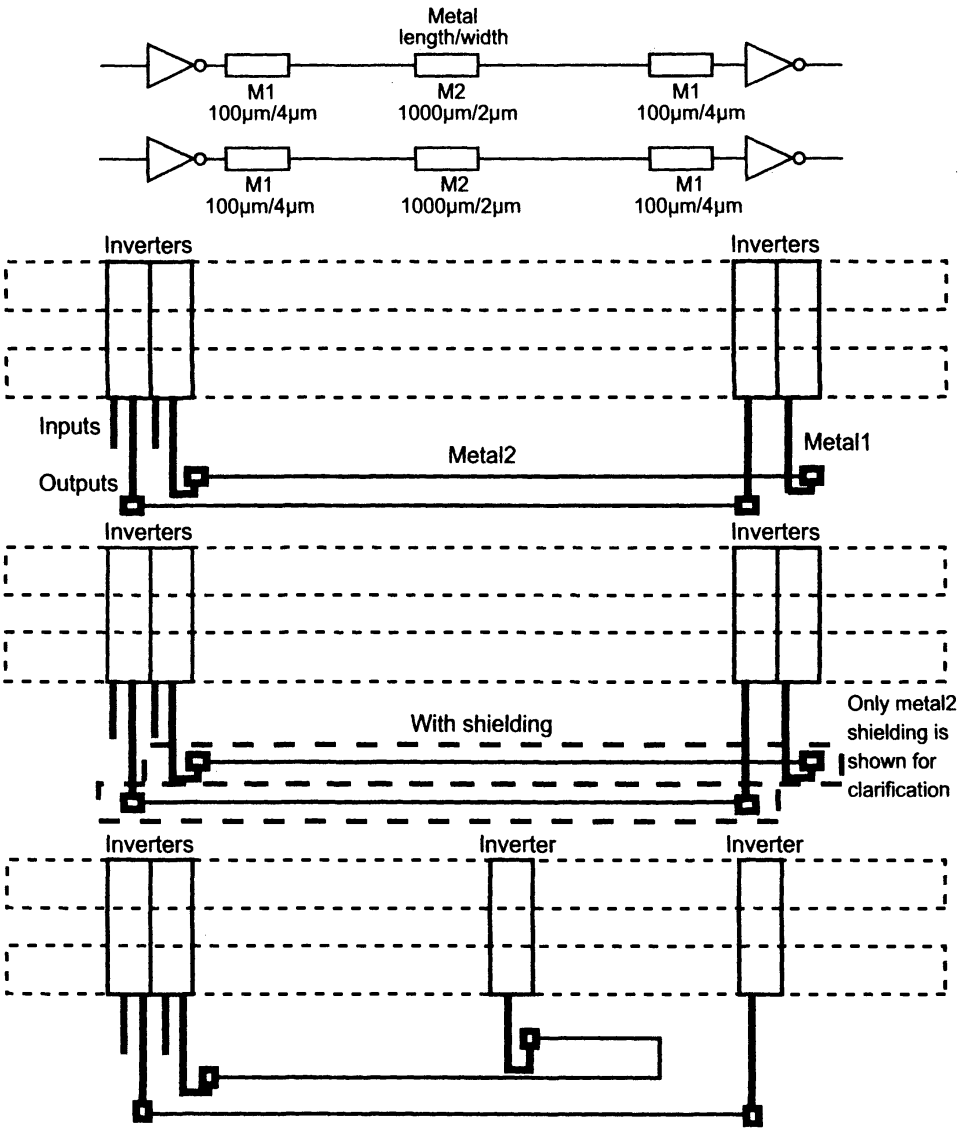


Figure 9.5 Point-to-point matched routings.

The placement (locality) of input paths can greatly impact the length and shape of the route and should be floor-planned in the early design phases. Shielding further improves matching by eliminating in-phase or out-of-phase coupling. However, shielding also increases the overall layout area and capacitive loading.

Three routing styles, with and without shielding, are seen in Figure 9.6:

1. Size maze: Efficient for adding length to horizontal routes. (Style can be rotated for vertical routes.)
2. Trombone: For adding length to vertical routes.
3. Accordion: For adding length, but requires excessive corners.

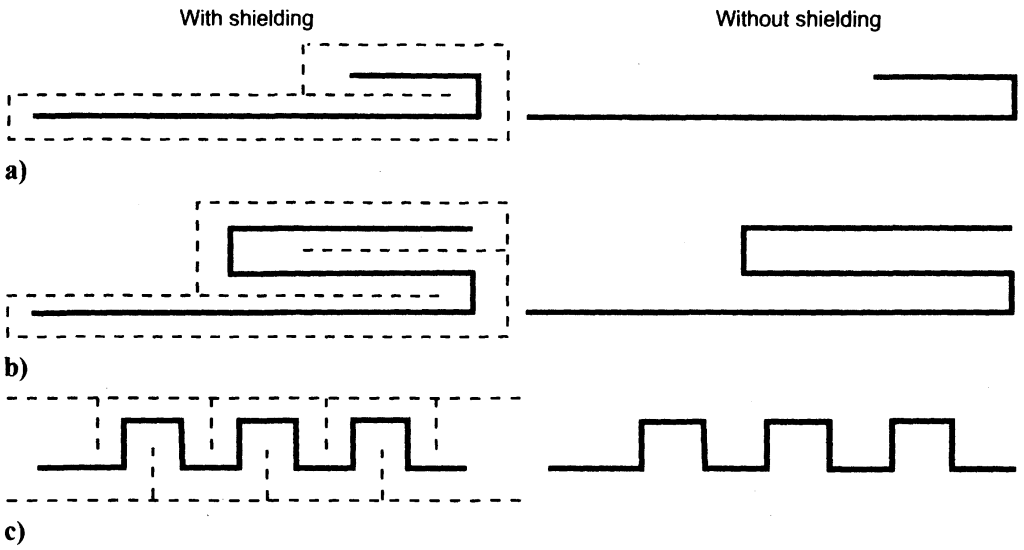


Figure 9.6 Matched routing styles: a) size maze, b) trombone, and c) accordion.

Matching the clock/strobe path to the data path is another challenge. Instead of point-to-point, the clock path must branch or tree out to multiple locations. Routing distances are longer for the clock path than for the data path. Extra buffer or repeater stages are required to drive the clock over a long distance with a minimum propagation delay. Loading difference introduces another delay variation. A tree-typed clock distribution network (CDN) may be used for matched routing. An example for data and clock matching is shown in Figure 9.7, with both point-to-point and tree-typed routes.

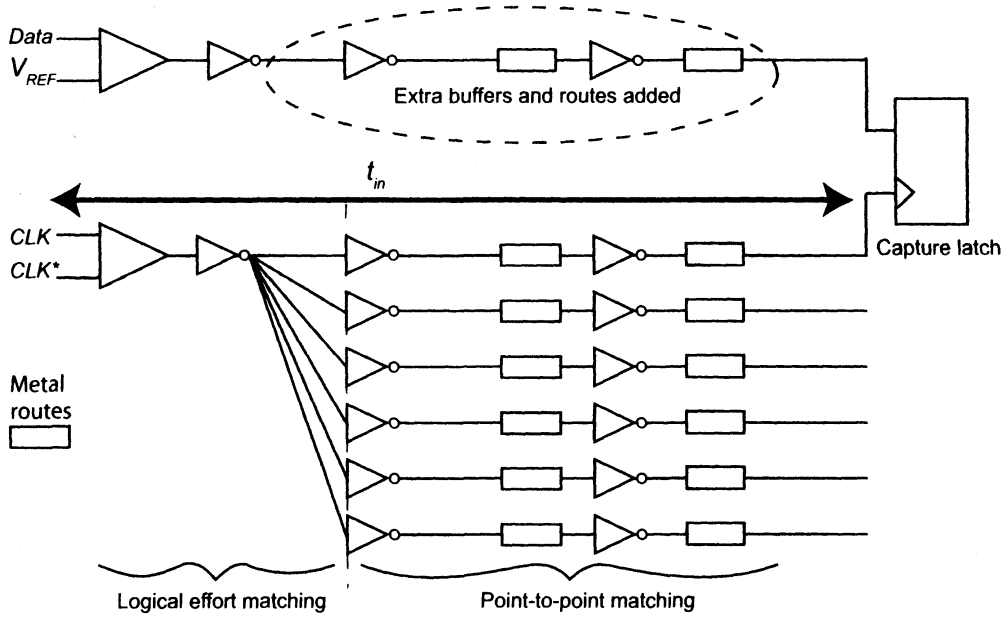


Figure 9.7 Matched routing for data and clocks/strobes.

In order to properly size clock drivers, logical effort [4] can be evaluated based on logic style, loading, and propagation delay. Extra buffer stages and routes are added on each data or *C/A* path to match the clock/strobe path. Input path delay, denoted t_{in} in Figure 9.7, represents the delay from the input pins to the capture latches, including the receiver, buffer, and matched path delays. One measurement for the quality of matched routing is the difference of t_{in}. For well-matched input paths, less than 10 ps timing skew can be achieved across simulation corners with layout back-annotation. Less than 10 ps skew across the data bus was confirmed in silicon for a GDDR3 part. However, when matched routing is applied, a trade-off must be made between capture timing and the power and layout area consumed by the routes.

From a layout point of view, matched routing is rarely accomplished by automated tools due to its complexity and restrictions. Before preparing full custom layout, a preliminary-matched route design must be budgeted and floor-planned. Basic route topologies, routing distances, materials, dimensions, buffers, and loads must be determined and tuned to obtain the desired signal quality. The initial design phase relies on using *parasitic resistor and capacitor* (PRC) models to estimate routing parasitics, which enable dimensional and electrical tuning of the routes. Pad order and pad pitch also have a direct impact on the route cells. An automated tool is helpful for checking the final quality of matched routing by comparing conductors, topologies, bends, and sizes. SPICE back-annotated simulation provides timing verification at the final design phase.

9.4 CAPTURE LATCHES

The next logical step after matched routing is data capture. A capture latch acts like a storage element controlled by clock edges. High-speed data requires two capture latches: one for rising clock edges; the other, falling edges. *C/A* typically needs only one latch in response to only rising clock edges. After the capture latch, data is split into two parts, running at half the input data rate for each part. *C/A* also gets extended to a full cycle (t_{CK}) for a valid window. Figure 9.8 shows a capture timing diagram, where t_{CQ} is the clock to Q delay of the capture latch, and t_{in} is the input path delay, which includes the receiver and matched routing.

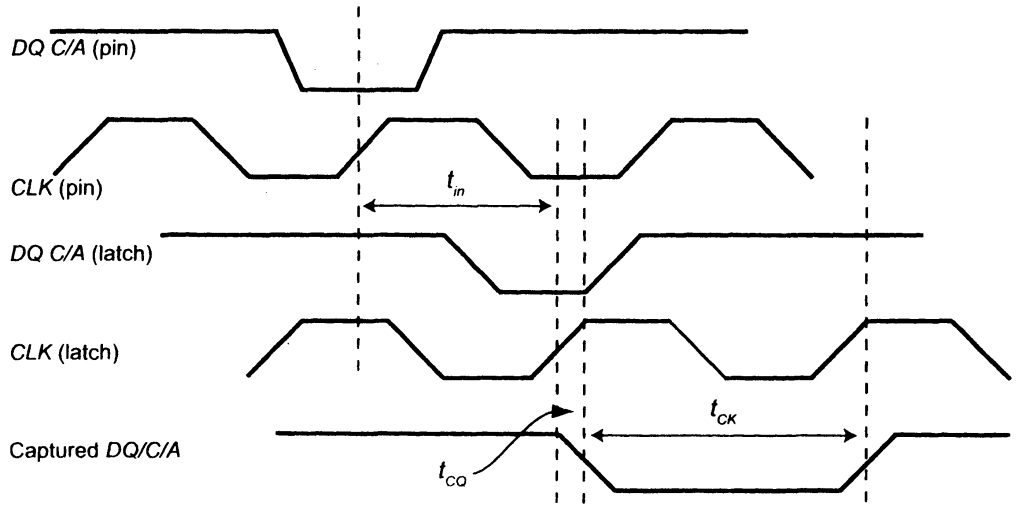


Figure 9.8 Timing diagram for input capture latches.

The implementation of a capture latch is straightforward if the clock and data relationship (center aligned in this case) can be preserved. A static edge-triggered flip-flop (FF) is shown in Figure 9.9, with master and slave latches and transmission gates (TG). Complementary clocks sample the inputs and control the latches. Set-up time for this FF is in the order of hundred picoseconds, depending on PVT corners.

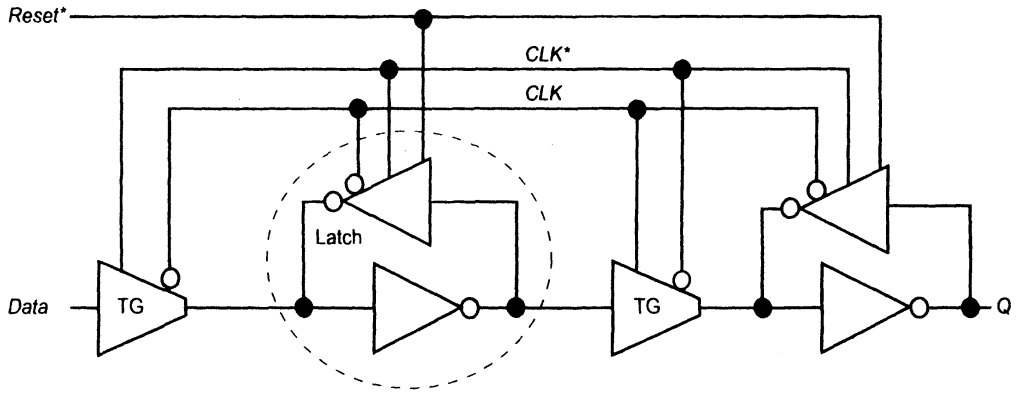


Figure 9.9 A static edge-triggered flip-flop as a capture latch.

A dynamic latch with tens of picoseconds set-up time is shown in Figure 9.10. Compared to a TG and latch in the static FF, loading for the input data is only a transistor. For it to switch, the input need only overcome the threshold voltage of MP1, making it significantly faster compared to static FF. Sampling clocks, *CLKS1/2*, have much less loading compared to static FF. *CLKS2* is one gate delay version of *CLKS1* to reduce overlapping current. A keeper circuit at the output tries to hold the value during the Precharge phase (clock is HIGH). This domino-typed dynamic FF is useful for high-speed capture latches.

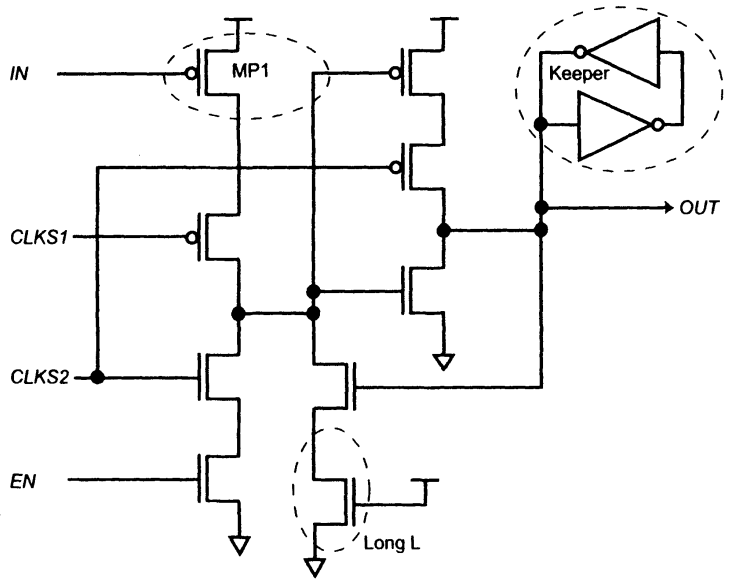


Figure 9.10 Domino-typed dynamic FF.

For high-speed data capture, sense-amplifier latches can also be applied, as shown in Figure 9.11. With cross-coupled devices, they are fast and have high gain. They can sense small voltage differences at the inputs in response to clock edges, which makes them ideal even as input receivers. A second stage, like the one shown in Figure 9.10, is necessary to store and convert the sensed value. To reduce offset and ensure proper sense function, careful layout of the input pairs is imperative.

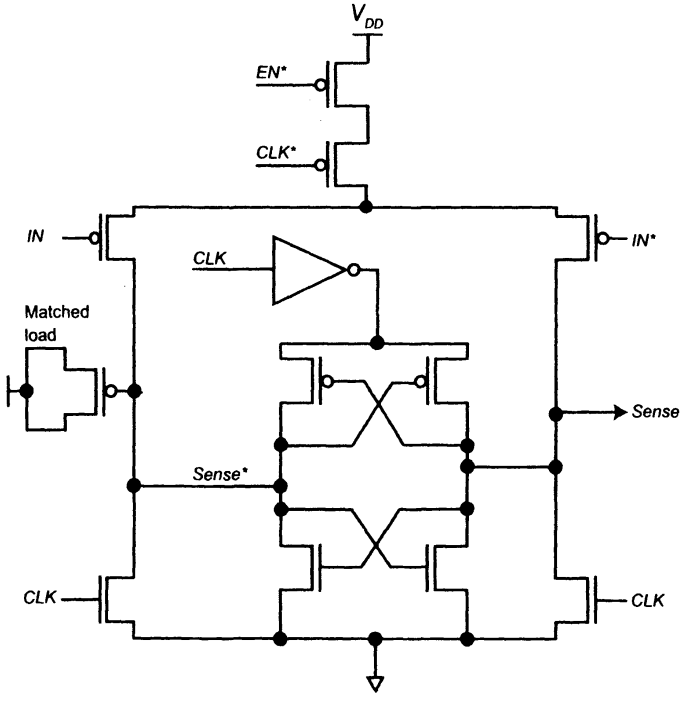


Figure 9.11 Sense amplifiers as front stage of capture latches.

9.5 INPUT TIMING ADJUSTMENTS

Write data is received and captured by the Write strobe (*WDQS*), although the data timing essentially remains in the *WDQS* domain. The Write command is decoded via command and address (*C/A*) paths, which are timed by and remain in the system clock (*CLK/CLK**) domain. Therefore, two clock domains exist: one for the Write command and the other for Write data. The timing difference between system clocks and data strobes at pins is defined as t_{DQSS} and is tightly controlled by the system to limit the timing variation to less than 0.25 t_{CK}, for example. Internal mismatch between the data path and the Write command path makes this timing uncertainty even greater under PVT variations. Timing adjust circuits, like delay-locked loop (DLL) or phase-locked loop (PLL), can be useful to reduce timing uncertainty, cross clock domains, and ensure deterministic Write latency. Block diagrams with and without *C/A* DLL are shown in Figure 9.12a and Figure 9.12b, respectively.

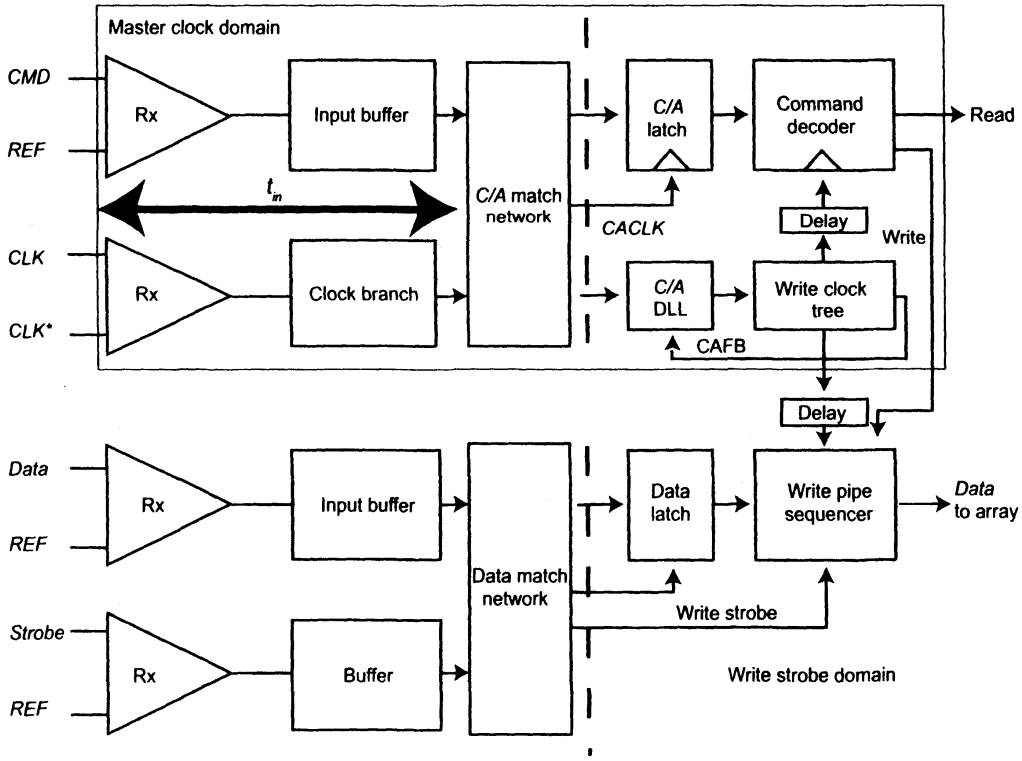


Figure 9.12 Input clock domain crossing with a) *C/A* DLL.

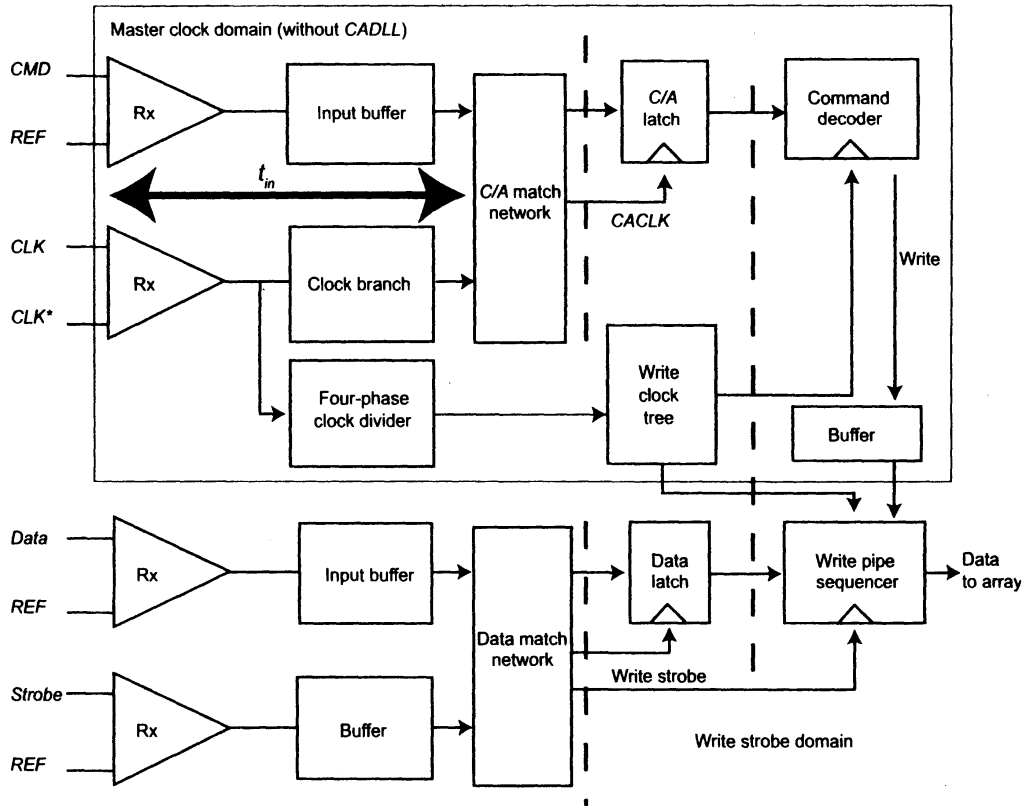


Figure 9.12 Input clock domain crossing with b) matched routing.

As shown in Figure 9.12b, a clock path that includes a clock divider and Write clock tree, is not strictly matched to the input paths. The Write clock tree is routed to both the command and data paths, which makes it longer than the input matched network. However, the Write command must be placed precisely in order to hit the setup and hold window in the Write pipeline sequencer, which is controlled by the Write strobe. So, clock domain crossing and timing adjustment may require extra buffers and delays in the input matched paths, which increase latency, power, area, and sensitivity to PVT variations. Data training can provide better alignment between strobes and clocks during initialization. This open-loop "de-skewing" exercise conducted by the memory controller can help meet t_{DQSS}, t_{DSS}, and t_{DSH} specifications. This is referred to as *Write leveling.*

Instead of training or matching, an on-die timing adjustment circuit (i.e., DLL) can be used. With the input *C/A* DLL shown in Figure 9.12a, the Write clock tree delay can be hidden by synchronizing *CACLK* with the outputs (*CAFB*) of the clock tree. Capture latch delay can also be modeled in the feedback path. The *C/A* DLL provides flexible tuning over PVT variations and permits on-die Write leveling. However, the DLL also adds some timing uncertainty compared to strictly matched routing. There are also concerns about extra power, area, and "wake-up" time.

As signaling speeds continue to increase, the corresponding valid data window shrinks due to intersymbol interference (ISI), random jitter (RJ), SSO, duty-cycle distortion, supply noise, and cross talk. The eye diagrams shown in Figure 9.13 compare an ideal case (Figure 9.13a) running at 5 Gbps to a more realistic system (Figure 9.13b) with all of the additive noise and jitter. Distributing data and clock at such high speeds on-die is a challenge. Matched routing may not be suitable for data rates greater than 3Gbps. The clock distribution must be redesigned to minimize jitter as well as clocking power. One way to relieve this problem is to divide the clock down, for example, so that it runs at a quarter of the data rate. Another practical approach is to recover the clock on-die based upon the data stream. These circuits are referred to as *clock recovery circuits* (CRC) in most communication systems [5]. Since clocks recovered by CRC are not source synchronous with incoming data, it is critical to generate low-jitter local clocks, which usually require a phase-locked loop (PLL). To provide enough transitions so as to avoid "frequency wandering" by PLL oscillators, data scramble is mandatory. However, the added complexity and cost make it unattractive for a memory interface.

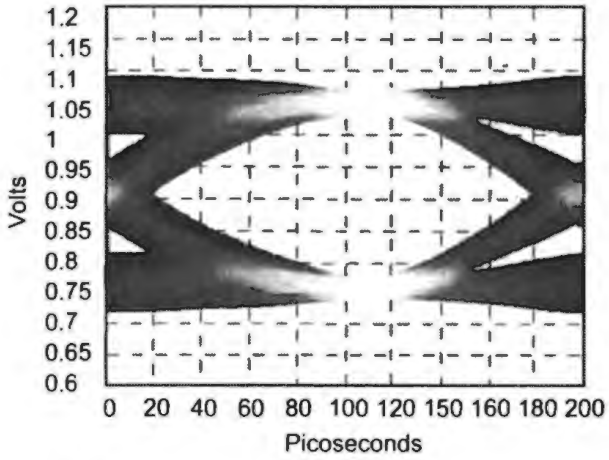


Figure 9.13 Eye diagrams for 5Gbps received data signals: a) ideal eye with ISI.

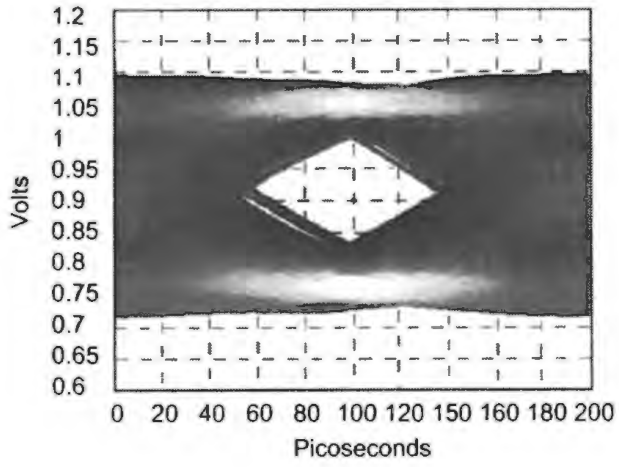


Figure 9.13 Eye diagrams for 5Gbps received data signals:
b) ideal eye with ISI, RJ, SSO, DCD, and clock jitter.

A compromise of source synchronous and CRC systems is shown in Figure 9.14. A continuous clock is still sent source synchronous with data (*DQ*) from the memory controller (TX), although they may run at different rates. To reduce routing and pin capacitance, data receiver or latch, in this case, is placed close to the *DQ* pad. Clocks at the capture latch are re-aligned by a DLL/PLL with reference to the external clock. Quarter-rate clocks can simplify the design of the CDN, whereas internal multi-phase clocks still keep up with the data rate or unit interval (*UI*). A set of sense amplifiers shown in Figure 9.11 can function as an input receiver (*RX*) sampled by internal multi-phase clocks. The external clock can be edge aligned with the data, while the data eye is centered on internally distributed clocks during initialization or training. A timing diagram drawn in Figure 9.15 describes the relationship of the clocks.

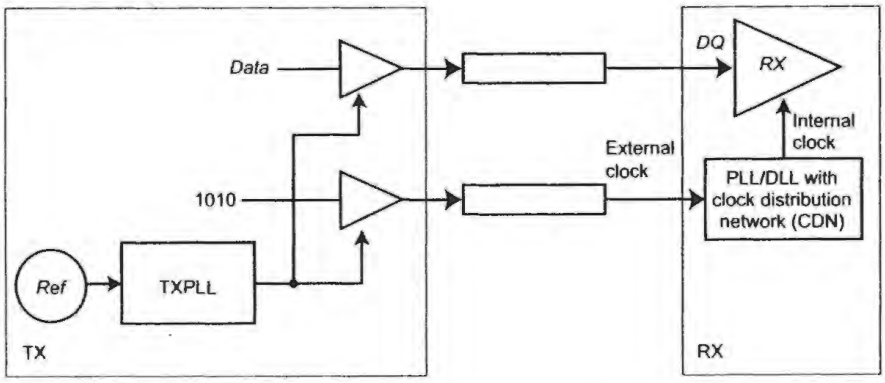


Figure 9.14 A PLL/DLL-based transceiver with clock alignment.

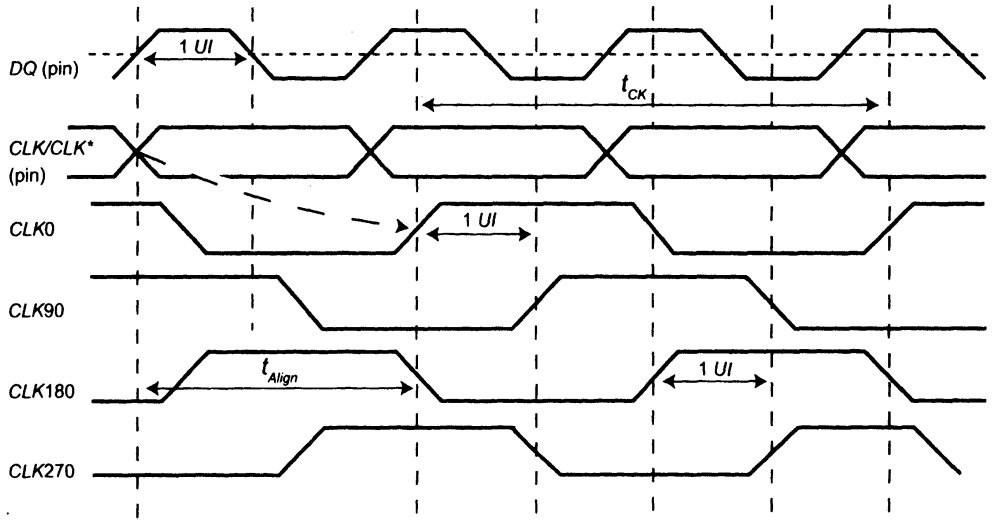


Figure 9.15 Timing diagram for input clock alignment.

In Figure 9.15, *DQ* represents incoming data at pins, while *CLK* and *CLK** are differential source synchronous clocks running at one quarter of the data rate. One clock cycle (t_{CK}) equals four unit intervals (*UI*s). *CLK*0, *CLK*90, *CLK*180, and *CLK*270 represent internal, four-phase clocks, one *UI* apart from each other. Referred to the *CLK/CLK**, *CLK*0 is delayed by t_{Align} due to the input PLL/DLL and the clock trees. Although only a 2-*UI* delay is shown for t_{Align} in the timing diagram, in reality more than 6-*UI* delay should be considered for the overall timing budget, depending on the data rate and circuit topologies. The t_{Align} parameter can be defined as

$$t_{Align} = t_{Rx} + t_{Adj} + t_{Tree} = n \cdot UI \tag{9.1}$$

where t_{Rx} is the delay for clock receiver; t_{Adj}, for PLL/DLL delay; and t_{Tree}, for CDN. The receive clock path delay can also be rounded up in terms of unit intervals (*UI*). Therefore, the clock edge used to capture the data is *n*-bit-shifted compared to the edge that launches it at the transmit site (TX). When the clock and data are out of phase, such *n·UI* delay amplifies jitter at certain noise frequencies. To reduce the impact of the *n·UI* input path delay, clock path circuits should be insensitive to PVT variations and efforts should be made to minimize t_{Align}. Step size and adjustment range also steer the timing circuit design toward an analog approach.

One problem associated with traditional CMOS logic is its sensitivity to PVT variations, especially to variations in process and supply voltages. A plot of delay versus PVT corners for two standard NAND gates is shown in Figure 9.16. The delay can vary from 103 ps (fast process-FF, 1.5 V, and

0°C) to 268 ps (slow process-SS, 1.2 V, and 120°C), which is a 160% increase. If these CMOS buffers were used in the critical timing path, like digital DLL, or clock trees, the accumulated jitter would be so great that clock alignment would be impossible.

Another problem with CMOS buffers is duty-cycle distortion and cross talk, which can severely impact signal quality. Common-mode noise injected from the reference voltage, power supplies, and substrate introduces dynamic jitter and requires careful design for power bussing, shielding, and isolation. The trade-offs for better signal quality at higher speeds, though, are excessive power and area. An alternative logic style must be investigated to mitigate these problems.

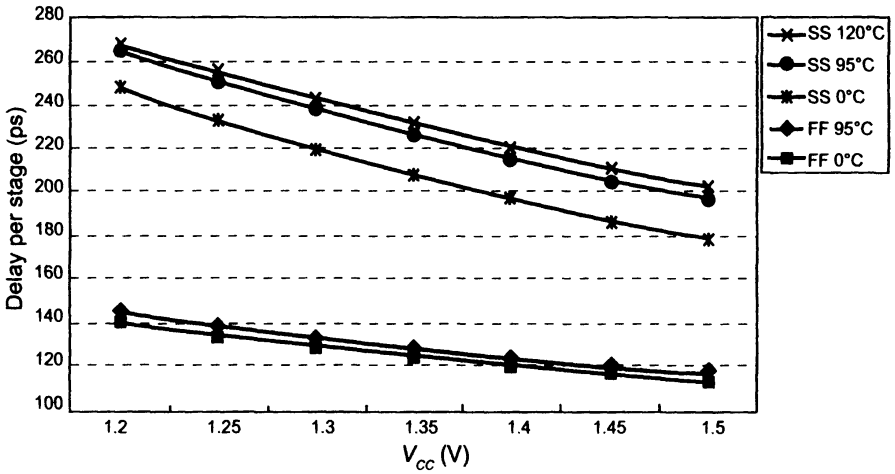


Figure 9.16 CMOS buffer delay versus PVT corners.

9.6 CURRENT MODE LOGIC (CML)

Current mode logic (CML), shown in Figure 9.17, is also referred to as MOS CML (MCML). A fully differential CML buffer in Figure 9.17a is basically a differential pair with resistive loads (R_D) and constant bias current (I_{tail}). Differential signaling makes CML much more immune to common-mode noise. Instead of swinging from rail-to-rail, CML outputs swing (in Figure 9.17a) only from V_{DD} to $V_{DD} - R_D \bullet I_{tail}$. The small swing not only makes CML fast, but also reduces electromagnetic interference (EMI) and cross talk.

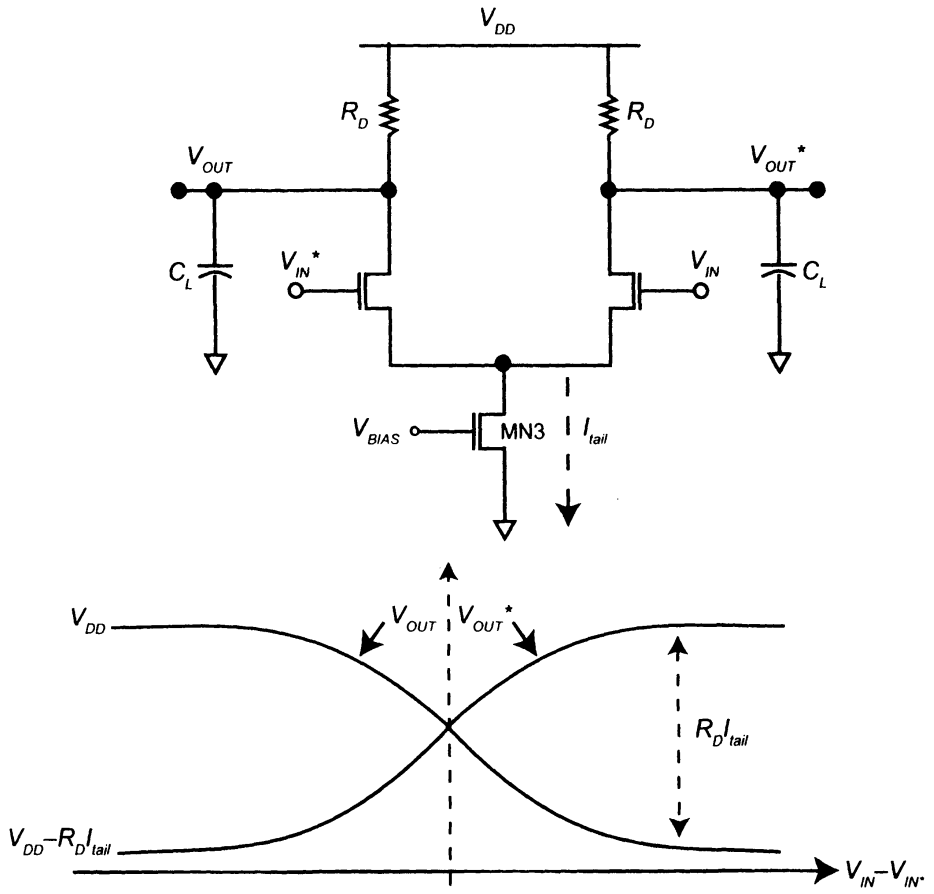


Figure 9.17 Current-mode logic: a) CML buffer and its I-V characteristics.

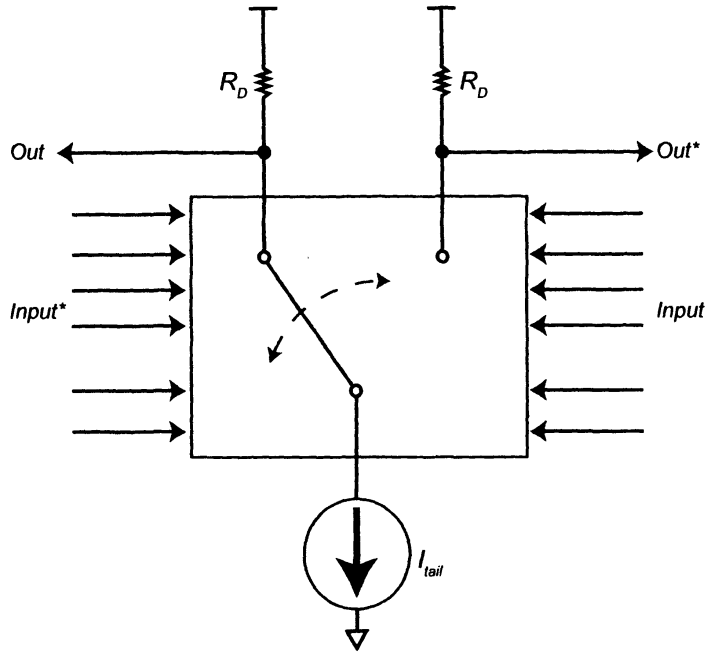


Figure 9.17 Current-mode logic: b) MOS current-mode logic.

With clock frequencies higher than 2 GHz, the power consumption of CMOS logic becomes unmanageable, while the power consumption of CML remains constant across the entire frequency range [5]. Delay (D), power (P), and energy (E) products between CML and CMOS are tabulated in Table 9.1 [5], where N is the logic depth of the critical path, I is the constant tail current for CML, k is the transconductance of MOS devices ($\mu C_{ox}W/L$), C is the load capacitance, Δ V is the output swing, V_t is the threshold voltage, and α is a parameter approximately 1.3.

It is interesting to see that the propagation delay of CML logic has no relationship with supply voltage (V_{DD}). If bias current (I) and voltage swing are kept constant over PVT corners, a constant delay can be achieved, which keeps PVT sensitivity at a minimum. Now, as an example, we'll explore the design of a clock distribution network (CDN) using both CML and CMOS buffers.

CDN is very important for *I/O* circuit design. Its primary goal is to maintain the same propagation delay across all destinations under PVT variations. With the input clock alignment depicted in Figure 9.14, a CDN is implied for distributing multi-phase clocks to various *DQ* and *C/A* latches. Minimizing delay and sensitivity through the tree is essential for this scheme to work.

Table 9.1 Delay, power, and energy products comparison.

CMOS	CML
$D = \frac{N \times C \times V_{DD}}{\frac{k}{2} \times (V_{DD} - V_t)^a}$	$D = NRC = \frac{N \times C \times \Delta V}{I}$
$P = N \times C \times V^2_{DD} \frac{1}{D}$	$P = N \times I \times V_{DD}$
$P \cdot D = N \times I \times V^2_{DD}$	$P \cdot D = N^2 \times C \times \Delta V \times V^2_{DD}$
$E \cdot D = N^2 \times 2 \times \frac{C^2}{k} \times \frac{V^2_{DD}}{V_{DD} - V_t}$	$E \cdot D = \frac{N^3 \times C^2 \times V_{DD} \times \Delta V^2}{I}$

A simulation setup for both CMOS and CML clock trees is shown in Figure 9.18. Two buffers drive two parasitic *RC* models (PRC) for metal routes, each with a distance of 1,500 μm, a width of 0.5 μm, and a space of 1 μm in an 80 nm DRAM process. A resistor is used for a CML load with improved bandwidth. The CML buffer is biased to get a constant swing of approximately 400 mV at 1.35 V_{DD}. For CMOS inverters, the width of PMOS is 50 μm and for NMOS, 25μm. A minimum length is applied to all devices except the tail NMOS (MN3 in Figure 9.17a) for the CML buffer, which has twice the minimum length. The input clocks (CLK_{IN} and CLK_{INB}) are differential and run at 1 GHz. Rise and fall times for the CML and CMOS buffers are 135 ps and 108 ps, respectively, to maintain reasonable duty cycle and signal quality. The simulation is set up at a typical process with PRC, 1.35 V, and 95°C corners.

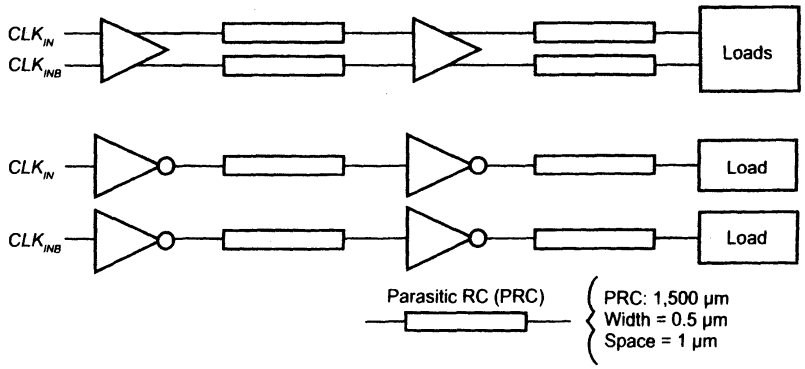
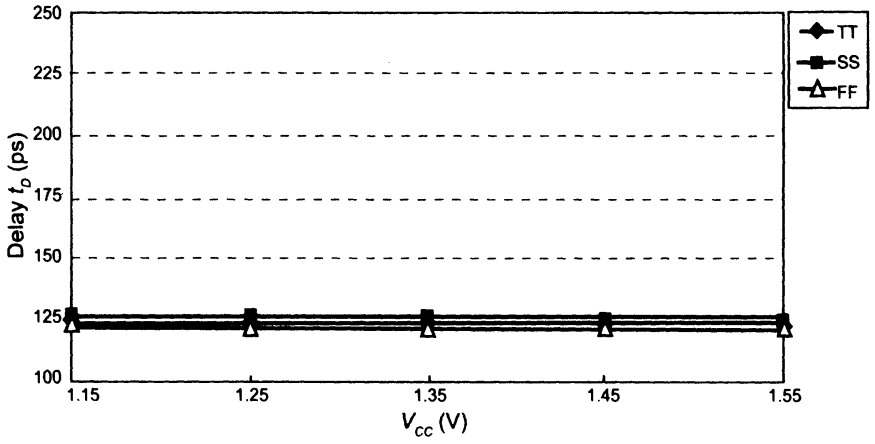
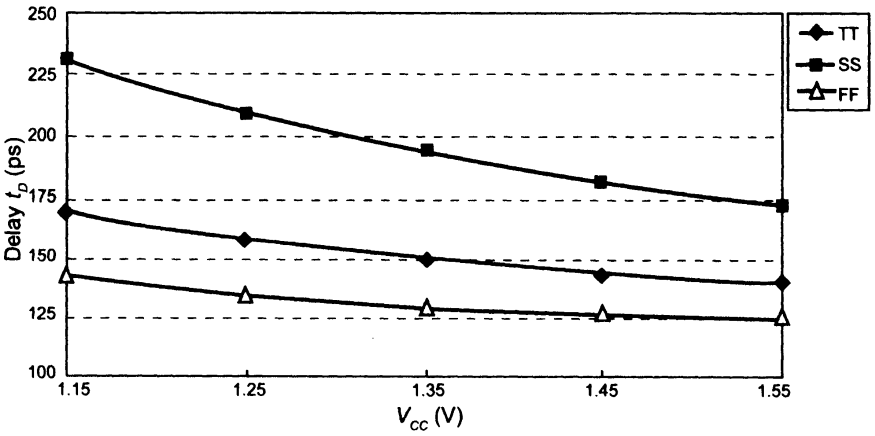


Figure 9.18 Simulation setup for CML (top) and CMOS (bottom) clock trees.

Simulation results for voltage and process sensitivities are shown in Figure 9.19, where the difference of CMOS propagation delay (t_D) is six times greater than it is for CML buffers. The worst-case absolute delay is also halved for the CML tree. In Figure 9.20, temperature sensitivity is comparable between the two designs with approximately 26 ps delay difference per 100°C.



a)



b)

Figure 9.19 a) CML and b) CMOS clock trees delay time under process and voltage variations @ 95°C.

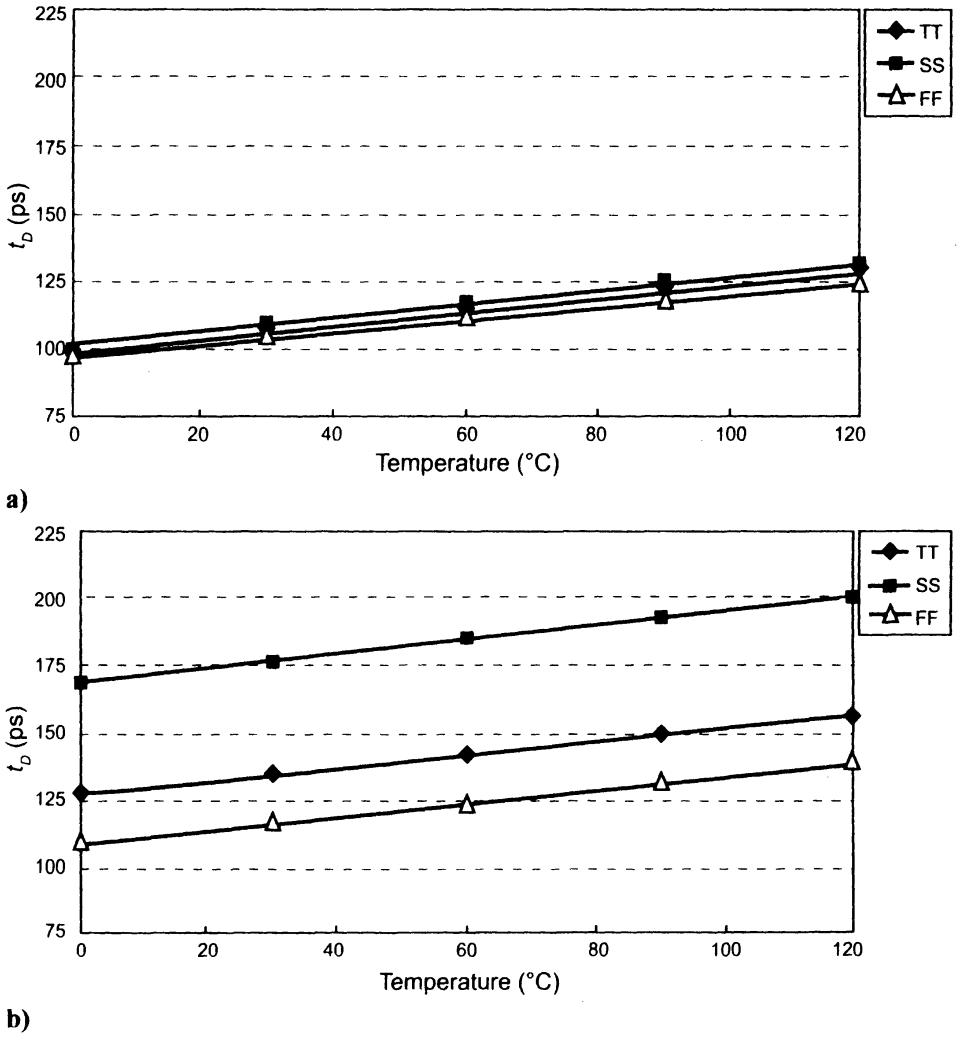


Figure 9.20 a) CML and b) CMOS clock trees under process and temperature variations @ 1.35V.

Indeed, the CML clock tree is immune to voltage and process variations, as predicted in Table 9.1. Near-constant delay makes the CML buffer ideal for designing a critical timing path, especially when using a clock alignment scheme. (Timing circuits using the CML buffer are discussed in Chapter 11.) The CML buffer is also a natural fit for clock receiver circuits, as shown in Figure 9.3b. Extensive use of CML logic makes input circuits capable of handling much higher data rates and more aggressive scaling.

So, what are the obstacles preventing designers from using CMLs in DRAM timing paths as opposed to using simple CMOS inverters?

1. A more complex circuit topology and differential signaling require a larger area and more restricted matching.

2. Dedicated bias generation and distribution is essential to maintain a constant delay over PVT variations. A low-voltage bandgap circuit can generally fit in this role.
3. A signal conversion circuit (differential-to-single-ended translator) is needed to interface with CMOS logic. Duty-cycle distortion and PVT sensitivity may be exposed at this stage. Multi-phase clock generation or duty-cycle correction is inevitable and complicates the design.
4. Increased static power is another concern. A CML buffer consumes current regardless of whether it is switching or not. For the example described, the average current consumed by the CML tree is approximately 6.8 mA, compared to 2.2 mA for a CMOS tree. On the other hand, the current profile for a CML tree is fairly constant with small spikes, which translates to less switching noise and quieter buses.

Looking forward, as speed in memory systems continuously accelerates, memory interface design becomes formidable for timing closure. Teamwork is necessary for analyzing signal integrity, packaging, transmission line, power, and clock delivery. The critical timing path must be identified and modeled early in the design phase followed closely by a timing analysis. Performance and cost must be balanced when designing high-speed *I/O* circuits.

REFERENCES

[1] Micron technical note, TN-47-06, "Updated JEDEC DDR2 specifications."

[2] Micron technical note, TN-47-02, "DDR2 offers new features/functionality introduction."

[3] Micron technical note, TN-47-11, "DDR2 differential DQs introduction."

[4] I. E. Sutherland, R. F. Sproull, and D. Harris, *Logical Effort: Designing Fast CMOS Circuits*. San Diego, CA: Academic Press, 1999.

[5] B. Razavi, *Monolithic Phase-locked Loops and Clock Recovery Circuits, Theory and Design.* Piscataway, NJ: IEEE Press, 1996.

[6] A. Tanabe, M. Umetani, I. Fujiwara, K. Kataoka, M. Okihara, H. Sakuraba, T. Endoh, and F. Masuoka, "0.18-μm CMOS 10-Gb/s multiplexer/ demultiplexer ICs using current mode logic with tolerance to threshold voltage fluctuation," *IEEE Journal of Solid-State Circuits*, vol. 36, pp. 988–996, June 2001.

[7] M. Mizuno, M. Yamashina, K. Furuta, H. Igura, H. Abiko, K. Okabe, A. Ono, and H. Yamada, "A GHz MOS adaptive pipeline technique using MOS current-mode logic," *IEEE Journal of Solid-State Circuits*, vol. 31, pp. 784–791, June 1996.

Chapter
10

Output Circuit Paths

Serving as the driving engine of the memory data path, the output path multiplexes the internal data and sends it out in a timely manner, based on *CAS* latency and burst length. Timing control, which is addressed in Chapters 11 and 12, is critical to determining when to send data out after receiving a Read command. Besides data, strobes similar to those found on the input paths, can be sent out source-synchronously (SS) with data to the memory controller (MC). SS strobes can greatly simplify data capture at the controller site. And, strobes can be bi-directional like data or unidirectional like clocks, in which separate Read and Write strobes are needed. The latter can improve the Read/Write turn-around time, albeit at a higher pin count. Trade-offs between performance and cost must be considered when designing the *I/O* interface.

As *I/O* speeds push toward and beyond the gigahertz range, signal integrity (SI) becomes increasingly important to *I/O* design. Transmission line, bus structure, termination strategy, load capacitance, inductance, and packaging technology all directly or indirectly impact *I/O* design. To meet design targets and *I/O* specifications, a variety of issues, such as impedance calibration, simultaneous-switching noise (SSN) reduction, slew-rate control, signal return path analysis, and electrostatic discharge (ESD), must be examined. In this chapter, we start by exploring these issues and their implications on design. Then we move to examine high-speed, parallel-to-serial conversion, and the interface between the internal data path and output drivers. Finally, we conclude by describing some features of emerging memory *I/O*.

10.1 TRANSMISSION LINE, IMPEDANCE, AND TERMINATION

Signals can be carried via wires, coaxial cables, or microstrip or stripline transmission lines. For signals transmitted on board at higher frequencies, PCB (printed-circuit board) traces can no longer be treated as just wires: they become radio frequency (RF) transmission lines. Signal integrity can be affected in two ways. The first is transmission line (TL) effects, such as ringing and reflections. The second is the interaction with other elements, such as cross talk and noise on the power plane. Drive impedance control and on-die termination (ODT) tend to reduce reflections, slew-rate control can reduce ringing, and signal return-path analysis can help to reduce cross talk and noise on the power planes (return path).

Let's start the discussion about signal integrity with basic transmission line modeling and characterization. A simple model of a transmission line is shown in Figure 10.1. The signal trace is on the top (shaded) with a return path at bottom. The trace can be divided into unit length portions, each with identical electrical characteristics.

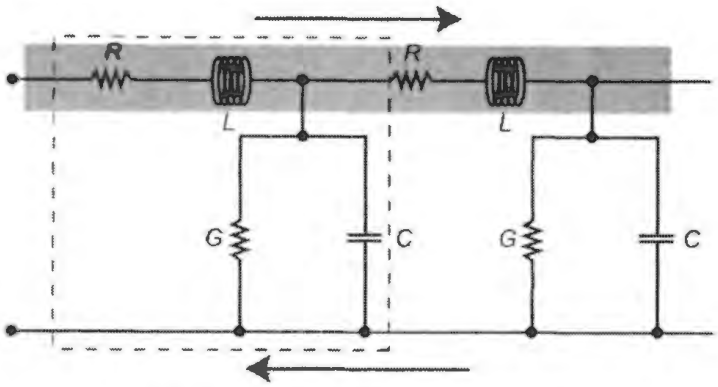


Figure 10.1 Transmission line model.

The single-most important feature of transmission lines is characteristic impedance *Zc*. *Zc* as derived from this model is given in Equation (10.1), where *R*, *G*, *C*, and *L* are the resistance, conductance, capacitance, and inductance per unit length, respectively, and *f* is the signal frequency:

$$Z_c = \sqrt{\frac{R + j2\pi fL}{G + j2\pi fC}} = \sqrt{\frac{R + j2\pi fL}{j2\pi fC}} \tag{10.1}$$

As shown, the characteristic impedance is frequency dependent and specified in ohms. The conductance per unit length *G* is due to loss through the dielectric material separating the two traces. For nearly all types of conductors, it is safe to assume that $G=0$ at frequencies below the gigahertz range. Higher frequencies introduce some AC conductance, which may not

be negligible [1]. The resistance R can also be assumed to be zero, except at high frequencies where skin effect comes into play. Therefore, with assumptions that both G and R are zero, the transmission line could be considered *lossless*, and the characteristic impedance, in this case Z_0, would become frequency independent.

$$Z_0 = \lim_{f \to \infty} Z_c = \sqrt{\frac{L}{C}}$$

(10.2)

The skin effect is a phenomenon in which resistance increases as frequency increases due to the fact that the signal is pushed into a very thin layer of the conductor surface. The thickness of this layer is inverse proportional to the square root of frequency *(f)*. The skin resistance (R_S) of the transmission line is given in Equation (10.3) along with AC resistance (R_{AC}), where μ is permeability, σ is conductivity, and W and l are the width and length of the conductor, respectively.

$$R_S = \sqrt{\frac{\pi f \mu}{\sigma}}, \; R_{AC} = \sqrt{\frac{R_s l}{W}}$$

(10.3)

At the frequency that some of the point-to-point DDR systems are running, the skin depth is down to a few tenths of a mil (0.0002 inches) for the conducting layer, which, in turn, increases the AC resistance. The skin effect is the main reason that the resistance generally cannot be ignored in calculating the characteristic impedance of a transmission line. The high frequency loss due to skin effect can be compensated through equalization to reduce inter-symbol interference (ISI).

When a transmission line is terminated at both ends with its characteristic impedance (Z_0), the line is said to be matched, as shown in Figure 10.2. Being matched means that no reflections should occur on the line, which is desirable for high-frequency signal transmission. Without termination and impedance control, reflections on the line distort the signal, which, if the distortion is large enough, prevents reliable detection by the receiver.

In Figure 10.2, at the driver side, the matched impedance (Z_0) is equal to the output driver impedance (e.g., 18 ohm) plus the termination impedance (e.g., 45 ohm). On-die termination (ODT) can provide a matched impedance on the receiver end. An ODT pin is added to DDR2 parts, as shown in Figure 10.3, to control the termination impedance as needed. For DDR2, ODT is required only for the data, data mask, and strobes. For clock

frequencies passing into the gigahertz range, as in GDDR4 parts, ODT is also required for command and address signals.

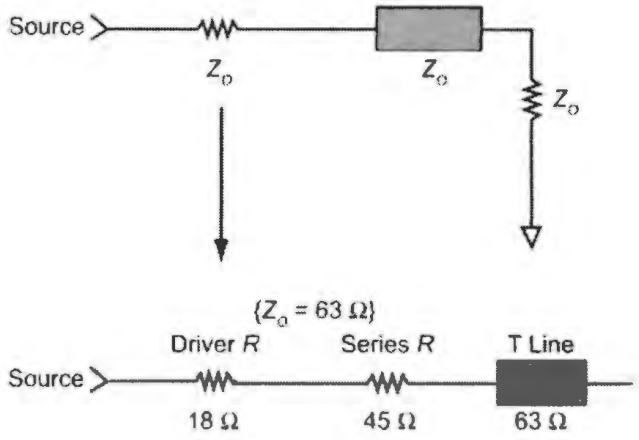


Figure 10.2 Matched transmission line (two-end termination).

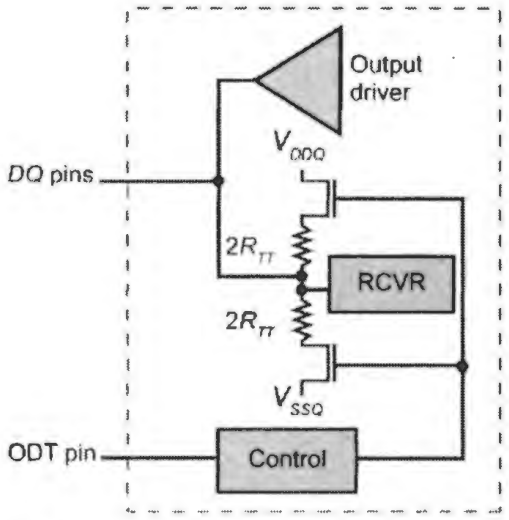


Figure 10.3 Functional diagrams for on-die termination (ODT).

Examples of both Read and Write ODT configurations in a dual-ranked module system are shown in Figure 10.4. Compared to the stub series terminated logic (SSTL) bus of Figure 5.2, a bus with ODT demonstrates far better signal quality for both Read and Write operations. Eye diagrams with data running at 555Mbit/s for both configurations are shown in Figure 10.5. Obviously, the ODT system produces increased voltage margin, larger data eyes, and improved slew rate even under heavily loaded conditions.

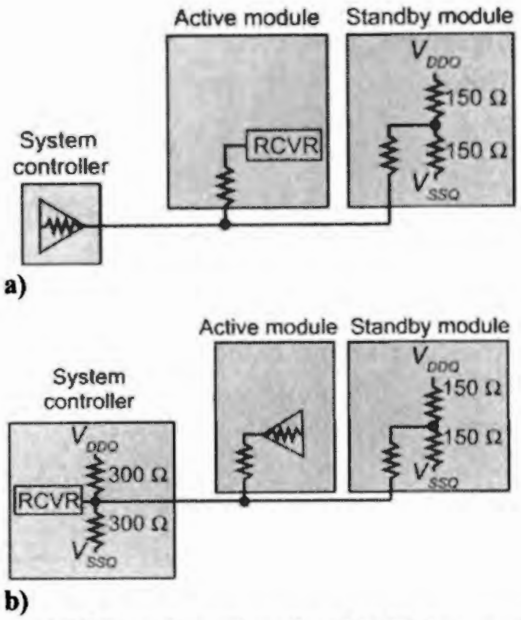
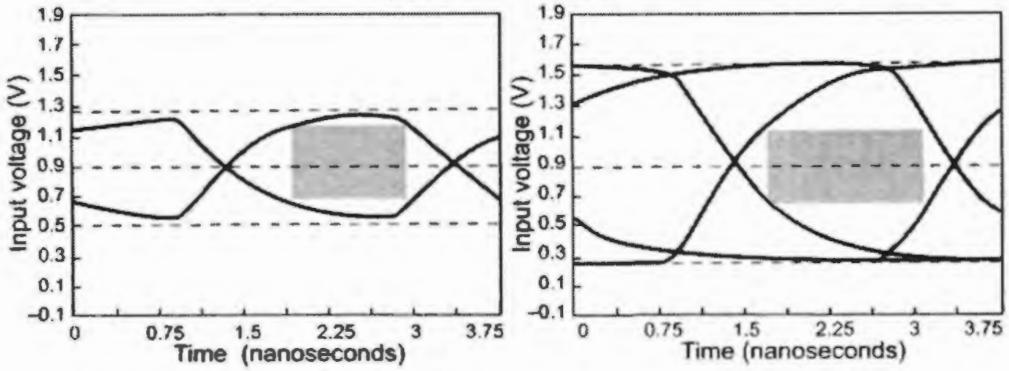
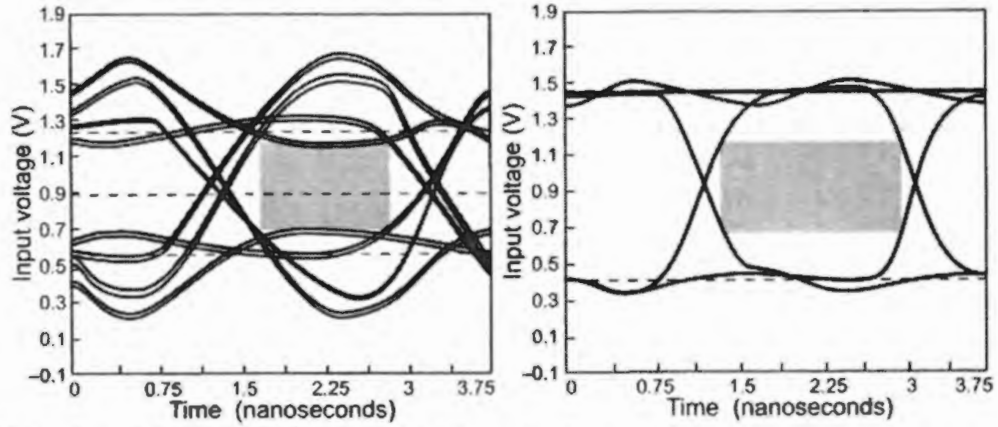


Figure 10.4 Typical ODT configurations for DRAM (two modules populated) a) Writes and b) Reads.



a) Writes with SSTL (left) and ODT (right) terminations.



b) Reads with SSTL (left) and ODT (right) terminations.

Figure 10.5 Comparisons between SSTL and ODT for dual-ranked module.

10.2 IMPEDANCE CONTROL

So, how do we control impedance in order to match the characteristic impedance of the transmission line? An impedance calibration circuit is generally the answer.

A typical calibration system using an external precision resistor (R_{ZQ}, e.g., 240 Ω ± 1%) is shown in Figure 10.6. R_{ZQ} can be set to m times Z_0 to improve accuracy and reduce current during calibration. For example, if Z_0 = 40 Ω and R_{ZQ} = 240 Ω, m = 6, the driver can be divided into six identical legs, each calibrated to 240 Ω and placed in parallel to achieve an equivalent impedance of 40 Ω. This configuration is also helpful for ODT and drive strength control.

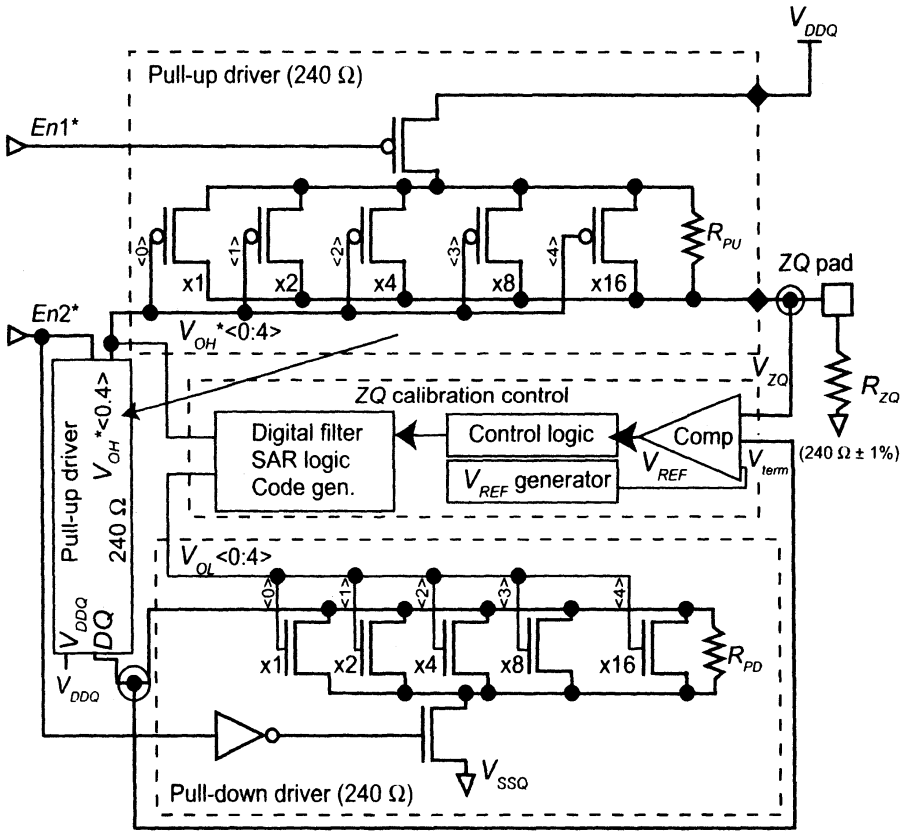


Figure 10.6 Impedance calibrations using external resistor R_{ZQ}.

On-die impedance calibration, also known as ZQ calibration, uses a dedicated ZQ pin. During initialization, the calibration usually starts after the system is powered up and the clock is stable. The resistance of one leg of the output driver, i.e., the pull-up driver in Figure 10.6, with a combination of transistors and resistors, compared to the external resistor R_{ZQ}. The R_{ZQ} is placed between the ZQ pin and ground (V_{SSQ}). The transistors are binary weighted to provide a certain resolution, i.e., 5–10% of Z_0, and a tun-

ing range depending on process, voltage, and temperature (PVT) variations. In Figure 10.6, five-bit digital codes (V_{OH*} or V_{OL}) generated by ZQ control circuits are used to select or deselect different pull-up or pull-down transistors. A total of 32 steps are available with a quantization error of one LSB (least significant bit). The resistor (R_{PU}/R_{PD}) in parallel with the transistors improves the linearity of the effective impedance. R_{PU} / R_{PD} must be greater than R_{ZQ} to allow for transistor tuning. Due to large current flow through these drivers, wider metal and sufficient vias or contacts must be used in the layout to avoid metal electromigration problems.

The first step of the calibration sequence starts with the signal $En1*$ transitioning LOW. The voltage (V_{ZQ}) is tapped and fed in to the ZQ circuit for comparison. An internal reference voltage (V_{REF}) is generated and connected to the other input of the comparator. A simple resistor divider can be used as the V_{REF} generator. Its voltage level can be adjusted if necessary. For 50% V_{DDQ} calibration (V_{REF} = 0.5 V_{DDQ}), the calibrated pull-up impedance $(R_{Cal,UP})$ is equal to R_{ZQ} with one LSB error (R_{ULSB}). The next step is to calibrate the pull-down impedance by using another pull-up driver and the generated V_{OH} codes (V_{OH*} <0:4>). To start this step, signal $En2*$ transitions LOW, in this case. A voltage (V_{term}) tapped between the pull-up and pull-down drivers is fed back to the ZQ circuit for comparison. The final pull-up impedance ($R_{Cal,UP}$) and pull-down impedance $(R_{Cal,DN})$ are given in Equation (10.4) and Equation (10.5), respectively, with regard to the driver multiplication factor m, pull-down LSB error R_{DLSB}, and external resistance R_{ZQ}.

$$R_{Cal,\,UP} = (R_{ZQ} \pm 1 \bullet R_{ULSB})/m \tag{10.4}$$

$$R_{Cal,\,DN} = R_{Cal,\,UP} \pm (1 \bullet R_{DLSB})/m = (R_{ZQ} \pm R_{ULSB} \pm R_{DLSB})/m \tag{10.5}$$

Compared to the pull-up impedance $R_{Cal,UP}$ in Equation (10.4), the pull-down impedance ($R_{Cal,DN}$) may have much greater calibration error. Given the previous examples, R_{ZQ} = 242.4 Ω (+1%), m = 6, and R_{ULSB} = R_{DLSB} = 6 Ω, then $R_{Cal,UP}$ = 41.4 Ω, and $R_{Cal,DN}$ = 42.4 Ω, which have a mismatch of 3.5% and 6%, respectively. The 6% error may not be acceptable for a high-speed interface, especially for open-drain configurations where only NMOS pull-down drivers are used. Improving the pull-up impedance calibration can help to tighten up the specs. Instead of five bits, one extra bit can be added for the pull-up transistors and the LSB reduced to

3 Ω, for example. Therefore, the calibrated pull-up and -down impedances would be 40.9 Ω and 41.9 Ω, respectively, with a maximum error of 4.75%. Longer calibration time is one drawback of such an improvement.

Depending on the searching algorithm, the *ZQ* calibration could take thousands of cycles during initialization. A binary search algorithm using successive-approximation registers (SAR) is usually fast and well suited for *ZQ* calibration. A diagram for a four-bit binary search algorithm is shown in Figure 10.7, with the code initially set to "1000" (MSB, most significant bit). A maximum of three steps are needed to complete the search, compared to eight steps without it.

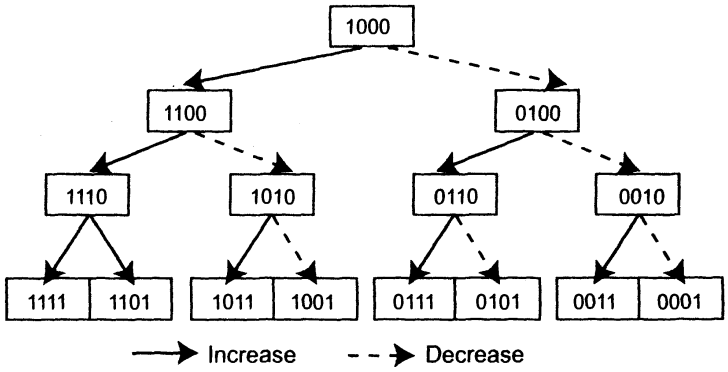


Figure 10.7 Diagram for binary search algorithm (4-bit).

Usually, an analog comparator can be used to discriminate the difference between V_{ZQ}/V_{term} and V_{REF}. A large input device may be used for the pre-amplifier to reduce offset and increase gain. A decision circuit with positive feedback and an output buffer can be used to improve the sensitivity and generate digital control signals (i.e., "Increase" or "Decrease") for digital filters. Digital filters are used to ensure the right direction for adjustment under noisy environments. An up/down counter can be used to log the decisions made by the comparator. After certain continuous "Increases" or "Decreases," a true adjustment is made to the outputs, and the comparator is reset for the next comparison. The initial calibration time ($t_{Cal,ini}$) for either a pull-up or pull-down impedance can be approximated as

$$t_{Cal,\,ini} = (n-1) \cdot x_F \cdot t_{SCK} \tag{10.6}$$

$$t_{Cal,\,ini} = 2^{(n-1)} \cdot x_F \cdot t_{SCK} \tag{10.7}$$

where n is number of bits, x_F is the digital filter factor for continuous counts, and t_{SCK} is the sampling clock period for the digital filter. The calibration starts at midpoint. Equation (10.6) represents a SAR binary search whereas Equation (10.7) is for counting step-by-step. The speed of the comparator can really slow down initial ZQ calibration at certain PVT corners. Time-out and lock detect circuits must be included in the filter in case the input voltage is close to V_{REF} and the comparator fails to make a decision within a predetermined period.

An alternative way to make a decision is by using a voltage-to-time conversion circuit (VTC), as shown in Figure 10.8. The input voltage of the VTC is converted to a current and charges or discharges an internal capacitor. When the trip point of the output buffer is reached, the output of VTC is switched to create a transition. When two identical VTCs are used, with one hooked to V_{REF} and the other to V_{ZQ} or V_{term}, the voltage difference can be converted to a timing difference. A phase detector (PD) can then make the judgment and send the results to the digital filter. Quick turn-around time makes this attractive for some applications.

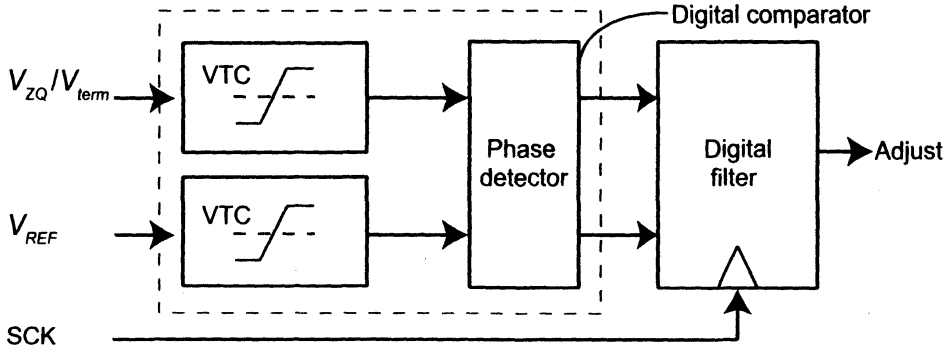


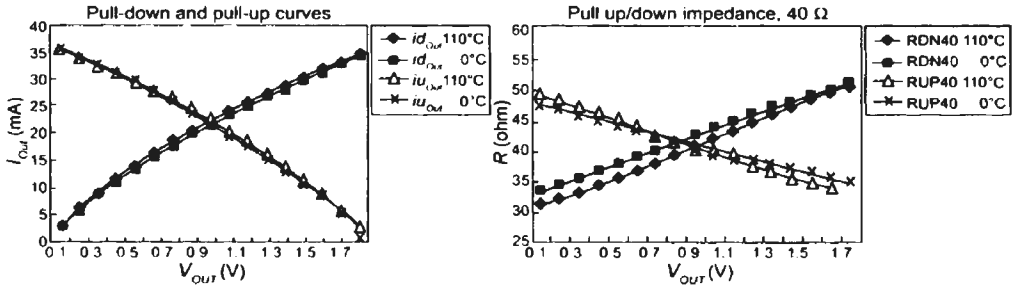
Figure 10.8 ZQ calibrations using digital comparator.

Due to voltage and/or temperature (VT) drift, on-die impedance may require readjust for better matching after initial calibration. Such ZQ updates may take a shorter time by changing only a couple of LSBs. This event can be controlled by an up-down counter and triggered by on-die timing circuits, like a delay-locked loop (DLL). When the DLL makes continuous timing adjusts after initial lock, a potential impedance mismatch caused by VT drift can occur. The information is fed back to the ZQ calibration circuit to start the ZQ update. Periodic updates are also possible depending on DRAM operations. For instance, ZQ updates can be scheduled during DRAM Auto Refresh. Besides internal updates, an external calibration scheme controlled by a memory controller is possible.

Layout can be another challenge for precise impedance control. Replicated drivers should be used for ZQ calibration as well as for ODT. The

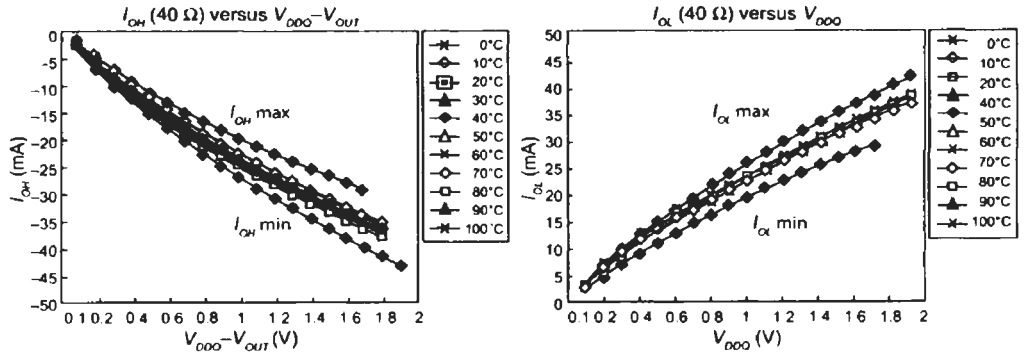
same orientation and power buses should be applied to all drivers. To avoid the physical limitations caused by tuning transistors, a minimum-sized transistor may need to be stacked (connected in serial) to provide better resolution. Because of mobility difference, a different number of legs can be used in parallel for pull-up or pull-down calibrations. For instance, two pull-up legs versus one pull-down leg results in 120 Ω $R_{Cal,DN}$ compared to 240 Ω $R_{Cal,Up}$. It is also important to isolate critical analog circuits such as the comparator from the digital control logic. To minimize noise and cross talk, analog signals V_{REF}, V_{ZQ}, and V_{term} must be shielded and de-coupled. To reduce errors, V_{ZQ} and V_{term} should be tapped in the middle of the connection (small circles in Figure 10.6). Finally, layout extraction and simulation are essential to ensure that the design meets the specs across the PVT corners.

Now, following design and layout, let's talk about how to characterize the output impedance. The output drive impedance can easily be approximated with the dV/dI curves during Test and Characterization. The curve is obtained by sweeping the V_{out} from 0 to V_{DDQ} and measuring I_{out}. An example of GDDR3 output impedance versus temperature is plotted in Figure 10.9, with both simulated and silicon data running at V_{DDQ} = 1.8 V and at typical process corners. To extract the driver impedance from the curves, divide the delta V by delta I in the output driver's operating range. The pull-up impedance shows tighter spread than the pull-down impedance in a two-step ZQ calibration. Nonlinearity is also observed when V_{out} is approaching 0 for pull-down impedance or V_{DDQ} for pull-up impedance.

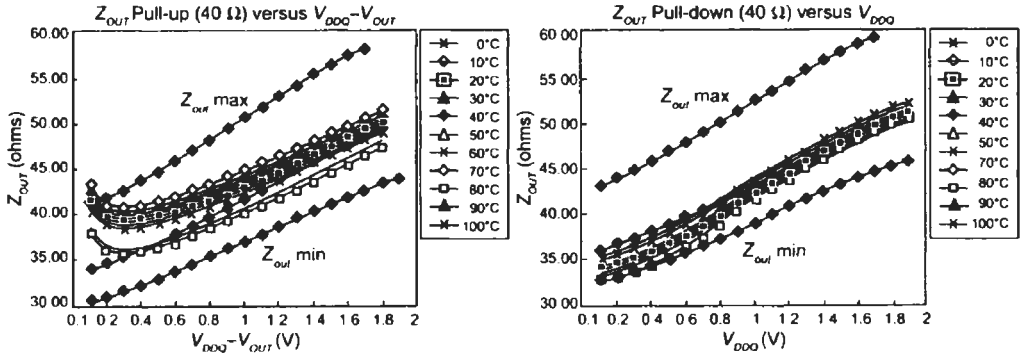


a) Simulation data for output I-V curves and driver impedance.

Figure 10.9 GDDR3 output I-V curves and impedances versus temperature.



b) Silicon data for output I-V curves.



c) Silicon data for output impedances.

Figure 10.9 GDDR3 output I-V curves and impedances versus temperature.

Other impedance control schemes are reported in [2–3] with different termination circuits, as shown in Figure 10.10. Passive resistors are binary weighted for both pull-up (UPVR) and pull-down (DNVR) impedances. Similar two-step self-calibration is adopted for impedance and ODT controls. Seven-bit digital codes (pull-up and pull-down code) generated by the up-down counters select different impedances. Compared to transistor-only tuning, as shown in Figure 10.6, this scheme can achieve better linearity because there is a resistor in each tuning path. However, temperature and process variations of tuning resistors must be addressed. Matching and a potentially large area are the primary concerns.

Instead of digital-controlled calibration, an analog impedance control scheme is shown in Figure 10.11 [4]. The impedance is adjusted by changing the bias current, which is controlled by an op-amp. Proper compensation (*Rc* and *Cc*) stabilizes the op-amp. The benefits are good linearity and continuous impedance adjustment.

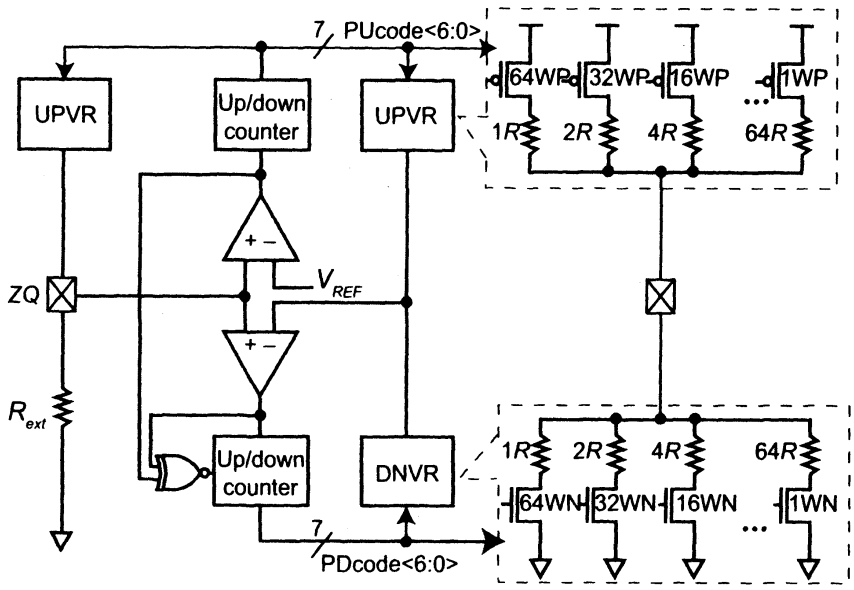


Figure 10.10 Digital-controlled impedance calibration [2–3].

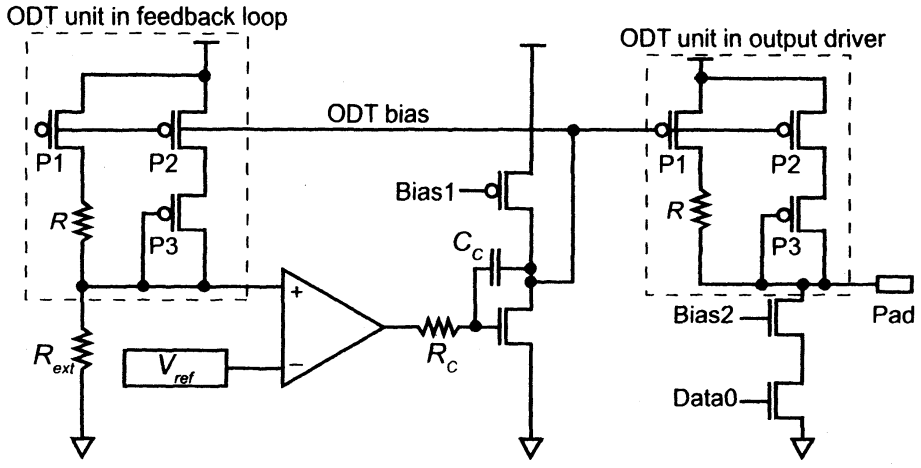


Figure 10.11 Analog impedance controls [4].

10.3 SIMULTANEOUS SWITCHING NOISE (SSN)

Simultaneous switching noise (SSN) is a serious concern for high-speed *I/O* design. SSN is referred to as *I/O* power supply noise generated by simultaneously switching output drivers. In order to drive data from the chip, output drivers must supply large currents, usually more than ten milliamps depending on the DQ supplies and Z_0. When there are numerous (N) simultaneous switching outputs (SSO) connected to the same pair of power and ground pins, a substantial discharge current (i_S) flows through the power supply bus causing voltage bounce (V_S), or SSN, given as

$$V_S = L_P \cdot \frac{di_S}{dt}$$

(10.8)

In Equation (10.8), a lumped effective inductance L_P models the parasitic components of the power supply bus, pad, bonding wire, and package. As N is increased, the voltage bounce increases, although the effect on driver performance is not linear but rather has a square-root relationship due to the gate-to-source voltage dependency.

There are several ways to reduce SSN as a result of high-speed operation. The first is to add more power and ground pins, which effectively reduces parasitics and improves noise immunity. A 136 FBGA (fine ball grid array) GDDR4 ball-out diagram is shown in Figure 10.12 with 22 V_{DDQ} pins and 24 V_{SSQ} pins dedicated to *I/O*s. Every two *DQ*s share a pair of V_{DDQ}/V_{SSQ}, which is essential for power delivery and SSN reduction. Separated power pins (i.e., V_{DD}/V_{SS} and V_{DDA}/V_{SSA}) also reduce noise coupling from the *I/O* supplies into other sensitive timing circuits, like the DLL and clock distribution network (CDN), for example. Power bus simulation (Chapter 13) can be useful for analyzing SSN with power grid and package models.

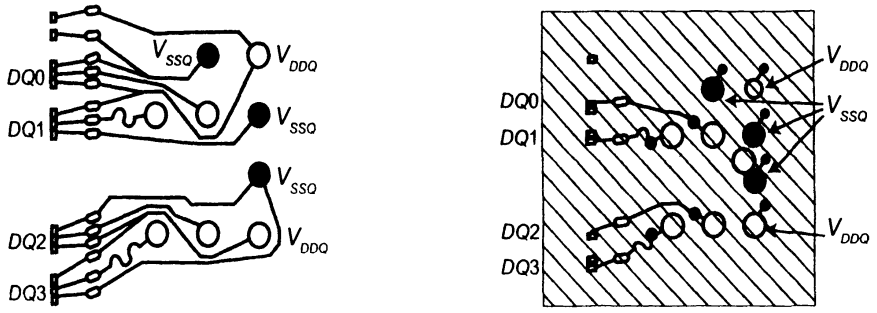
More advanced packaging techniques can further reduce effective L_P and efficiently reduced SSN. Substrate layouts for both single-layer and multi-layer packages are compared in Figure 10.13 for four *DQ*s and their associated power supplies. The black dots are for V_{SSQ} and the rest are for *DQ0-3* unless otherwise stated. Notice that the *DQ* traces are matched. Also, each *DQ* is sandwiched by power and ground, an effective way to reduce cross talk and inductance.

The measured impedances for these two packages (value normalized by dividing by $2\pi f$) are shown in Figure 10.14. The normalized impedance values allow us to easily compare the effective low-frequency inductance (in nH) of the two package designs. The resonance in the multi-layer package is due to the parallel resonance between plane capacitance and ball inductance. The resonance can be suppressed by adding a decoupling capacitor (C_{fat}) between power and ground. Based on the data, over 1nH reduction can be achieved with a multi-layer package.

8x17 balls (0.8X, 0.8Y mm pitch)

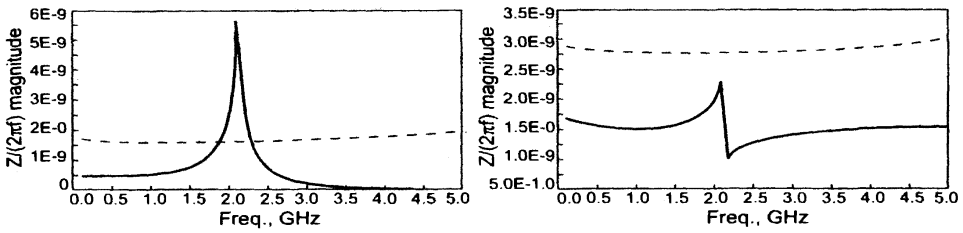
	1	2	3	4	9	10	11	12
A	VDDQ	VDD	VSS	ZQ	MF	VSS	VDD	VDDQ
B	VSSQ	DQ0	DQ1	VSSQ	VSSQ	DQ9	DQ8	VSSQ
C	VDDQ	DQ2	DQ3	VDDQ	VDDQ	DQ11	DQ10	VDDQ
D	VSSQ	WDQS0	RDQS0	VSSQ	VSSQ	RDQS1	WDQS1	VSSQ
E	VDDQ	DQ4	DM0	VDDQ	VDDQ	DM1	DQ12	VDDQ
F	VDD	DQ6	DQ5	VSSQ	VSSQ	DQ13	DQ14	VDD
G	VSS	VSSQ	DQ7	CAS*	CS*	DQ15	VSSQ	VSS
H	VDDQ	RAS*	CKE*	B0/A1	B1/A5	WE*	NC	VREF
J	VSSA	RFU	RFU	VREFC	VREFD /NC	CK*	CK	VSSA
K	VDDA	A10/A0	A12/A2	VSS	VSS	B2/A6	A5/A4	VDDA
L	VSS	VSSQ	DQ25	A11/A3	A9/A7	DQ17	VSSQ	VSS
M	VDD	DQ24	DQ27	VSSQ	VSSQ	DQ19	DQ16	VDD
N	VDDQ	DQ26	DM3	VDDQ	VDDQ	DM2	DQ18	VDDQ
P	VSSQ	WDQS3	RDQS3	VSSQ	VSSQ	RDQS2	WDQS2	VSSQ
R	VDDQ	DQ28	DQ29	VDDQ	VDDQ	DQ21	DQ20	VDDQ
T	VSSQ	DQ30	DQ31	VSSQ	VSSQ	DQ23	DQ22	VSSQ
V	VDDQ	VDD	VSS	SEN	RES	VSS	VDD	VDDQ

Figure 10.12 GDDR4 x32 136 FBGA ball-out diagram.

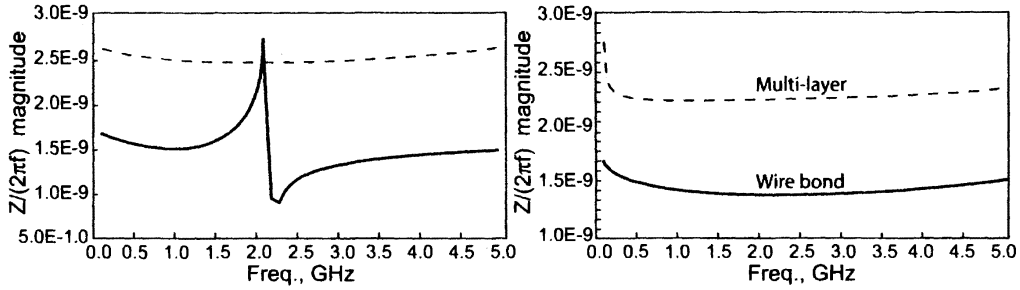


a) b)

Figure 10.13 Layout of a) wire bond and b) multi-layer.



a) Power-ground (left) and signal-power (right) normalized loop impedance.



b) Signal-ground normalized loop impedance without C_{fat} (left) and with C_{fat} (right).

Figure 10.14 Measured loop impedance for different packages.

To investigate the package impact on SSO, a simulation was set up based on the packages just described with the following:

- Open-drain drivers, with three *DQs* (*DQ1*, *DQ2*, and *DQ3*) drivers as aggressors and 1 *DQ* (*DQ0*) driver as victim.
- Aggressors run PRBS pattern (same phase among aggressors) and victim runs clock pattern.
- Data running at 1.333Gbps with a rise time of 300ps.
- 1000 mil transmission line (60 Ω characteristic impedance) with 60 Ω termination to V_{TT}.

The simulation results are shown in Figure 10.15. Timing jitter induced by SSO noise is reduced by 53% using the multi-layer package.

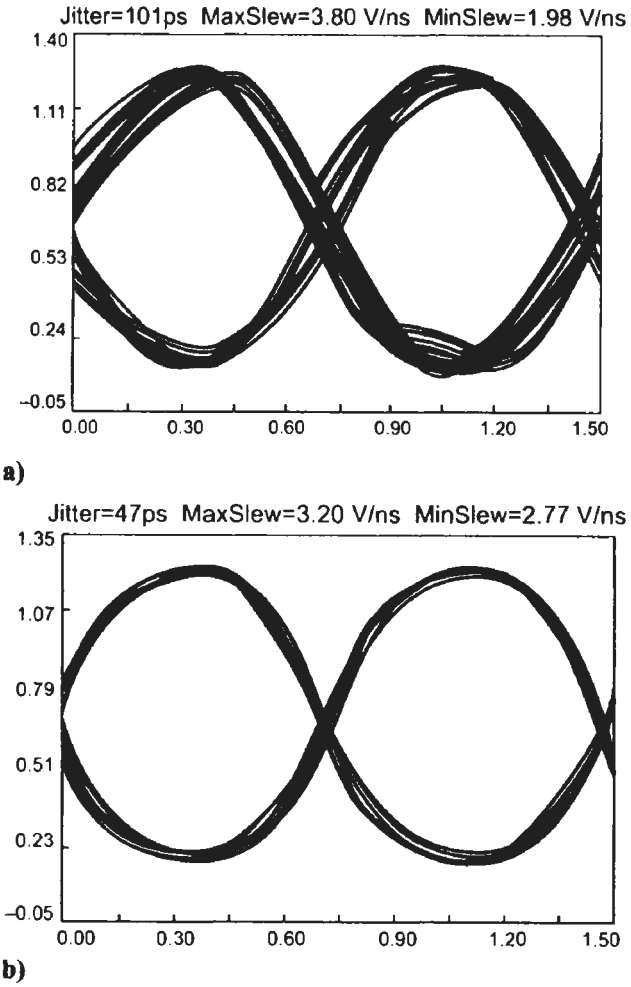


Figure 10.15)SSO simulation results for a 60 ohm load: a) wire bond package and b) multi-layer package.

Advanced package technologies reduce effective inductance at a higher cost. From a circuit point of view, slew-rate control is a cost-effective way to reduce SSN and output ringing. Slew-rate control is the technique of controlling large transient current spikes during signal switching, i.e., di_S / dt in Equation (10.8). As mentioned, the last stage of the output driver usually contains large-sized devices in order to drive the load (pin capacitance, in several pico-farad ranges). A tapered pre-driver is usually needed to scale up transistor sizes prior to the output driver. The delay of the pre-driver can be adjusted to avoid crowbar current and to achieve a certain slew rate from the output driver. The output driver can also be divided into several sub-buffers (legs), which are connected in parallel to achieve the desired impedance and drive strength. The distributed pre-drivers must be matched to ensure identical timing for each output path.

Moreover, by staggering the turn-on time of the three legs of the output driver, shown in Figure 10.16, peak switching currents are reduced, resulting in less ringing and lower SSN. Gate delay or RC delay can be used to create a small delta τ, the value of which depends on the targeted rise/fall times and data rate. The range of τ can be easily tailored from 30ps to 100ps.

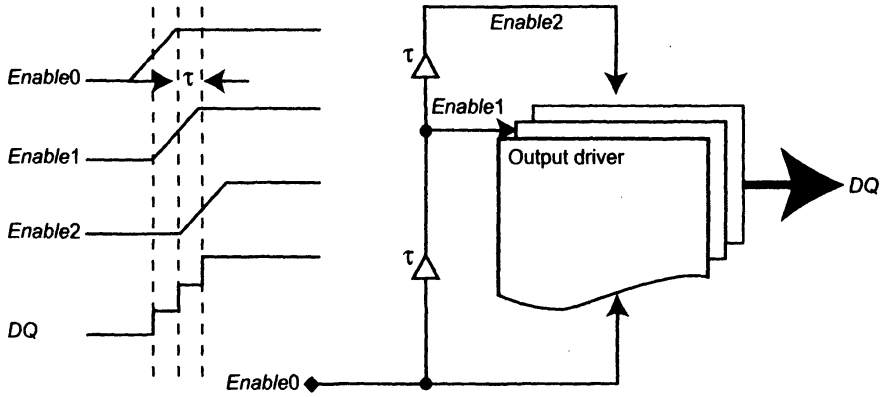


Figure 10.16 Slew-rate controls with staggered output enabling.

Minimizing *I/O* pin capacitance (C_{IO}) is another way to improve speed and reduce both active current $(C_{IO} \cdot dv/dt)$ and SSN. Lower C_{IO} can be achieved by using advanced package technologies. All DDR2 SDRAM require FBGA packages versus TSOP used for SDR/DDR SDRAM. Diagrams for these two packages are shown in Figure 10.17. Table 10.1 lists some example electrical parameters for TSOP and FBGA packages as used in commodity DDR parts. AC performance can be improved even further by using flip-chip and build-up package technologies.

Table 10.1 Pin RLC results for TSOP and FBGA packages.

	FBGA	TSOP	% Change
L (nH)	1.4	4.1	–65%
C (pF)	0.4	1.0	–60%
R (mΩ)	20	200	–90%



Figure 10.17 Lead-frame-based (TSOP) packages on the left versus fine ball-grid array-based (FBGA) board-on-chip (BOC) packages on the right.

From a circuit point of view, different termination methodologies, like center-referenced (V_{TT}) or V_{DDQ}-referenced termination, impacts driver linearity and pin capacitance differently. Generally a V_{TT}-terminated push-pull driver is capable of producing smaller pin cap than an open-drain driver for V_{DDQ} termination [3]. Consolidating an output driver with the ODT circuits can be a good way to reduce pin capacitance, which occurs naturally with the previously discussed impedance control scheme. To avoid excessive current, timing control for ODT is essential for V_{TT} termination. You can also reduce parasitics by placing the receiver and driver very close to the pad.

Last but not least, data-bus inversion (DBI) can be used to reduce SSN. By analyzing the data-stream pattern, *I/O* data can be manipulated internally to improve signal integrity. A DBI signal can be sent out via an inactive data mask or dedicated pin to indicate whether or not the data state is inverted. Three DBI algorithms are investigated for a high-speed graphics memory system:

1. For an NMOS open-drain configuration, the driver is V_{DDQ}-terminated. We can reduce the maximum power by minimizing zeros since zeros dissipate more power than ones. This algorithm is realized by simply counting zeros. For a group of eight DQs, if more than four zeros are detected, the DBI bit is set and the data is inverted.
2. In a minimum transition algorithm we reduce SSN by reducing the number of signals transitioning. A side benefit of this algorithm is reduced cross talk (both capacitive and inductive) between parallel signal paths. The transition detection is made by comparing current data to the previous data through XOR operations. If more than four transitions in a byte group are detected, the DBI bit is set and the data is inverted.
3. In a balanced output algorithm we minimize the difference between the number of ones and zeros that are transmitted simultaneously. This technique measures the balance of zeros versus ones across the transmitted data. If it appears unbalanced by a specific amount, we flip half of the data prior to transmission. The effect is to make the average current drawn from the supplies more consistent from cycle-

to-cycle and hence reduce irregularities in the transmission (timing and voltage variations).

A simulation is set up with a five-inch MicroStrip channel. Open-drain drivers with slew-rate control are modeled in Figure 10.18. The staggered output enabling (V_{ind1}, V_{ind2}, and V_{ind3}) is controlled by a resistor string (RC delay). The eye diagram results for all three DBI algorithms are summarized in Table 10.2. With the minimum added complexity, data bus inversion can readily reduce SSO and improve signal quality.

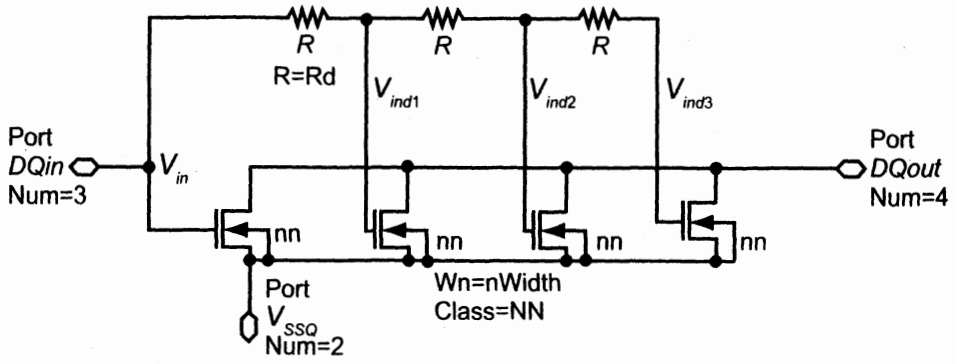


Figure 10.18 Output driver model used for DBI simulation.

Table 10.2 Average data eye parameters of four lanes with or without DBI.

	No DBI		DBI		Improvement	
Algo-rithm	Avg height (mv)	Avg jitter PP (ps)	Avg height (mv)	Avg jitter PP (ps)	Avg height (%)	Avg jitter PP (%)
1	237	137.8	270	101	15.24	–26.3
2	237	137.8	260	94.4	10.8	–31
3	237	137.8	273	110.3	16.1	–18.9

10.4 SIGNAL RETURN PATH SHIFT (SRPS)

Besides the signal path, there is always a signal return path, which is generally through a reference power plane, either V_{DDQ} or V_{SSQ}. When a signal is pulled up or down by the drivers at the near end or far end, its reference plane may change during transmission and cause a return path shift (SRPS). The impact and potential solution for SRPS are discussed in this section.

A simplified GDDR3 *I/O* system with a driver, ODT models (at the receiver side), transmission line, and V_{DDQ}/V_{SSQ} power planes is shown in Figure 10.19. Lumped inductances for V_{DDQ}, V_{SSQ}, and *I/O* pins are added, respectively, along with two decoupling capacitors: one (C_{plane}) between power planes at both ends and the other (C_{fat}) placed close to the *I/O*. These decoupling capacitors play an important role in reducing ground bounce and supply noise caused by SSO.

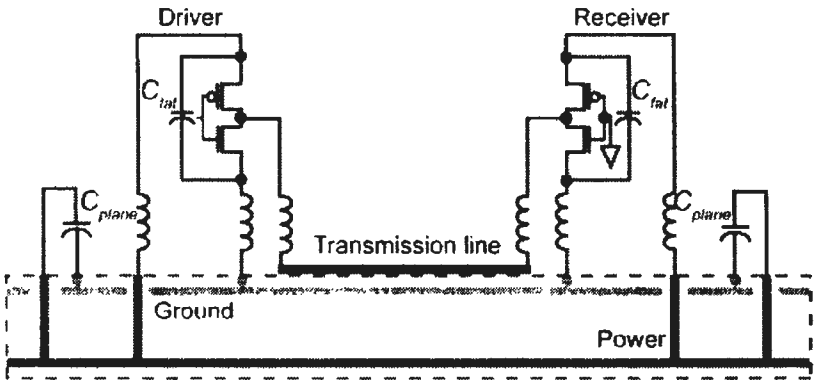
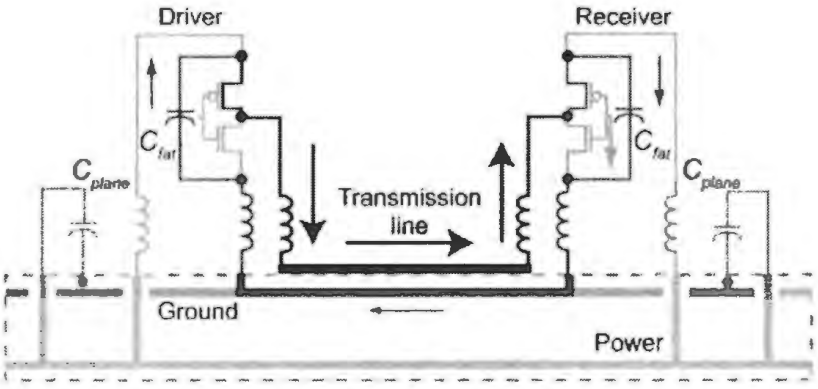
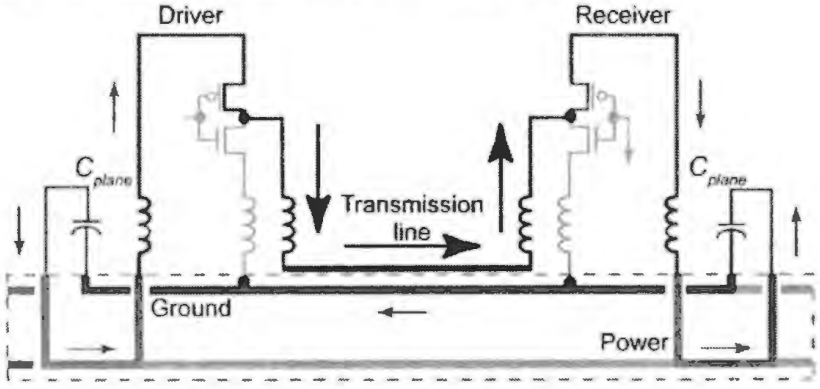


Figure 10.19 Diagram for a simplified GDDR3 *I/O* system with signals referenced to ground.

An example of SRPS with or without C_{fat} is shown in Figure 10.20a and Figure 10.20b, respectively. The signal is referenced to the ground plane and driven HIGH in this case. Cases for V_{DDQ} referenced and signal driven LOW can be analyzed accordingly. Notice that the decoupling cap (C_{plane}) offers little help for SRPS, as indicated by Figure 10.20b.



a) Signal return path with C_{fat}, no shift (V_{SSQ} return).



b) Without C_{fat}, signal return path is shifted between V_{DDQ} and V_{SSQ}.

Figure 10.20 Signal return path shift (SRPS) a) with or b) without C_{fat}, given output driving HIGH.

An experimental system is set up with eight *DQ* lines (T1 to T8) spaced 12 mils apart, with Z_0 = 60 Ω. Termination uses a 60 Ω fixed resistor. A four-layer, stack-up package is used with a signal-to-plane substrate thickness of 4.5 mils and a plane-to-plane thickness of 54 mil. Inductance per power or ground pin (L_P) is 0.5nH and 2.5nH per DQ pin. Data is running at 1500Mbps with 2^5 PRBS pattern. With one active aggressor (T1) and seven quiet victims, the cross talk is plotted in Figure 10.21, either with or without C_{plane}. Eye diagrams for one victim (T5) and seven aggressors are shown in Figure 10.22.

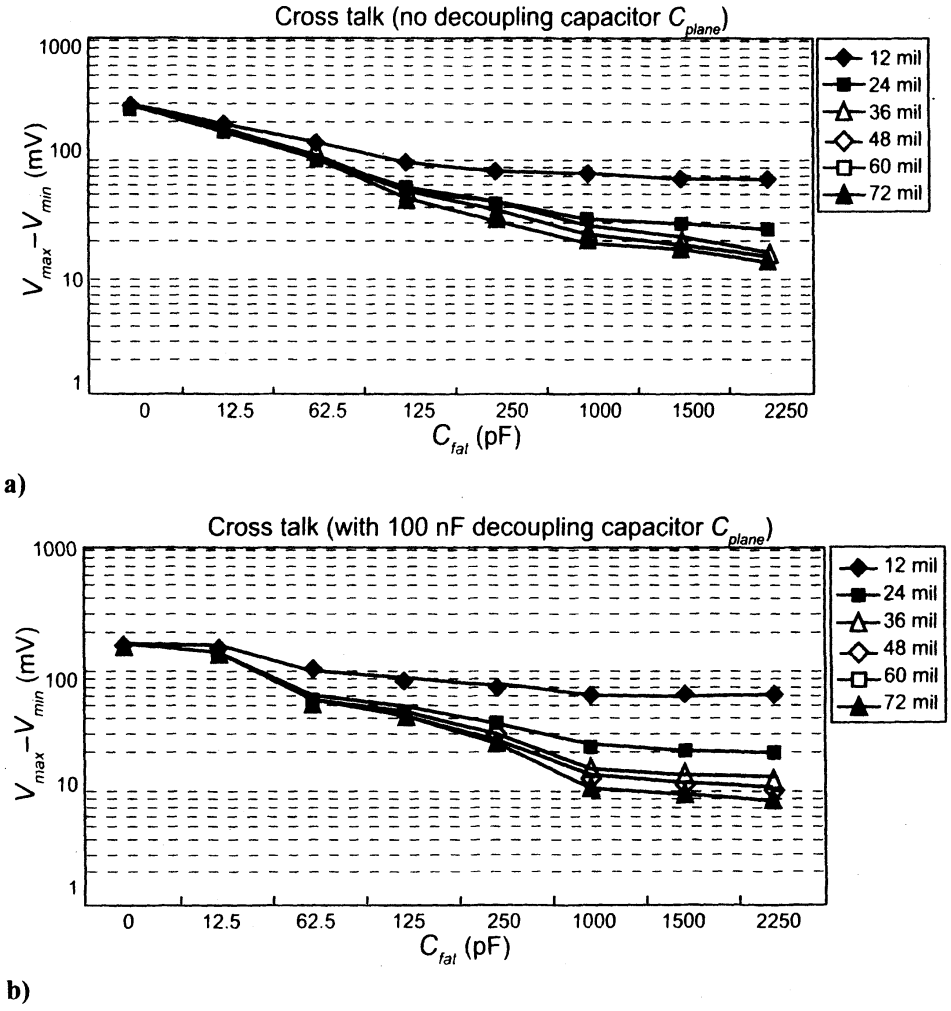


Figure 10.21 Measured cross talks with one aggressor:
a) no decoupling capacitor and b) 100 nF decoupling capacitor.

Two observations from Figure 10.21 follow:

1. When the victim trace is close to the aggressor, cross talk is the combination of SRPS (Signal Return Path Shift) noise and coupling noise.
2. When the victim is far away from the aggressor (>60 mil), cross talk is due only to the SRPS noise.

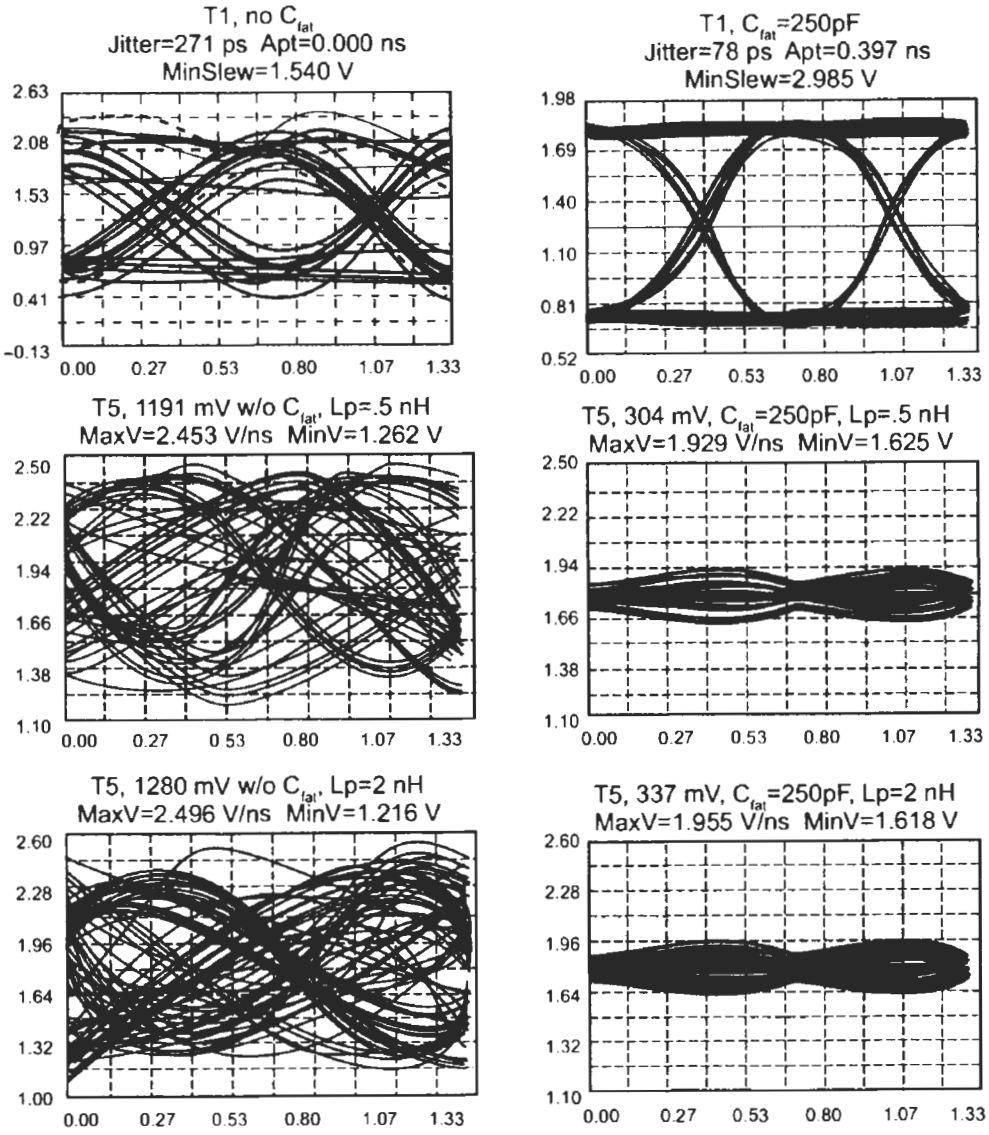


Figure 10.22 Eye diagrams for the experiment system, no C_{plane} (7 aggressors and 1 victim, T5).

The main SRPS issues are summarized as follows:

- Signal integrity of the main path degrades.
- Cross talk between lines increases, even with widely separated lines.
- Decoupling capacitors not located in close proximity to the SRPS points have little effect on reducing cross talk.
- With or without C_{plane}, the effects of SRPS can be significantly reduced by connecting the power ground planes with capacitors (C_{fat}) close to the SRPS points.
- The more SSO, the more cross talk.

- Noise can be reduced by lowering the inductance per pin.
- The amount of C_{fat} required for reducing noise varies for different systems and data rates. Noise can be managed with 250pF C_{fat} per driver in the exemplary GDDR3 system.

10.5 ELECTROSTATIC DISCHARGE (ESD)

Reliability issues such as ESD impose certain constraints in *I/O* circuit design [5]. ESD is a high-current, short duration stress. Three different models can be used to characterize ESD stresses:

1. Human body model (HBM)
2. Machine model (MM)
3. Charge device model (CDM)

The damages caused by ESD events include oxide breakdown for CMOS input gates, drain-substrate junction breakdown and contact punch-through for CMOS output buffers, and post-ESD leakage current increase between internal power pins. When designing an ESD circuit, the following rules should be kept in mind.

- Clamp the ESD voltage to shunt the ESD stress current.
- Turn on fast (less than 1 ns).
- Carry large currents > 2 amps or more for 150 ns.
- Occupy the minimum area at the bond pad.
- Introduce the minimum capacitance and series resistance.
- Retain immunity to process drifts.
- Offer protection for various ESD stress models.
- Do not interfere with the function, testing, or operation of the IC.
- Survive and remain inactive during burn-in tests.

Different ESD devices and circuits provide different protection. Carefully choosing ESD devices and circuits is very important to prevent any damage to the *I/O* circuits and, at the same time, achieve better performance. ESD devices can be grouped into two categories: breakdown devices, such as SCR (silicon-controlled rectifier) or GCNMOS (gate-coupled NMOS), and non-breakdown devices, such as diodes and MOSFETs. In general, the breakdown devices are very area efficient but difficult to design due to their more complicated behavior.

An ESD *I/O* protection circuit diagram is drawn in Figure 10.23. An input resistor (R_{input}), pass gate and CDM voltage clamps are applied to

protect input gate oxide. The GCNMOS device is for the power clamp. ESD diodes (ESDPN and ESDNP) serve as primary current clamps for the *I/O* supplies. A large decoupling capacitor (FatCap) between power and ground can shunt some ESD stress current and improve ESD performance.

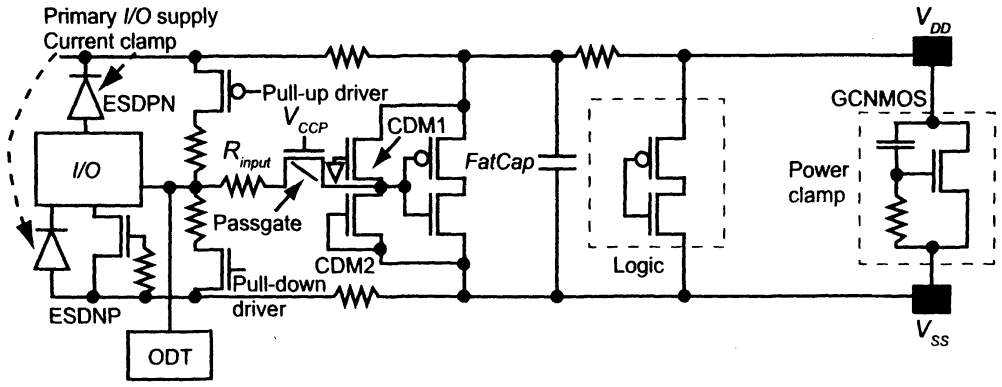


Figure 10.23 Diagram for *I/O* ESD protection circuits.

The output driver transistor can be built to act like a stand-alone ESD circuit or as a high-speed driver. If we lay out the driver to meet the high current requirements that ESD requires (i.e., 3 amps), then this would impact the resolution of impedance matching and high-speed performance of the driver. Instead, ESD and the driver can be separated into two different circuits. The tuning transistors for impedance and ODT control would be chosen to meet the desired LSB without any ESD considerations. The ESD circuit would be then added in parallel as the best possible ESD solution. This attempt can maximize the performance of both circuits independently. However, this approach works up to a certain bandwidth like 3 GHz. Beyond that, the drivers are built as both driver and ESD circuit in an attempt to allocate design space for the ESD circuit function.

As mentioned before, C_{IO} reduction is beneficial for high-speed operations. However, bidirectional *I/O* with push-pull drivers and ODT circuits add more capacitance per *I/O* pad. From the DDR2 C_{IO} spec, only 2.5–3pF is allowed, which leaves little room for the ESD circuits. On the other hand, the diodes for primary power and ground current clamps can add up to more than 0.3pF. If the ESD circuits are too small, then there will be a yield penalty as Assembly and Test still generate static electricity through their back-end process and handling. ESD circuits using a breakdown device like an SCR can reduce capacitance with a turn-on time limitation. A modified SCR for twin-well technology is reported in [3] with a maximum 0.45pF C_{IO} reduction. A diode-triggered SCR (DTSCR) [6] can achieve 500ps turn-on time with only 0.2pF pin capacitance.

Noise coupling from the ESD diodes among different power domains must be addressed. When the power bus networks are broken into sections,

i.e., V_{DD}, V_{DDA}, and V_{DDQ} as in Figure 10.12, the ESD discharge current event can cause voltage spikes on one power bus network and not the other. Those circuits that receive the signal from one domain to another are at severe risk of failing. This issue is already present with 32 Å gate oxides at the 90nm DRAM process. This problem will become worse for more advanced processes. The receiver gate oxides are at risk of seeing the high stress voltages, and they will either trap charge on their gates or be totally damaged, changing their switching speeds. One way to minimize this risk is to minimize the number of circuits at these power domain interfaces. The other option is to reduce the fan-out after a signal is sent from one power domain to another. For noncritical speed paths, the receiver circuit can be designed so that its gate oxide is thick and not thin. For critical speed paths, the circuits can be put in one specific area with very small ESD power clamps between V_{DD} and V_{DDQ}, for instance. One example would be in the pre-driver of the output path. These small ESD devices limit the voltage transients without affecting the high-speed signals.

Layout is another critical issue when designing high-speed *I/O* circuits. It has been found that the metal routing from the bond pad to the *I/O* drivers, ODT, and ESD circuits can add a large amount of parasitic fringe metal capacitance. Routing one metal into a circuit and then using a different level of metal to route out of the circuit helps decrease parasitic metal capacitance. By laying out the ESD circuits on the side of the bond pad, the routing to them from the pad has the smallest distance to travel.

Moreover, ESD circuits are designed to carry high current; consequently, metal interconnect resistance is a critical parameter. As the technologies have scaled, the contact aspect ratio has become significant. When the contact is formed, the stack height reduces the contact area. Adding more contacts becomes inevitable to drive extremely high current through these small contact areas. Unfortunately, adding more contacts widens the metal and increases the fringe capacitance.

10.6 PARALLEL-TO-SERIAL CONVERSION

After discussing the signal integrity issues and their influence on output driver design, we now investigate parallel-to-serial conversion.

As mentioned in Chapter 9, the memory internal data bus is fairly wide in order to pre-fetch multiple bits at the same time, enhance parallelism, and relieve the burden of array cycle time. Read data is typically loaded into pipelines prior to stream out. This internal data must be serialized and ultimately converted back to the external data rate. An output path is shown in Figure 10.24, with a parallel-to-serial (P2S) converter as the starting point. P2S generally employs a high-speed multiplexer controlled by the clock

edges. The outputs of P2S are fed into pre-drivers and finally output drivers with proper slew-rate and impedance controls.

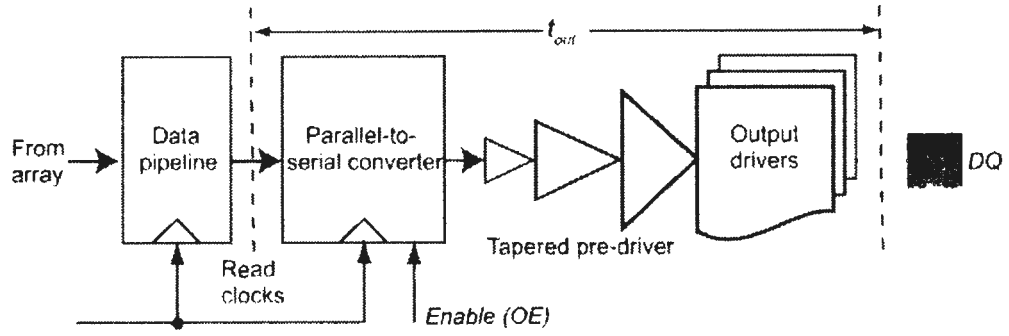


Figure 10.24 Block diagram of typical output path.

The overall output timing is denoted as t_{out}, which is from the internal Read clock (i.e., the DLL and clock distribution network) to the *DQ* at the pad. The output path is generally modeled in the timing control circuit (i.e, DLL) and latency control logic to maintain clock synchronization and the desired programmable *CAS* latency. Several timing parameters can be derived from or related to the output timing:

- t_{AC} as DQ output access time from external system clocks *CLK/CLK**
- t_{DQSCK} as Read strobe (*DQS*) access time from *CLK/CLK**
- t_{DQSQ} as DQS–DQ skew (*DQ* timing relative to *DQS*)

The first two parameters are linked to the performance of the on-die timing control circuits, like DLL. Due to power and area constraints, the model used in the DLL is not a straight copy of the output path. Instead, a scaled down version of the output path is used, especially for pre-drivers and output drivers. Moreover, the noisy *DQ* supplies (V_{DDQ}/V_{SSQ}) cannot be mapped into the output model, which could result in excessive timing jitter. On the other hand, *DQS–DQ* skew is a measure of the matched routing of output paths and SSO-induced jitter. For high-speed DRAM interface, t_{DQSQ} is more stringent than t_{AC} and t_{DQSCK} since strobes are used to capture the data at the memory controller. *DQS* is generally treated as a special data signal having a one-zero toggle pattern. For DDR2-800, the maximum t_{DQSQ} is only 175ps, which is half of the t_{DQSCK} spec. For GDDR4, t_{DQSQ} is reduced to only 80ps.

A circuit block diagram for a P2S is shown in Figure 10.25. Four identical latches can be used with outputs wire OR'd together for multiplexing four bits of data. Multiplexer fan-in can seriously limit the bandwidth of the output path [7]. In Figure 10.26, a dynamic latch with domino logic can be used for the Read latch. The dynamic latch has speed advantages over the static latch because the load for input (*D*) is only one NMOS transistor

(WN39). The internal sample clock *CLKsp* is single phase with light loading. A keeper circuit bleeds a small current to counter the leakage through an internal high-impedance node, *d**. Sizing the output stage of this latch and buffer stage in the P2S is critical to maintaining good duty cycle.

Data is first sampled by the rising edges of multi-phase internal Read clocks, which run at lower speed for less distortion and better clock distribution. A clock sampling window for bit 0 is shown in Figure 10.27, where C0 is at logic "1" and C90 at logic "0." C0 and C90 are connected to *CLK1* and *CLK2* signals of the Read latch, respectively. Clock routes and loads are matched inside the P2S to maintain the phase relationship. Implementation of the phase generator is discussed in Chapter 11.

An output enable (*OE*) is generated from the latency control logic to start the serialization. The *OE* signal passes through the same latch and selects either the latch output, ODT, or tri-state signal. Then proper pull-up or pull-down signals are generated and fed into the pre-divers and output drivers.

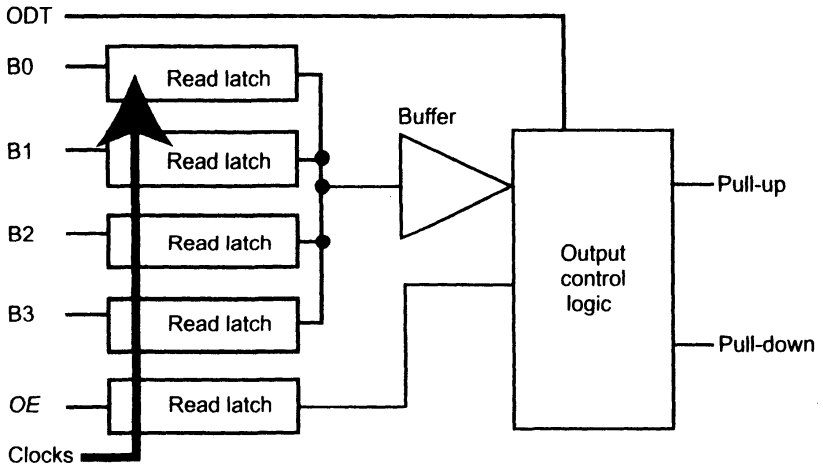


Figure 10.25 Circuit diagram of a parallel-to-serial converter.

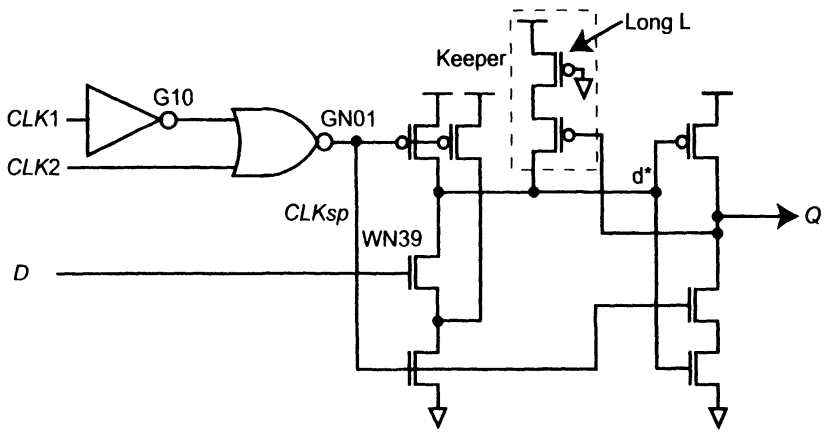


Figure 10.26 Dynamic Read latch for high-speed P2S.

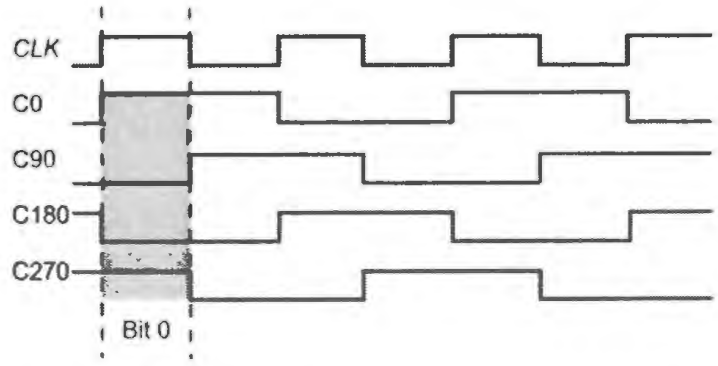


Figure 10.27 Four-phase clocking and sampling window for P2S.

Some speed limitation is imposed by the P2S because the data is serialized and running full-speed from this point. Due to the bandwidth limitation of the CMOS gates, it is difficult to push the bit time below 400ps (2.5Gbps data rate) without encountering some signal distortion. One way to handle this is to push the serialization out to the output drivers [8]. This output-multiplexed architecture requires multiple copies of the output driver, which is power- and area-hungry. Current-mode logic (CML) is another solution for the output path when the speed is beyond 2.5Gbps. With reduced voltage swing and fully differential signaling, CML offers superior performance over traditional CMOS logic with the added cost of constant current draw and a more complicated design. A CML-based output path [7] is shown in Figure 10.28. Pass transistors controlled by different phases of the clock are used for input multiplexing. True and complementary signals are required for CML, which effectively doubles the logic.

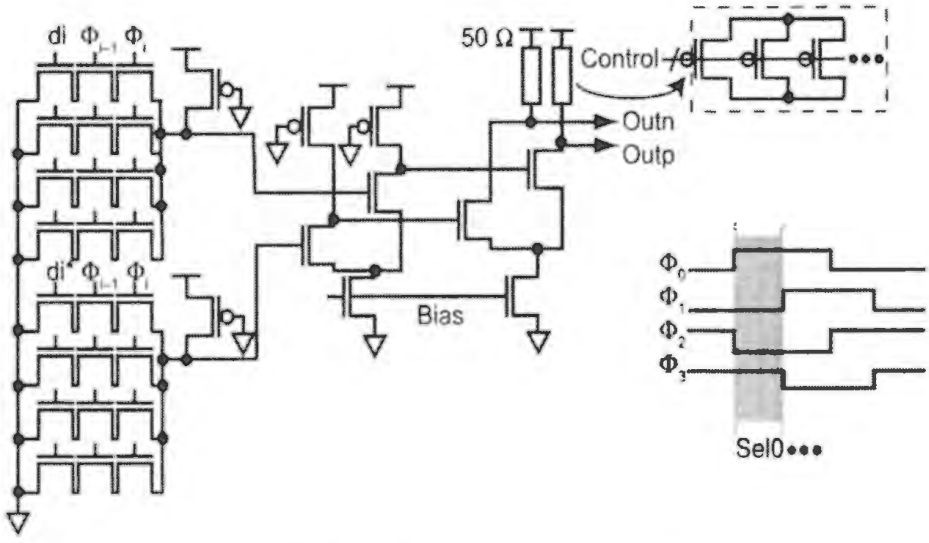


Figure 10.28 CML-based output paths [7].

10.7 EMERGING MEMORY *I/O* FEATURES

As *I/O* speeds continue to increase, circuit techniques generally found in communication systems will start penetrating into the memory interface domain. We discuss a few of these techniques in this section.

1. Differential signaling. Single-ended (SE) signaling is the standard for data in current DRAM interfaces due to pin restrictions and bandwidth requirements. SE signaling is susceptible to SSN and common-mode noise. Differential signaling, on the other hand, can reject common-mode noise, reduce ground bounce and SSN, minimize EMI, and double the slew rate. To justify the increasing pin count, however, data must run twice as fast. High-speed differential logic, like CML is well suited for differential signaling.
2. Unidirectional *I/O*. Bidirectional *I/O* generally becomes a burden for high-speed data operations. To avoid bus contention, timing restrictions are needed, which hurts Read and Write turnaround. Pin capacitance is another hurdle for speed increases. Separate inputs and outputs are generally beneficial except for the added cost of higher pin counts. Simultaneous bidirectional (SBD) signaling [9–10] is appealing if the bandwidth and pin counts are truly a concern. With limited swing and multiple reference points, SBD signaling would be a challenge for both Design and Test.
3. Transceiver equalization and pre-emphasis. From transmission line (TL) theory described in Section 10.1, attenuation of the TL increases with frequency due to the skin-effect resistance defined in Equation (10.3). The digital signal used in memory systems includes both low-frequency and high-frequency components. The superposition of non-attenuated, low-frequency signal components with attenuated, high-frequency signal components causes inter-symbol interference (ISI). This interference degrades noise margins and reduces the maximum frequency at which the system can operate. Equalization for both the transmitter and receiver can be incorporated to boost the high-frequency component and eliminate the problem of frequency-dependent attenuation. The desired frequency response should be flat over the entire operating range. Compared to receiver equalization, transmitter pre-emphasis is much easier to implement with a multi-tap FIR (finite impulse response) filter [7–8]. However, adding an equalization path increases the power and area of the output drivers.
4. Data encoding has long been used in communication systems for bit error correction. To achieve better bit-error-rate (BER), error-correcting code (ECC) or cyclic-redundant code (CRC) can be applied to a

memory system. However, data encoding would make memory design much more complicated.

Obviously, any of these circuit techniques to improve *I/O* performance are not free, especially considering added cost for test and validation. Nevertheless, the path toward higher performance always necessitates greater complexity as evidenced by the evolution of circuits used in SDR, DDR, and DDRII devices. For a specific memory technology, good engineering judgment is needed to balance performance needs against complexity and cost.

REFERENCES

[1] H. Johnson and M. Graham, *High-Speed Signal Propagation, Advanced Black Magic*. Prentice Hall, Upper Saddle River, NJ, 2002.

[2] H. Y. Song, S. J. Jang, J. S. Kwak, C. S. Kim, C. M. Kang, D. H. Jeong, Y. S. Park, M. S. Park, K. S. Byun, W. J. Lee, Y. C. Cho, W. H. Shin, Y. U. Jang, S. W. Hwang, Y. H. Jun, and S. I. Cho, "A 1.2Gb/s/pin double data rate SDRAM with on-die-termination," *ISSCC Digest of Technical Papers,* pp. 314–315, 2003.

[3] C. Park, H. Chung, Y. Lee, J. Kim, J. Lee, M. Chae, D. Jung, S. Choi, S. Seo, T. Park, J. Shin, J. Cho, S. Lee, K. Song, K. Kim, J. Lee, C. Kim, and S. Cho, "A 512-MB DDR3 SDRAM prototype with C_{io} minimization and self-calibration techniques," *IEEE Journal of Solid-States Circuits*, vol. 41, April 2006.

[4] Y. Fan and J. E. Smith, "On-die termination resistors with analog impedance control for standard CMOS technology," *IEEE Journal of Solid-State Circuits*, vol. 38, Feb. 2003.

[5] S. Dabral and T. J. Maloney, *Basic ESD and I/O Design*. Wiley & Sons, Inc., New York, 1998.

[6] J. D. Sarro, K. Chatty, R. Gauthier, and E. Rosenbaum, "Study of design factors affecting turn-on time of silicon-controlled rectifiers (SCRS) in 90 and 65mn bulk CMOS technologies," in *IEEE 44th Annual International Reliability Physics Symposium*, 2006, pp. 163–168.

[7] M. E. Lee, W. J. Dally, and P. Chiang, "Low-power area-efficient high-speed *I/O* circuit techniques," *IEEE Journal of Solid-State Circuits*, vol. 35, November 2000.

[8] W. J. Dally and J. Poulton, "Transmitter equalization for 4Gb/s signaling," *Proc. Hot Interconnect*, pp. 29–39, Aug. 1996.

[9] E. Yeung and M. Horowitz, "A 2.4 Gb/s/pin simultaneous bidirectional parallel link with per-pin skew compensation," *IEEE Journal of Solid-State Circuits*, vol. 35, pp. 1619–1628, Nov. 2000.

[10] J. Kim, S. Kim, W. Kim, J. Choi, H. Hwang, C. Kim, and S. Kim, "A 4-Gb/s/pin low-power memory *I/O* interface using 4-level simultaneous bi-

directional signaling," *IEEE Journal of Solid-State Circuits*, vol. 40, pp. 89–101, Jan. 2005.

Chapter
11

Timing Circuits

As mentioned in Chapter 5, timing circuits, such as those used for clock synchronization, are an important class of circuits. In this chapter, we explore various types of timing circuits and their implementation. Among these, delay-locked loop circuits (DLL), both digital and analog, are covered in detail due to their simplicity and wide use in high-speed DRAM.

11.1 INTRODUCTION

Timing circuits, in general, are associated with a clock. A system clock, which is generated by the memory controller, synchronizes the input and output (*I/O*) operations of the DRAM. It makes sense then that two of the biggest obstacles for high-speed *I/O* operations are clock-related: clock skew and jitter. Clock skew is any timing difference between the external system clock and the internal DRAM operating clocks. A propagation delay from the *I/O* circuits and local buffers (to drive long interconnects for clock distribution) can skew the internal clock relative to the external clock. Delay mismatch among different clock paths and duty-cycle distortion can impact the timing budget even further. Dynamic clock jitter, on the other hand, can be introduced by supply noise; cross talk; and voltage and temperature (VT) variations. Timing variations (both skew and jitter) along the critical data path could cross several clock cycles as speeds increase making it difficult to predict and track latency. To reduce such variations, timing circuits such as clock synchronization or de-skewing circuitry are generally used. These play an increasingly important role in high-speed DRAM circuit design.

To achieve better performance, the critical timing path of the memory device from input to output must be carefully analyzed and modeled. The critical timing path includes the input path (Chapter 9), output path

(Chapter 10), clock distribution networks, and one or more de-skewing circuits. Latency control (Chapter 12) and power delivery (Chapter 13) must also be considered when designing the timing circuits. Although high-speed circuit design for de-skewing is the focus of this chapter, each piece along the critical timing path inevitably interacts with the core sync circuits and contributes to the overall timing budget. Several critical timing specs, such as t_{AC}, t_{DQSQ}, t_{DQSS}, *CAS* latency, data setup and hold, etc., can be determined through the analysis. Therefore, the design of clock synchronization circuits are integral to the overall system; design modifications and improvements are necessary to accommodate increasing clock speed.

So, how is the design of synchronization circuits impacted by increasing clock frequencies? First, the timing margins shrink with a shorter clock cycle time. In turn, the clock and data alignment must be more precise and tolerate less jitter. Because both the rising and falling edges are utilized, duty-cycle distortion jeopardizes the accuracy of data processing. Second, clock domain crossing becomes more of a problem due to timing uncertainty arising from clock skew and short cycle time. With the inherently slow transistors and minimum number of metal layers of a DRAM process, clock and power delivery is a challenge: ground bounce, voltage bumps, and noise from cross coupling degrade timing. Last but not least, impedance mismatch and inter-symbol interference (ISI) degrade signaling, which necessitates on-die calibration and training. Advanced packaging technology must be considered and modeled in the design flow to analyze timing and power delivery. Since cost remains the primary concern for DRAM manufacturers, design trade-offs must be made between cost and performance, simplicity, and scalability.

Historically, sync circuitry first appeared in DDR SDRAM, in which the clock frequency was approximately 66–133 MHz. A simple digital delay-locked loop was used to synchronize the clock with output data. High-performance GDDR4 DRAMs have pushed speeds up to multi-gigahertz range, while scaling cycle times down to less than 1 ns. Achieving such high speeds requires enormous complexity in the sync circuit and system design. However, variability in the lock time, tuning range, resolution, jitter, power, and layout can greatly impact overall system performance.

Based on various design choices, sync circuitry can be implemented using digital or analog, open or closed loop methodologies or any combination thereof. The two most common sync circuits in use today are the delay-locked loop (DLL) and the phase-locked loop (PLL). The DLL is widely used in memory interfaces for its simplicity and good jitter performance. An alternative approach is the PLL, which is usually found in communication systems and microprocessors for clock data recovery and frequency synthesis. Both the DLL and PLL are closed-loop systems with a reference and

feedback signal. The biggest difference between a DLL and PLL is that a DLL merely passes the reference signal through a variable delay line, while a PLL generates a new signal that is both phase- and frequency-locked to the reference signal.

Although both the DLL and PLL are good for clock de-skewing, the PLL is rarely used in high-speed DRAM due to the following:

1. Increased complexity
2. Stability issues inherent in a higher order system
3. Need for frequency locking
4. Longer lock time

Unlike the PLL, jitter in a DLL is small and easy to control due to the lack of jitter circulation, which occurs in a PLL oscillator. However, any input jitter may directly pass to the DLL output without any filtering. With a low-jitter reference signal, a well-designed DLL can achieve the same jitter performance as a PLL for clock synchronization.

An analog approach generally includes a voltage-control delay line (*VCDL*) and a charge-pump phase detector to generate a control signal [3]. However, an all-digital DLL (ADDLL) can be implemented for greater scalability and portability across DRAM process generations. Because logic gate delay remains fairly constant as feature size and supply voltage scale, an all-digital design can be reused with little modification and is also easy for test and debug, an issue which cannot be over-emphasized for modern DRAM design. Good stability over process, voltage, and temperature (PVT) variations is another plus because it is a 0th-order system without integration. A register-controlled DLL [1, 2] is discussed in Chapter 5, and ADDLL design is addressed in detail in Sections 11.2 of this chapter.

A higher performance DRAM generally supports a variety of power-saving modes. To resume normal operation after exiting power-saving modes, the sync circuit must be relocked, resulting in some start-up delay. Lock time of a sync circuit becomes an important design issue, and digital implementations usually surpass analog counterparts in this area. Some open-loop architectures such as synchronous mirror delay (SMD) [5] and ring counter delay (RCD) can achieve lock within several cycles although they lack fine timing resolution. A combination of DLL and SMD, called measure-controlled delay (MCD), has been proposed to take advantage of quick initialization with a closed-loop structure for better voltage and temperature tracking after initial lock. Voltage and temperature variations introduced by the power saving in DRAM can greatly impact critical timing needed for high-speed operation. Power-saving modes may need to be traded for better overall timing performance.

As mentioned, clock and power delivery are big challenges for high-speed DRAM circuit design. To maintain proper slew rate and duty cycle, the external clock must be shaped and re-driven over a long distance internally. Clock power for the distribution network can represent as much as 70% of the entire timing path and even higher as the speed increases. One solution to this problem is scaling down the internal clock frequency by using a clock divider. Except for the *I/O* circuits, a single-phase, low-speed clock signal can be distributed globally across the die with less attenuation and power. However, the high-speed interface still processes data in full data rate (twice the clock frequency), and an edge recovery circuit must be added to transform the internally divided-down clock into multi-phase clocks that include all missing edge information. A mixed-mode DLL with an analog phase generator can be configured to suit this purpose. It is obvious that increased speeds demand more complex clocking system design.

Layout is another challenge for high-speed clock synchronization design. Full custom layout is generally required for better matching, routing, sizing, and shielding. The delay line (for delay adjustment) is without question the most important piece as it carries a clock signal. For mixed-mode layout, partitioning sensitive analog blocks from digital circuits with proper isolation, bussing, and placement is key for better performance.

A circuit design is not complete without built-in test capability. Design-for-test (DFT) is essential for clocking in high-speed DRAM products. Wafer-level probing is a challenge due to power delivery, high-speed clock and signal delivery, and availability of limited Probe resources. Trouble-shooting can be extremely time-consuming if not impossible. Along with circuit design, we address various test issues related to timing circuits.

11.2 ALL-DIGITAL CLOCK SYNCHRONIZATION DESIGN

Designing clock synchronization circuits starts with basic timing analysis, as shown in Figure 11.1, which depicts the *critical timing path*. The typical timing path includes an input path (t_{in}), a DLL with an adjustable delay line, a phase detector (PD) and feedback model (t_{model}), clock distribution (t_{tree}), and a data output path (t_{out}). An all-digital DLL (ADDLL) is used as an example to generate an internal clock for latching the data (from the array) and synchronizing outputs (the *DQ*s) with the external system clock (*XCLK*). The major building blocks of the ADDLL are discussed with respect to different design aspects, including layout and test issues. Design modifications and improvements for high-speed, all-digital implementation are the focus of this section.

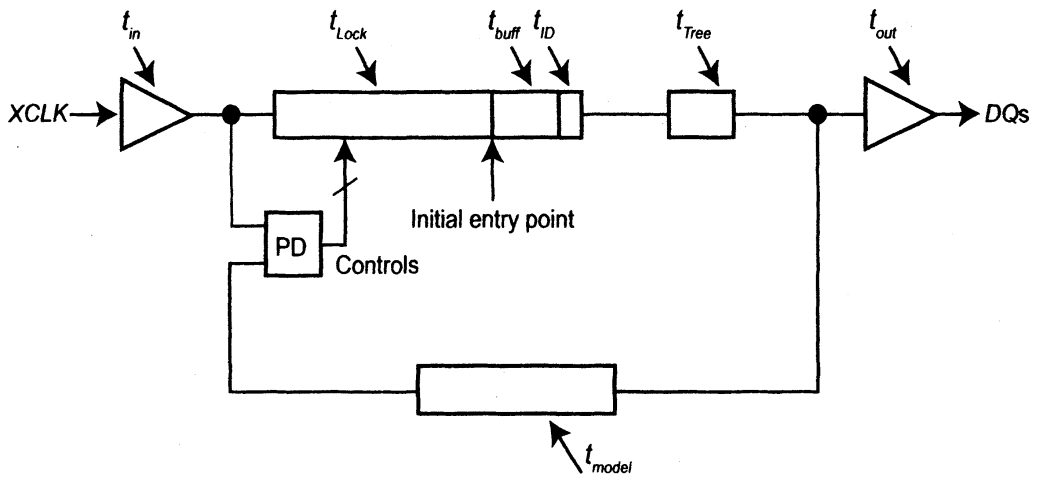


Figure 11.1 Timing parameters of a typical all-digital DLL used in a memory system.

11.2.1 Timing Analysis

Intrinsic delay (t_{INTRI}) is defined as the delay between the clock input (*XCLK*) and the data outputs (*DQ*s), excluding the actual insertion delay (t_{Lock}) provided by the DLL delay line. The intrinsic delay is also referred to as *intrinsic forward path delay.* From Figure 11.1, t_{INTRI} can be expressed as

$$t_{INTRI} = t_{in} + t_{buff} + t_{ID} + t_{tree} + t_{out} \tag{11.1}$$

where t_{in} is the delay of the clock receiver and input buffer, and t_{buff} is the delay reserved for tuning after initial lock due to voltage and temperature variations. Parameter t_{ID} includes DLL buffer delays and a fine adjustment delay, if any. The delay of the data output path t_{out} includes the *DQ* latches and *DQ* drivers. For low-speed operations, the cycle time (t_{CK}) is usually greater than t_{INTRI}. As speed increases, t_{CK} may be a fraction of t_{INTRI}, and tight synchronization is needed to reduce timing skew. In addition to the forward path, a DLL loop also includes a feedback path. An *I/O* model takes an input from the clock tree and generates an output signal fed back to the PD. The *I/O* model tries to track the delay of input and output paths and can be defined as

$$t_{model} \approx t_{in} + t_{out} \tag{11.2}$$

When the DLL is locked, the total forward path delay, t_{TOTAL}, is equal to m times the clock period (t_{CK}), where m is an integer number greater than zero. The relationship can be given as

$$t_{TOTAL} = m \cdot t_{CK} = t_{INTRI} + t_{Lock}, m>0 \tag{11.3}$$

where t_{Lock} is the insertion delay when the DLL is locked. Assuming a digital delay line with a total of N stages where the delay per stage is t_D (unit delay) and n is the number of delay stages involved in the delay line when the loop is locked, the tuning range can be given as

$$t_{Lock} = n \cdot t_D, 1 \leq n \leq N \tag{11.4}$$

Generally, it is important to know the timing difference caused by process, voltage, and temperature (PVT) variations. Table 11.1 and Table 11.2 present data based on an exemplary timing analysis. A digital delay line with fine and coarse tuning is shown as part of the DLL intrinsic delay in Table 11.1. Different implementations produce different DLL intrinsic delay. The clock input buffer also includes a clock divider for high-speed operation. The sum of the intrinsic forward path delay is shown as the total delay.

Table 11.1 Timing data under PVT variations.

PVT Corners	Input t_{in}	Output t_{out}	DLL Intrinsic	Clock Tree t_{tree}	Total Delay
Slow, 1.3 V, 85°C (ps)	1685	1615	3240	1967	8507
Fast, 1.9V, 0°C (ps)	721	780	1816	850	4167
Δ (ps)	964	835	1424	1117	4340
% of delay change	+57%	+51%	+44%	+56.8%	+51%
% of total delay	17~20%	19%	38–43%	20–23%	100%

The data in Table 11.1 show the sensitivity of major timing blocks versus PVT variations. The clock input path, clock distribution, and data output path represents 60% of the overall forward path delay at approximately 20% for each. Larger die size and additional data pins require a larger clock distribution network and produce longer delay. The ADDLL intrinsic delay accounts for the remaining 40% of the forward-path delay. Worst-case PVT variation could introduce 60% timing skew (five cycles of variation) for the total forward delay after lock for a 1GHz external clock.

Given a typical process, the voltage and temperature (VT) variations depicted in Table 11.2 produce up to 2.1 ns (40%) timing skew (1.6 ns is due to the 600 mV supply change). For every 100 mV drop in the nominal 1.5 V supply, the total forward path delay (excluding DLL delay) can change by 360 ps, almost an entire bit time at a 2.5GBit/s data rate. Compared to lowering the voltage, increasing the voltage has less impact on delay. On the other hand, a 100°C temperature change may vary the delay by half a nanosecond with higher voltages creating even greater temperature sensitivity.

Table 11.2 Timing data under voltage and temperature variations with a typical process.

Conditions	Input	Output	*CLK* Tree	Total Delay
1.5 V, 110°C (ps)	1298	1196	1472	3966
1.5 V, 0°C (ps)	1130	1057	1287	3474
1.2V, 110°C (ps)	1665	1519	1856	**5040**
1.2 V, 0°C (ps)	1499	1386	1669	4554
1.8 V, 110°C (ps)	1124	1028	1288	3440
1.8 V, 0°C (ps)	951	885	1087	**2923**
% of the total delay	33%	30%	37%	100%
Delay change (ps) / +100°C	157	130	183	470
Delay change (ps) / –0.1 V	123	109	128	359
Delay change (ps) / +0.1 V	–59	–57	–64	–180

Supply noise sensitivity is the number one cause of timing variations in high-speed, low-voltage DRAM. Slower transistors make it worse. All-digital implementation likely takes a much bigger hit due to the lack of supply noise rejection. Supply noise can be triggered by array activation, coupling, and IR drop. Delay variation due to supply noise can accumulate and translate to significant short-term timing jitter. A power delivery scheme is shown in Figure 11.2. Dedicated power buses with large decoupling capacitors can be applied to the critical timing blocks, e.g., input buffer, clock trees, etc. These "quiet" power buses are isolated from other global circuits. Dedicated power and GND pins are assigned to the DLL along with its own bussing. The DLL is placed close to the input path to produce shorter routes. Voltage regulation may be another way to reduce supply noise at the cost of lower voltage margin at high speed. If the clock is stopped for power saving, the voltage perturbation caused by supply current changes may result in longer settling and relock time.

I/O model mismatch is another factor for timing skew. Locality is important for better matching in that the input model should be placed in the proximity of real input buffers with the same buses. Not only logic style, but gate counts must be modeled, and metal routes must account for the feedback signal. However, noise injected from a noisy bus (like supplies for output data) can make the DLL loop unstable. A relative constant current profile for the entire timing path is less prone to supply noise. Layout with proper shielding, isolation, and placement for timing critical circuits should be the first priority. Substrate coupling is also a concern. It is desirable to properly strap the n-well and substrate. A circuit with better supply noise rejection, like fully differential current-mode logic (CML), can be very useful and is discussed in Section 11.3.1 of this chapter.

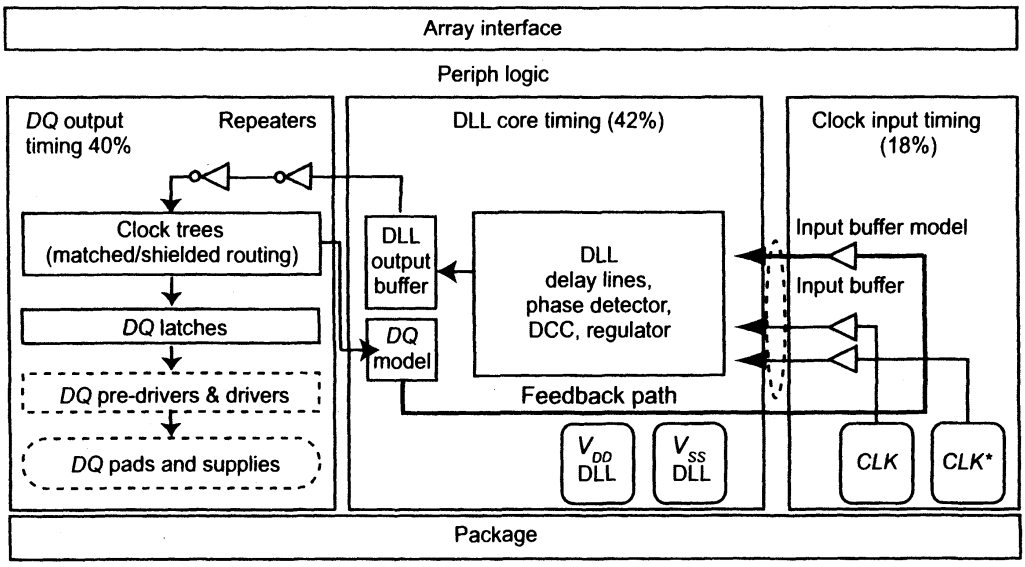


Figure 11.2 Power bussing scheme for a clock synchronization system.

11.2.2 Digital Delay Line

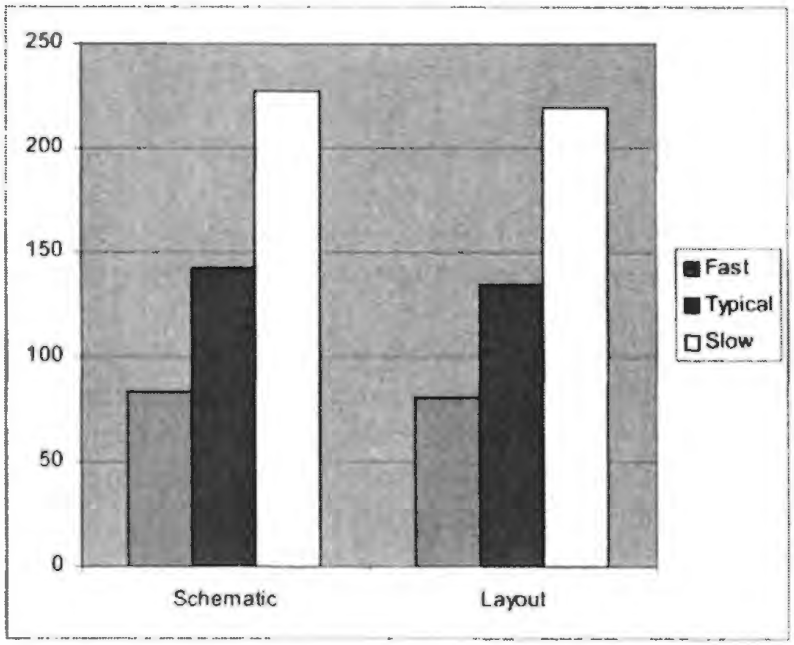
A digital delay line provides an adjustable delay for the ADDLL. The delay is adjusted discretely by adding or removing gates from the delay line. A delay line can be characterized with two parameters: unit delay (t_D) and a number of stages (N). The multiplication of the two parameters defines a lock range in which a DLL can operate under all PVT variations. The lock range for a synchronous memory system tends to be wide to cover various frequencies necessary to support speed grading. A speed grade represents a set of timing parameters tied to device operation at specific data bus frequencies. For high-speed DRAM, these timing parameters become more demanding and harder to meet. Fortunately, devices unable to meet all of the timing specs for a specific speed grade can be down-binned to a slower speed grade. Different speed grades put a burden on timing circuit design to cover a wider operating range for a given process.

11.2.2.1 Unit Delay (t_D) Unit delay of the digital delay line determines the maximum jitter and number of delay stages that must be included to ensure proper lock range. The value of this unit delay may vary –40/+60% with process, voltage, and temperature (PVT) variation. Typical unit delay in current DRAM process lies in the 100–250 ps range, with two NAND gates connected serially. The NAND gate, instead of an inverter, is chosen for its simpler control logic and reduced power consumption (due to unused delay stages). The delay resolution can be improved by increasing the drive of the basic delay element with the cost of more power dissipation and larger silicon area. More delay stages are needed to ensure a desirable lock range with a smaller unit delay. Table 11.3 shows the simulation results for the NAND-gate-based unit delay under different conditions (which may vary according to process and layout arrangements). A careful layout to put the critical delay elements close together can improve the unit delay by reducing interconnect parasitics. However, the best way to improve the resolution of the digital DLL is through delay interpolation, creating finer delays (as low as 10 ps).

Table 11.3 Unit delay (two NAND gates, in ps) under different conditions.

Simulation Corners*	Schematic (Fixed Load)	Layout (Extraction)
Fast: 1.9 V, FF, –40°C	83	81
Typical: 1.5 V, TT, 85°C	142	134
Slow: 1.2V, SS, 110°C	227	219

*(TT: typical NMOS and PMOS; FF, fast corner; SS, slow corner.)



Graphical representation of Table 11.3.

The capacitive load (C_d) of one delay element (half of the delay stage) is the sum of output capacitance, input capacitance of the next delay element, and interconnect capacitance. Because the CMOS NAND gate has four transistors versus two in the CMOS inverter, C_d is comparably larger in the NAND-based delay stages. The size of the delay element also determines the value of C_d, which should be a compromise between speed, power dissipation, and area. For high-speed, low-voltage applications, each delay stage should provide enough gain (or drive) to prevent the signal transition slope or slew rate from being too sluggish. The transition time, rise time (t_{PLH}) + fall time (t_{PHL}), should be less than 40% of the cycle time for better signal quality and lower attenuation. Unlike analog implementations, which change delay by varying current for every stage, the all-digital approach preserves slew rate of the clock signal regardless of the delay range provided by the DLL.

Making a symmetrical transition ($t_{PHL} = t_{PLH}$) is very important for low duty-cycle distortion. A seemingly small (10 ps mismatch) for the delay stage can result in significant distortion at the DLL output. Because the mobility of P devices and N devices are significantly different, it's not easy to match the rise and fall time over process variation. Stresses seen during burn-in may result in negative bias temperature instability (NBTI), causing device mismatch and producing excessive duty-cycle distortion. Duty-cycle amplification along the critical timing path can make such mismatch even

worse. A two-input, NAND-gate-based delay line has lower overall duty-cycle distortion [2] than an inverter-based delay line in part because it doesn't require a MUX array to select an exit point. Matching interconnects between node A and B in Figure 11.3 is critical for achieving symmetrical transitions. The same logic state must be preserved for each delay element even if it is not used.

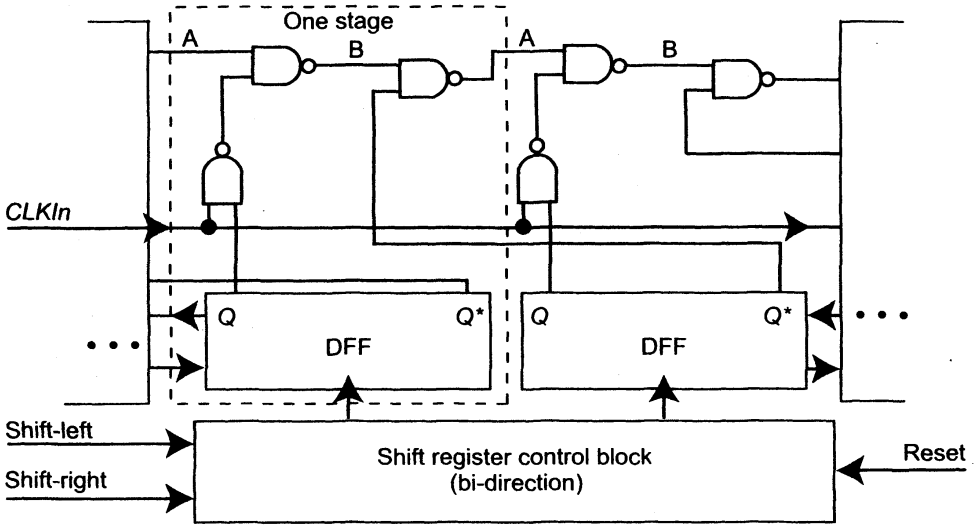


Figure 11.3 A symmetrical NAND-gate-based digital delay line.

For an entry-point selection delay line (like a register-controlled delay line), clock selection and loading is a potential problem. Because the input clock signal (*CLKIn*) is fed into every delay stage and only one entry point exists at a given time, as shown in Figure 11.4, the clock must be gated or treed for power saving. It is not trivial to control the input clock signal for reduced jitter and power consumption. Although only a NAND gate is shown in Figure 11.4 for entry-point selection, based on the shift-register design, other logic can be used to save power and prevent control discontinuity during shifting.

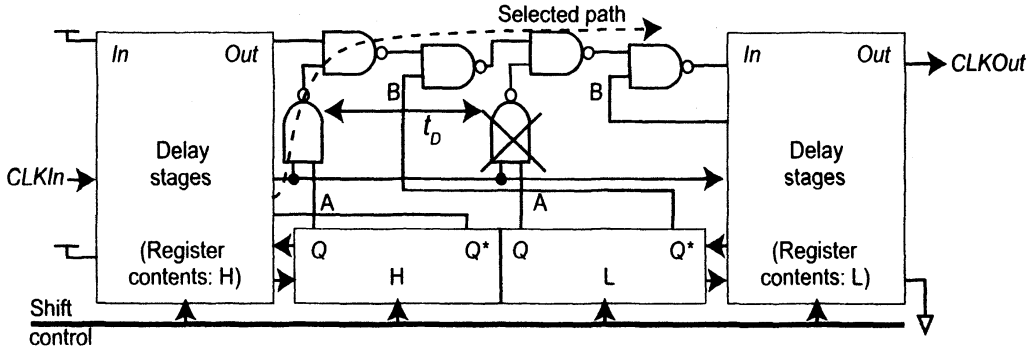


Figure 11.4 Register-controlled DLL with entry-point selection.

To reduce duty-cycle distortion, other configurations can be implemented, such as dual delay lines: one for rising edge clock and one for falling edge clock. Matching between two delay lines and excessive power consumption are concerns. A method of phase-locking to both the rising and falling edges of the external clock can be accomplished by comparing the falling edges after the rising edges of the reference and feedback signal are locked. The increased complexity of such an implementation, by using an extra DLL for duty-cycle correction, is anticipated when a high-performance ADDLL is required for a high-speed DRAM.

11.2.2.2 Lock range and input phase selection Lock range is also referred to as the *tuning range* of the DLL, t_{Lock}, which is the only design variable in Equation (11.3). When the system clock period (t_{CK}) is changed, t_{Lock} should be able to adjust the delay in the worst-case situation. For instance, the number of total effective delay stages, N, is determined by the lowest operating frequency. Assume that t_{CK} is equal to 5ns and delay per stage (t_D) is 100 ps (fast corner: high voltage and low temperature), then N can be calculated to 50, according to Equation (11.4). Taking t_{INTRI} into account, the number of effective delay stages is still close to 50, depending on t_{INTRI} and process variations. Equation (11.5) can be used to estimate the N used in the digital delay line.

$$N = (t_{CK,\,slow} - t_{INTRI,fast})/t_{D,fast} \tag{11.5}$$

While both frequency and process are factored in during initial lock, the delay line still needs to be able to track voltage and temperature (VT) variation. As shown in Table 11.2, the intrinsic delay can vary as much as 40% at lower voltage and higher temperature. The delay line should have sufficient tuning range to cover such variations. Some other applications may require a frequency change over a long period of time (called *frequency slewing*) for EMI reduction. Depending on the initial lock and lock point, the delay line needs to accommodate such frequency, voltage, and temperature (FVT) changes at the cost of extra intrinsic delay, power, and area.

The initial phase relationship between the internal clock and the feedback signal is uncertain at power up because of delay variations. After power up and reset, the delay line should be set to known initial conditions, which establish a unique clock propagation path. In general, we want n from Equation (11.4) as small as possible in order to reduce power dissipation and increase noise immunity. Therefore t_{Lock} should be limited to one clock cycle (t_{CK}) after lock, even though the delay line could provide a delay range more than two times the clock period (in some literature, it's referred

to as *false lock* when t_{Lock} is greater than t_{CK}). Assume N=50, t_{CK}=2.5 ns, and t_D=200 ps at the slow corner, the tuning range can be up to 10 ns, which is four times the clock period at higher speed.

For the register-controlled delay line [1, 2], the initial entry point is toward the right end of the delay line for the lowest intrinsic delay. To meet this design criterion, the phase detector (PD) should be modified to disable shift-right and, under that condition, initially only add delay to the delay line. However, setting the initial entry point at the right end of the delay line is undesirable. One scenario follows: initially $t_{CK} = t_{INTRI}$, and the DLL is locked with t_{Lock}=0. During the Read operation and under PVT variations, t_{INTRI} becomes greater than t_{CK}. The DLL wants to compensate for this change and instead of removing delay, it must add delay until $t_{INTRI} + t_{Lock}$ equals $2 \cdot t_{CK}$.

The DLL relocking requires more time, in this case, and creates timing uncertainty between the internal and external clocks. Several spare delay stages must be added to allow the DLL to subtract delay after the initial lock even if the lock point is close to the initial entry point. The spare delay stages, however, add more intrinsic delay (t_{buff}) and noise sensitivity to the timing path.

A better solution for high-speed DRAM is to permit the delay line to always add delay and to centralize the lock point initially. Depending on the initial conditions between the reference and feedback signal, the input to the delay line can be phase shifted to reduce the insertion delay and improve the efficiency of the delay line. The phase shift can be 180° (*CLK*180), as shown in Figure 11.5, which is merely an inverted version of the *CLK* signal. When the feedback signal (*CLKFB*) lags the *CLK* within a certain phase shift (Δ), e.g., 20°, or *CLKFB* leads the *CLK* within (180–Δ), e.g., 160°, the phase-shifted signal (*CLK*180) instead of the *CLK* can be used as the delay line input and the lock point centralized in the delay line for better FVT tracking. The insertion delay can be reduced from t_{CK}-t_{PE} to t_{CK}-t_{CKD}-t_{PE}. Ignoring intrinsic delay, the worst-case number of delay stages can be given by

$$N = \frac{t_{CK,slow}}{t_{D,fast} \cdot \phi} \tag{11.6}$$

where the ϕ is the phase-shifting factor. For 180°, ϕ equals 2. Using the previous example, t_{CK} = 5ns and t_D = 100 ps, the worst-case number of delay stages needed is reduced from 50 to 25. The worst-case insertion delay is

cut in half by simply adjusting the phase of the input signal. No spare delays are needed with this arrangement.

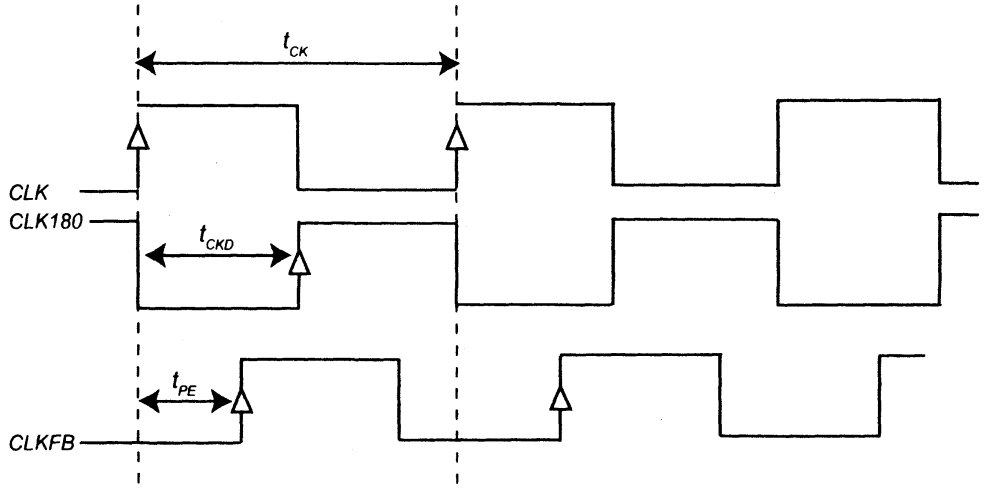


Figure 11.5 Input phase selection to improve the efficiency of delay line.

If a clock divider is used for improved clock delivery, multiple 90-degree, phase-shifted signals are available, as shown in Figure 11.6. The efficiency of the delay line, including power, layout area, and lock time can be further improved by selecting one of the clock signals (CK0, CK90, CK180, and CK270) as the input to the delay line. Clock selection only occurs once as the system is reset and can be disabled to save power after the selection is made. Using the prior example with an external clock period of 2.5 ns and an internal cycle time of 5 ns (based on Equation (11.6) $\phi = 4$), only 13 delay stages are needed, in theory, which is a 74% improvement over the original design. Duty-cycle correction (for two phases) or phase generation (for multiple phases) is required to retain all of the edge and timing information contained in the external clock.

Another benefit of input phase selection is power saving. The average dynamic power dissipation of the delay line is given by

$$P_{avg} = 2 \cdot n \cdot C_d \cdot f_{CLK} \cdot V_{DD}^2 \tag{11.7}$$

where n is the number of delay stages when the DLL is locked, C_d is the capacitance associated with each delay element, V_{DD} is the power supply voltage, and f_{CLK} is the operating frequency. Supply voltage is the biggest knob with regard to power reduction. This reduction may cost some speed and, due to smaller voltage swings, increase sensitivity to supply noise (Table 11.2). Reducing the clock frequency with an internal clock divider is another way to save power. If the clock frequency and supply voltage are

fixed, the only design choices remaining are n and C_d. Since P_{avg} is proportional to the number of active delay stages (n), power can be reduced by a factor of ϕ with appropriate input phase selection. To further save power when the DLL is not in use (during active power down, for example), a disable signal can be applied to stop the clock that is fed into the delay line. The DLL, however, must reacquire lock after the clock is re-enabled.

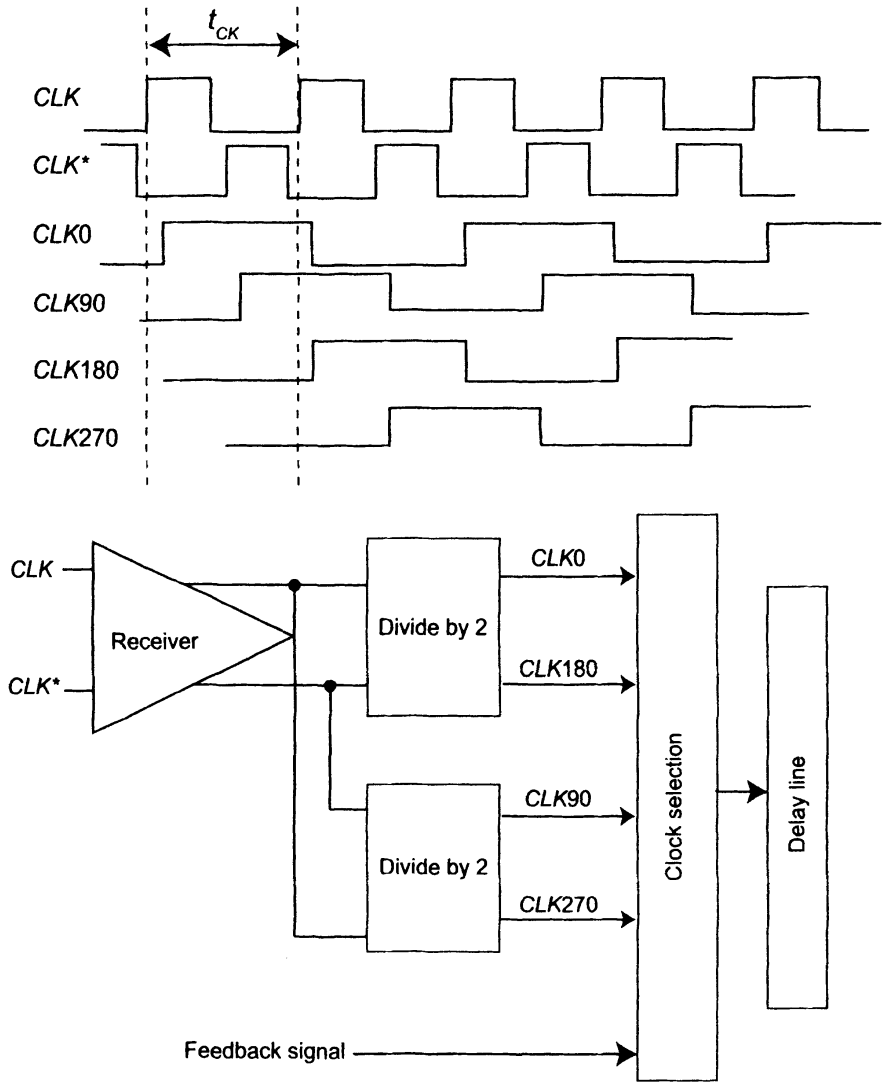


Figure 11.6 A clock path with clock dividers and four-phase input signals.

11.2.2.3 Delay resolution and jitter For ADDLL, jitter is primarily determined by the resolution the digital delay stages. The peak-to-peak jitter ($J_{P\text{-}P}$) and RMS jitter (J_{RMS}) can be defined as

$$J_{P-P} = \pm t_D, J_{RMS} \leq \frac{t_D}{2} \tag{11.8}$$

For a traditional delay line made up of logic gates, the minimum gate delay (an inverter) is approximately 50 ps. To preserve signal polarity, two inverters are needed, and peak-to-peak jitter will not be any better than 100 ps. This jitter is evident at the device outputs whenever the DLL makes an adjustment.

To improve resolution for high-speed operations, a finer delay element can be used, which in turn reduces jitter. Two examples of a fine delay element are shown in Figure 11.7: one with load adjustment and the other with path selection. By adding or removing a small load capacitor, controlled by a shift register, the delay can be varied dependent upon the selected load capacitance and the driver. A pair of N and P MOSFETs with their source and drain tied together, respectively, can serve as load capacitors. Delay resolution is determined by the unit load capacitance. The delay will be fairly constant over PVT variation, and the intrinsic delay is small (basically two inverters for eight stages in Figure 11.7). For the path selection delay element, a delta delay will exist between the slightly different-sized inverter strings. This topology offers better delay tracking over PVT variations. If connected in serial, the fine delay line introduces a large intrinsic delay, which is not good. An interleaved architecture using both interpolation methods is commonly implemented.

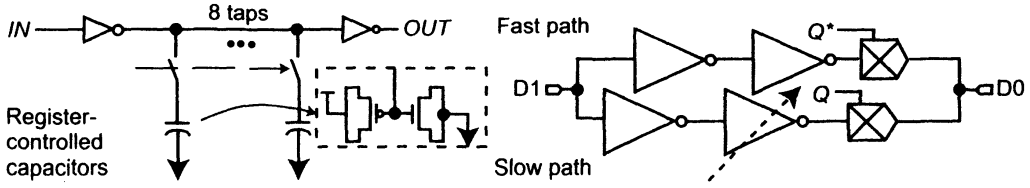


Figure 11.7 Two types of fine delay adjustments.

Another approach is shown in Figure 11.8. This design combines the outputs of two invert matrix blocks fed by two input signals (A and B) that differ in time by one unit delay (typically 100 ps) from the traditional delay line. By mixing the outputs together we create a delay block with fine edge control and low intrinsic delay. Each matrix block consists of a multitude of parallel, but differentiated delay paths. A control code determines the overall delay of each matrix and, in essence, the slew rate at their outputs. This approach is called *phase blending* in some literature [12].

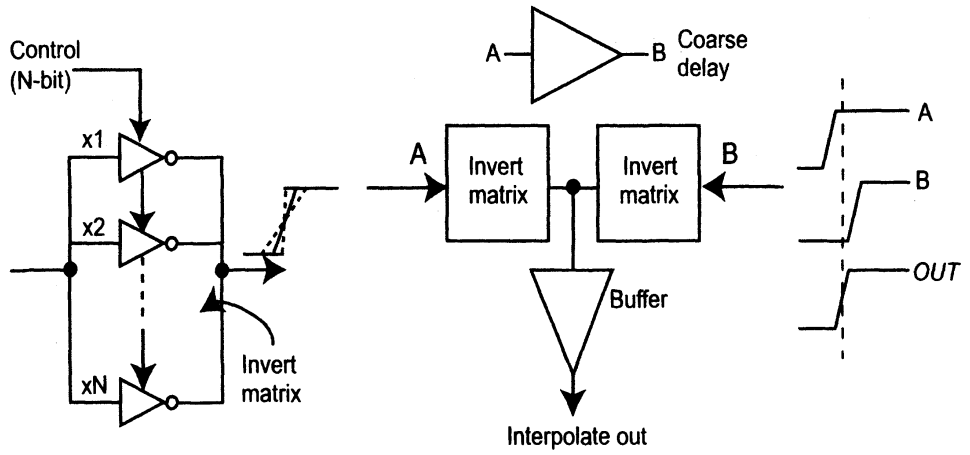


Figure 11.8 Phase mixing by adjusting slew rate and driving strength.

The tuning range of the fine delay should cover the delay variation of one coarse delay stage in either direction: up or down. The step size (unit delay) of the fine delay line (t_{FD}) is a fraction of the coarse unit delay (t_D). A separated loop is required to perform fine delay adjustment in addition to the conventional DLL loop, referred to as a coarse loop. Dual-loop architecture provides an efficient way to achieve wide lock range (via a coarse loop) and tight synchronization (via a fine loop). Smooth transition between the coarse and fine adjustments is a challenge and may cause a large phase jump or discontinuity of the DLL output.

11.2.3 Phase Detector (PD)

A phase detector (PD) is the decision maker in DLLs. The PD discriminates phase error between the reference and feedback signals and adjusts the delay line accordingly. For an all-digital implementation:

1. No integration happens in the PD.
2. Mutual exclusive signals are needed to control the all-digital loop.
3. Such control signals should be well-defined digital signals.

One concern is that the resolution of a DLL is limited by the unit delay stage, which needs to be factored into the PD design to prevent further action if the phase error is within a certain boundary (sometimes referred to as a *window detector*). Loop dynamics, including lock time, tracking time, and stability are determined in large part by the phase detector design.

11.2.3.1 Lock time Lock time, measured in clock cycles, is another important design aspect. Two situations must be considered:

1. After a DLL is reset and enabled, how long will it take to initially lock the loop before a valid Read command can be issued?
2. After initial lock (if the lock is lost due to voltage and temperature drift), how long will it take to regain the lock condition?

To answer these questions, we must determine the sampling frequency at which a digital phase detector (DPD) will make DLL adjustments. For a high-speed, low-voltage memory system, the intrinsic forward path delay (t_{INTRI}) could be several times the clock period, as shown in Table 11.1 and Table 11.2. The loop must be allowed to settle down after a delay adjustment (the input signal propagates through the delay path and feeds back into the DPD, sometimes referred to as *loop settling time*). To achieve this, the sampling frequency (f_S) must not exceed

$$f_S = \frac{1}{t_{CK} \cdot k} \tag{11.9}$$

where the sampling factor k can be chosen according to

$$k = INT\left(\frac{t_{INTRI,\,slow} + t_{Lock,\,slow}}{t_{CK,\,fast}}\right)_{EVEN} + k_0 \tag{11.10}$$

The subscript *slow* refers to a slow corner condition, while $1/t_{CK,fast}$ refers to the highest operating frequency. The function $\text{INT}(x)_{\text{EVEN}}$ makes the x to the closest even integer, for example, $x = 0.6$ gives 0, and $x = 1.4$ gives 2. The k_0 can be either 1 or 2. The side effect of this sampling factor is an increase in the lock time by a factor of k.

Making adjustments beyond the desired sampling frequency causes stability problems for a feedback system. Because the sampling frequency is determined by the highest operating frequency, it may be too pessimistic for low-speed operations. Dynamically adapting the sampling frequency based on clock frequency and intrinsic delay is possible by measuring the loop delay under varying PVT conditions. As part of the DPD, a digital filter like an up/down counter or shift register can be used to control the sampling frequency.

With the help of the previous discussion on lock range and input phase selection, the worst-case lock time can be determined by

$$t_{L,\,worst} = \frac{N \cdot k \cdot t_{CK}}{\phi}$$

(11.11)

where N is the number of total effective delay stages and ϕ is the phase selection factor. If $N = 50$, $k = 5$, and $\phi = 1$, it will take 250 clock cycles to initially lock the loop, which may be unacceptable (DDR and DDRII SDRAM usually specs 200 clock cycles for initial lock time). With input phase selection, the lock time can be reduced to 125 cycles (with 180° phase shift) or 65 cycles (with 90° phase shift, $\phi = 4$).

Another way to reduce lock time is to perform *segment selection* (grouping several delay stages into a segment) prior to locating the entry delay stage. After the segment is located, the final entry point can be found within that segment. The number of delay stages grouped into a segment (S) can be determined by

$$S = INT\left(\frac{t_{CK,fast}}{2 \cdot t_{D,\,slow}}\right)$$

(11.12)

For example, assume that the fast clock period is 2 ns and the delay per stage at slow corner is 200ps, then S is equal to 5. The lock time with segment selection is given by (worst-case):

$$t_{L,\,worst} = \left(\frac{N}{S} + S\right) \cdot k \cdot t_{CK}$$

(11.13)

For the same parameters used in the previous example, the new lock time calculates to 75 clock cycles, which is four times faster than the lock time without the segment and phase selections. For a high-speed system, the ratio of t_{CK} to t_D is small, somewhat reducing the benefits of segment selection. Initial lock time can also be reduced by performing a binary search with a *successive approximation register*, or SAR. Local feedback (bypassing the clock tree and *I/O* model delays) can speed up the initial adjustment by making continuous changes to a dummy delay line. Also it's possible to make loop adjustments proportional to the magnitude of the phase error without impacting loop stability. This would require an adaptive PD capable of selecting big (several stages) or small (one-stage) adjustments based upon phase error.

Another method for quick initialization is the synchronous mirror delay (SMD), as shown in Figure 11.9. Two delay lines are used in the SMD: one for measuring the timing difference between the clock period and the intrinsic delay (*I/O* and clock tree) and another for the proper insertion delay based upon mirroring the timing measurement made in the first delay line. The SMD can generally acquire lock within two clock cycles at the cost of an extra delay line. For high-speed operation, the adjustment may require several cycles to be evident at the output. The delay model is a significant challenge for an open-loop sync circuit such as this.

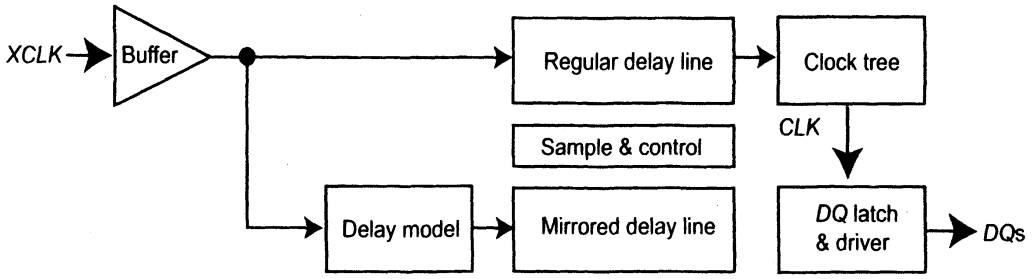


Figure 11.9 Block diagram of a synchronous mirror delay (SMD).

A closed-loop quick initialization scheme, called *measure-controlled delay* (MCD), is illustrated in Figure 11.10. During the initialization, the internal reference clock (*CLKREF*) is selected by a MUX as the output signal. The feedback signal (*CLKFB*) is sampled inside the measure delay line by the *CLKREF*. The timing difference between *CLKFB* and *CLKREF* is converted into digital code by taking a snapshot of each measure delay line output. The digital code stored in the control logic is mirrored to select the proper entry point of the DLL delay line. After initial locking and loop settling, the output of the regular delay line is picked as the output, and the measure delay line is disabled for power saving. Normal DLL operation is resumed by using a phase detector (not shown) to track the VT variations. The initial lock time is like that of the SMD (just a couple of cycles), which makes it attractive for quick recovery from power-saving modes.

The lock time for the SMD or MCD is fixed and independent of clock frequency. With dual-delay line structure, the efficiency of the delay line, especially the area, can be further improved by using input phase selection. Any mismatch that exists between the two delay lines can be mitigated by adding a PD to the MCD, as previously mentioned.

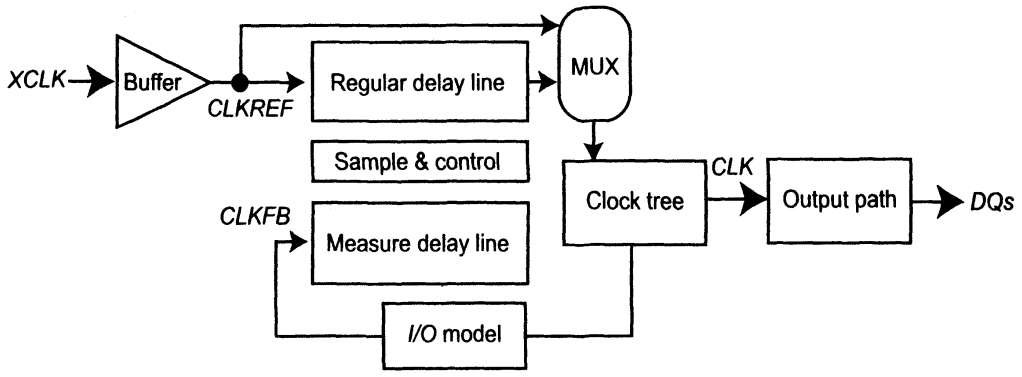


Figure 11.10 Block diagram of a measure-controlled delay (MCD).

After initial lock, a loop can lose lock due to excessive frequency, voltage, and temperature (FVT) variations. Generally, a DLL can dynamically track out these variations and adjust delay to relock the loop. Depending on the sampling frequency and the magnitude of the phase error, the tracking time may vary. After initial lock and during memory Read operations, it's undesirable to make a big delay change that appears as timing jitter at the outputs. Rather, small adjustment steps, which improve jitter performance, tend to extend tracking time. While sync circuits are designed to track long-term FVT variation, proper filtering is important to ensure loop stability.

11.2.3.2 Circuit implementations For a coarse delay line, the best resolution is a two-gate delay. The delay line is primarily used to ensure adequate tuning range and proper lock. A PD block diagram used for course adjustment is shown in Figure 11.11.

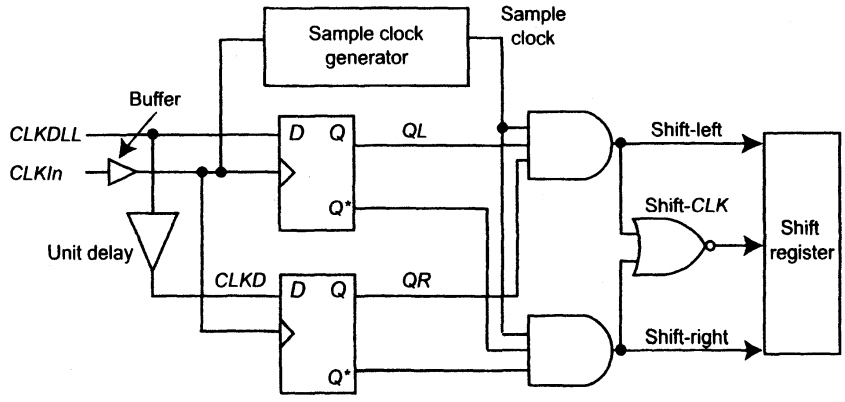


Figure 11.11 Block diagram of a PD used for coarse adjustment.

To meet the setup time requirements needed by the input edge-triggered D flip-flop (DFF), the unit delay in Figure 11.11 can be increased above the unit delay used in the delay line to prevent jittering (bouncing back and forth in the delay line) under all conditions. A buffer for the *CLKIn,* which

is a delay approximately one-half of the unit delay, is useful to improve the jitter performance and helpful to catch correct data for the DFFs when the DLL is close to lock. The sampling frequency (Equations (11.9) and (11.10)) for this PD depends on the operating frequency and intrinsic delay of the loop.

A timing diagram in Figure 11.12 illustrates three different situations: shift-left, shift-right, and adjusted (locked). An up-down counter or shift register can be added as a digital loop filter to average out the intermittently generated *QL/QR* to further stabilize the loop operation. After initial lock, the sampling or filtering rate can be adjusted for improved tracking capability. This DPD can also be frozen (no shifting) by disabling the sample clock to prevent roaming caused by power supply noise and jittering due to coarse shifting. For high-speed phase detection (HSPD), the feedback signal (*CLKDLL*) may distort and cause malfunctions to occur. It is a always good practice to divide the clock down internally for high-speed phase detection.

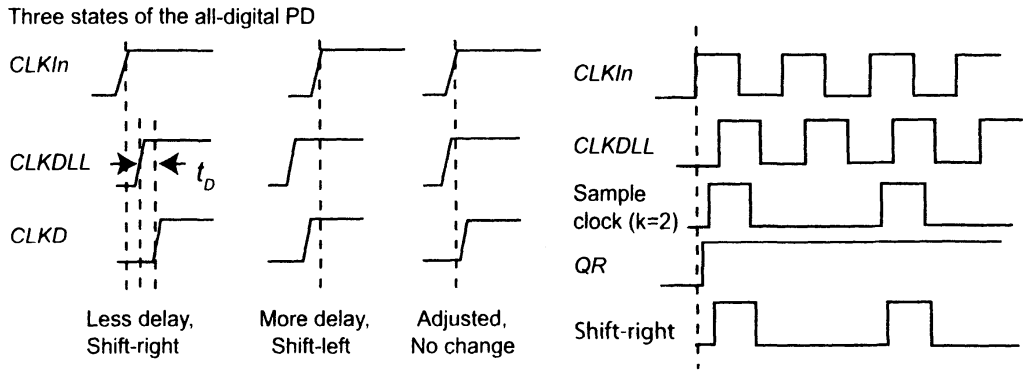


Figure 11.12 Timing diagram of a coarse DPD for an ADDLL.

An arbiter-based fine PD (FPD) in Figure 5.30 can be used to discriminate small phase error for a fine loop. The arbiter-based FPD can identify phase error as small as 10 ps with mutual exclusive outputs. A digital filter can also be used in this FPD to provide stable operation. A window detector can also be configured using two FPDs, as shown in Figure 11.13. If the feedback is within a delay window (defined by the fine delay t_{FD}) from the reference, the decision made by each FPD will be different and no fine adjustment (shift-right or -left) should be made.

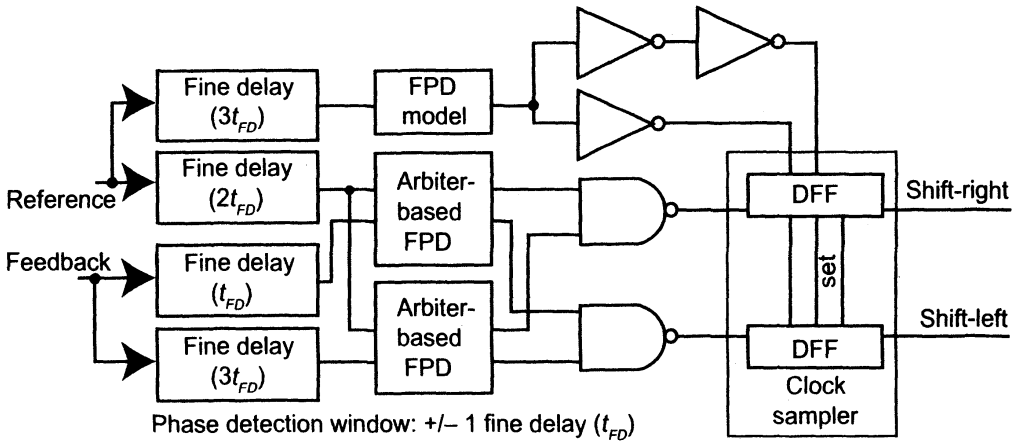


Figure 11.13 Arbiter-based phase detector with a fine window detector.

The FPD is active only when the coarse loop is locked. The coarse PD can take over the control to correct large phase error while the fine loop is active. Extra time is needed to lock a fine loop. Similar searching algorithms to those previously described can be used if a large number of fine delay stages are used.

11.2.4 Test and Debug

Test and debug for ADDLL circuits is very important, and proper test functions must be integrated into the design for both coarse and fine PDs. The most important test issues include loop initialization, loop dynamics (lock range, lock time, and jitter), and tracking capabilities. Most test functions can be implemented by using test modes and fuses. Test-for-speed is generally a big challenge and requires a compromise between test resources and functionality. Suggested procedures and techniques follow:

1. Characterize the coarse or fine delay line by disabling the respective PD and performing manual shifting.
2. Monitor the loop dynamics by outputting shift control signals to designated output pins for packaged parts. The most important signals include DLL lock, shift controls (shift-left, -right), etc.
3. Check the timing specs under different PVTF conditions by monitoring a free-running output data strobe.
4. Check different initialization schemes by enabling or disabling different functional blocks, for example, MCD initialization, input phase selection, and segment selection. It is desirable to share the same timing path with different control logic for evaluation, test, and debug.
5. Build a DLL bypass path for low-speed functional test.

6. Check various power-saving modes versus DLL functionality. Lock time and loop dynamics can be observed in Step 2.
7. Adjust *I/O* models to meet t_{AC} timing specs. Static adjustment can be taken outside of the DLL intrinsic path as illustrated in Figure 11.14.
8. Check DLL tracking capability (track time, jitter, etc.) by varying voltage and temperature (VT). The VT tracking can be tested in three configurations: coarse loop only, fine loop only, and dual-loop (both coarse and fine) at various frequencies. Different sampling frequencies can be analyzed based on the clock speed.

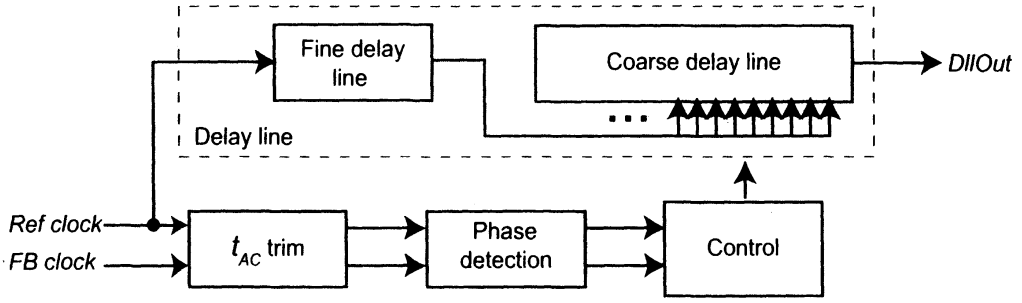


Figure 11.14 Block diagram of a dual-loop ADDLL with t_{AC} trim.

11.2.5 Dual-Loop Architecture

A DLL with dual-loop architecture generally has one loop for wide lock range and one for better timing resolution. A transition occurs when one loop (i.e., the fine loop) reaches its limit and interactions between the two loops are necessary. Smooth transitions are essential for better performance. Other concerns when designing a dual-loop DLL are tuning range, tracking time, power, and area.

For an all-digital implementation, the dual-loop DLL is made up of a coarse loop and a fine loop. Each loop has its own delay line and phase detector (shown in Figure 11.14 and Figure 11.15). The initialization always starts with the coarse loop. After the coarse loop is locked, the fine loop is enabled to dial in the timing with its higher resolution capability. It is costly to rely on only the fine delay for VT tracking. When the fine delay line reaches its end, transitions must be made to activate the coarse adjustment and reset the fine delay for further tuning. Depending on the coarse and fine delays at different PVT corners, timing discontinuity can occur, which, in turn, could violate timing specs due to clock glitches and jitter. It is not trivial to monitor the ratio of the coarse and fine delays and dynamically update and synchronize the transitions. Without resetting the fine delay line, phase mixing between the two signals (one coarse unit delay apart, as shown in Figure 11.8) is a preferred method for producing seamless transitions, that is, provided that a proper control signal can be timed to select the clock entry point in the regular coarse delay line.

Another preferred method uses dual-delay paths, as illustrated in Figure 11.15: one path for active fine delay adjustment and the other path for both coarse and fine adjustments offline. An extra fine PD can be used to guarantee that the phases of the dual delay line outputs are close to alignment when switching. The internal time constant is smaller than the DLL loop time constant, and the phase error can be adjusted quickly after a coarse shift is made at the inactive delay line. The coarse adjustment will only happen in the background (e.g., the coarse delay 2 in Figure 11.15) and won't impact the active timing path (e.g., the coarse and fine delay 1 in Figure 11.15). The transition is seamless and occurs when:

1. The active fine delay hits a certain boundary.
2. The phase error between two delay lines is within one fine unit delay.
3. A switch signal (SEL2) is timed with the falling edge of the output signal from the fine delay line.

The regular delay line gets adjusted only during initialization and won't change after initial lock. The coarse delays in the dual-delay lines provide the proper VT tuning range at the cost of extra intrinsic delay, power, and area consumption.

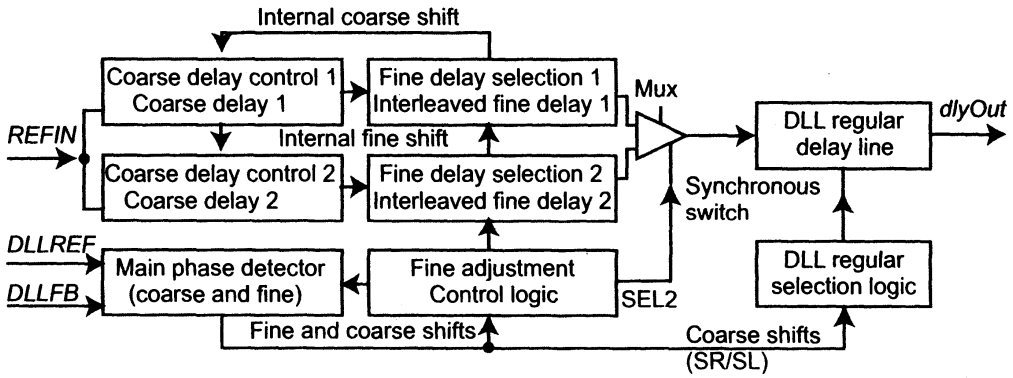


Figure 11.15 Dual-path fine and coarse adjustments with smooth transitions.

11.3 MIXED-MODE CLOCK SYNCHRONIZATION DESIGN

A basic delay-locked loop has only two building blocks: delay line and phase detector (PD). The delay line provides an adjustable delay, and the PD directs how to vary the delay. To synchronize data outputs with a system clock in memory devices, an *I/O* model is also included in the feedback path. For high-speed DRAM, extra circuits like clock dividers and edge recovery circuitry (e.g., a phase generator), as shown in Figure 11.16, must be included to improve performance. The core DLL can be left intact, and the design rules previously discussed for the digital DLL can remain the same due to similar internal clocking speeds. However, with a clock divider,

edge information for the internal clock must be recovered for full-speed *I/O* operation. Multiple phases from the phase generator can represent the external full-speed clock and are used to time the output data path accordingly. An analog DLL with a voltage-controlled delay line (*VCDL*) is generally suited for such applications. Combining both digital and analog implementations, a mixed-mode DLL is investigated in this section with a focus on analog DLL design.

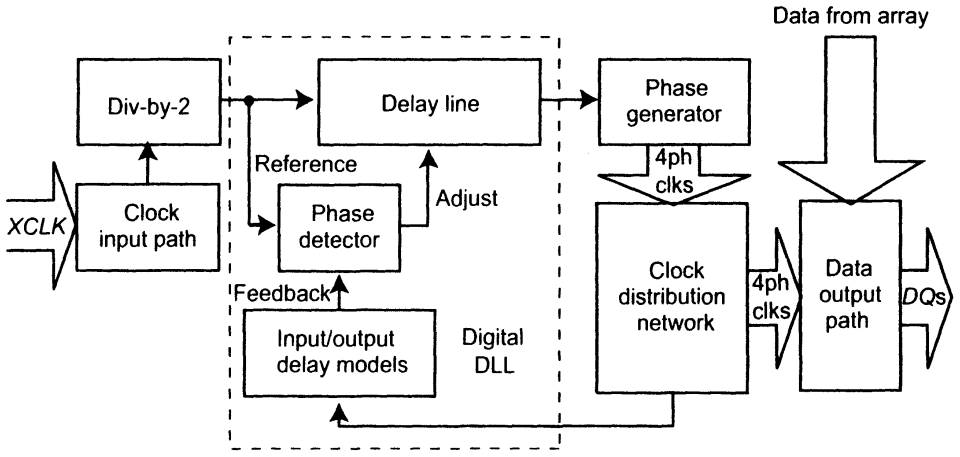


Figure 11.16 Block diagram of a high-speed, mixed-mode DLL.

As the timing window gets smaller and smaller for high-speed interfaces, the focus shifts somewhat from just output timing to *both* input and output timing. The entire data path, from command address capturing to data output at specific *CAS* latency, must be taken into account for floor planning and timing analysis. Design changes based on interactions between the data path and timing circuits are possible. These may include the need for more than one DLL, depending on the timing budget, flexibility, power, and area constraints.

11.3.1 Analog Delay Line

Compared to a digital delay line (DDL), an analog delay line (ADL) has a different configuration. The delay is adjusted by changing the delay per stage (t_D), not the number of stages (n) (Equation (11.4)). The delay adjustment is continuous for an ADL based upon an analog control signal while adjustment for a DDL is discrete and based on the minimum delay resolution. A block diagram of a voltage-controlled delay line (*VCDL*) is shown in Figure 11.17 with N delay stages. If the timing difference between *CLKIn* and *CLKOut* is locked to one clock cycle (t_{CK}), then each stage can provide a nominal t_{CK}/N unit delay. For periodic signals, one cycle of delay provides a 360° phase shift. Various phase shifts can be generated based upon different numbers of N. The ADL is well suited for multiple phase generation.

Fewer delay stages make the ADL smaller than a DDL, but with comparable tuning range for clock synchronization. Phase interpolation between two adjacent taps is generally required to achieve full-range (360°) coverage and precise de-skewing. For high-speed DRAM, analog implementations become the only choice in some applications.

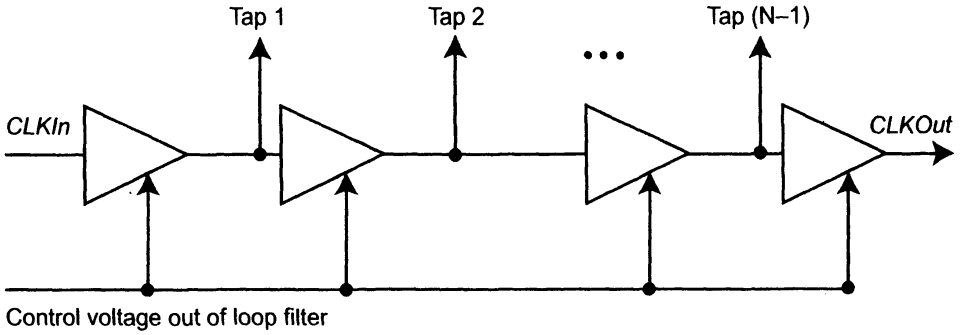


Figure 11.17 Block diagram of a voltage-controlled delay line (*VCDL*).

11.3.1.1 Unit delay Choosing a proper unit delay is the first step in building a delay line. An analog control signal (e.g., a bias voltage) is generally applied to vary a current (I_{tail}) used to charge or discharge internal node capacitance (C_n). A unit delay (t_D) can be defined as the time it takes an internal node to be charged or discharged by the bias current I_{tail}.

$$t_D = \frac{C_n}{I_{tail}} \cdot \int dV \tag{11.14}$$

A single-ended, current-starved delay buffer is shown in Figure 11.18. The bias voltages V_{BP} and V_{BN} control the current source and sink, respectively. C_i is the intrinsic capacitance associated with the input MOSFET MP and MN (current stage and next stage). C_v can be a voltage-controlled capacitor. A fully differential delay, also referred to as *current mode logic* (CML), is also shown in Figure 11.18. A voltage-controlled resistor (R_L) can be realized by a P-type MOSFET pair, which is biased on the triode region. A poly resistor can also be chosen for better linearity and greater voltage margin at the cost of larger layout area, matching, and process sensitivity. Within the linear operating region, there is negligible duty-cycle distortion and better common-mode noise rejection for the fully differential delay as compared to single-ended. The signal swing doesn't need to be rail-to-rail for the differential approach. However, level translation is required to bring the signal levels back to full CMOS.

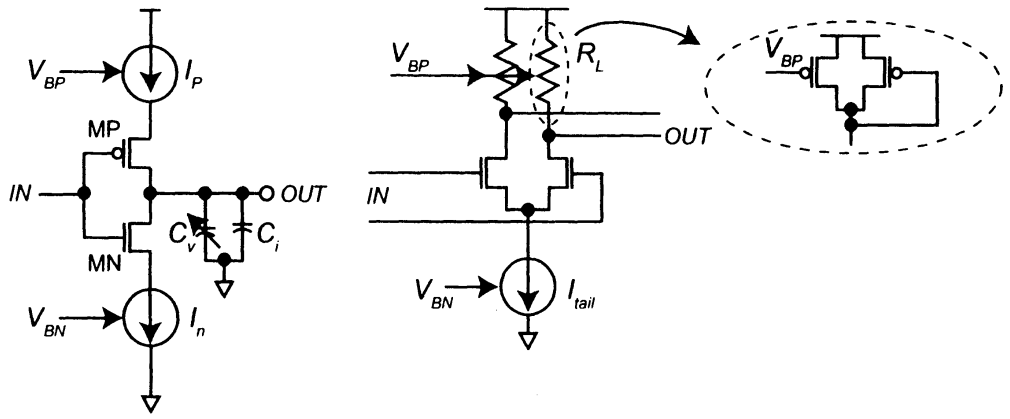


Figure 11.18 Single-ended (left) and fully differential (right) analog unit delays.

It is usually difficult to control the bias voltage over process, voltage, and temperature variation. Self-bias techniques [3], shown in Figure 11.19, can be applied to the differential delay buffer for:

1. Better dynamic supply noise rejection.
2. Better linearity for delay versus control voltage.
3. Elimination of bandgap or other bias generators.

The control voltage (V_{ctrl}) is fed from the phase detector and loop filter. The bias voltage V_{BP} for the loads can be generated from V_{ctrl} with the aid of an op-amp. A replica delay is used to generate V_{BN} for the current sinks. The same current is mirrored inside the delay buffer to produce equal rise and fall times. For a self-bias fully differential buffer, the unit delay can be given as

$$t_D = C_n \cdot \frac{1}{g_m} = \frac{C_n}{\sqrt{k \cdot I_{tail}}} = \frac{C_n}{k \cdot (V_{DD} - V_{ctrl} - V_{TP})} \tag{11.15}$$

V_{TP} is the threshold voltage and k is the transconductance of the PMOS device. The unit delay has better supply noise rejection if self-bias techniques produce a constant current.

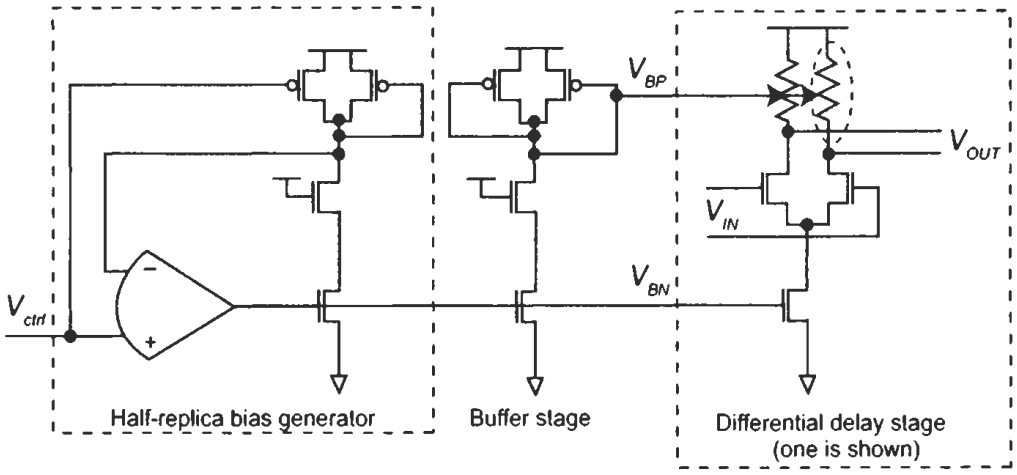


Figure 11.19 Self-bias generators with a replica delay buffer.

The unit delay (measured from the crossing points of the differential signals) versus control voltage at different process and temperature corners is plotted in Figure 11.20 for a clock speed of 625 MHz and a supply voltage of 1.2 V. A slow process corner (SS) and lower temperature (–40°C) make the delay tuning range smaller due to limited voltage margin (higher threshold voltage). The signal may get severely attenuated due to sluggish transitions and limited gain. A general rule of thumb is to keep the unit delay less than 15% of the cycle time (t_{CK}) at worst PVT corners. Fast process corner (FF) and high temperature (110°C) may cause problems for low-speed operation because the delay is entering a nonlinear operating region.

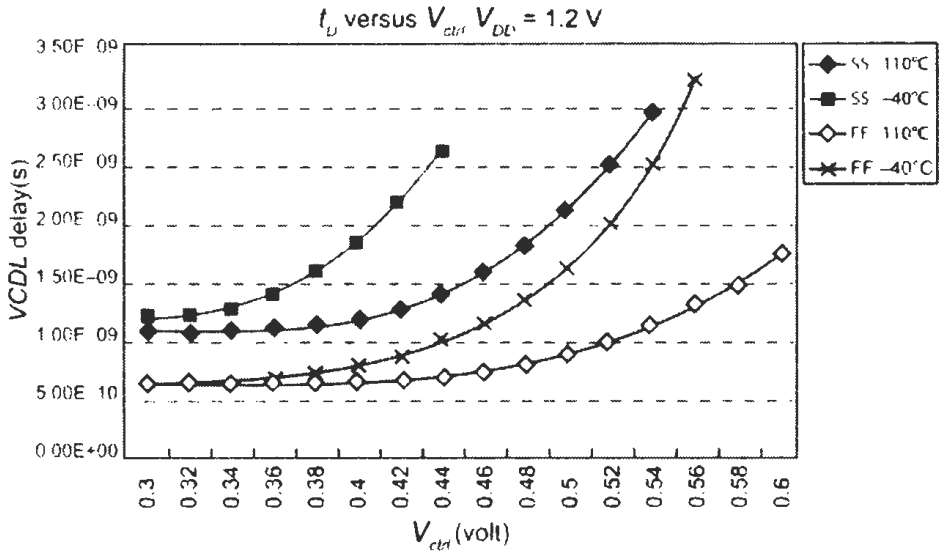
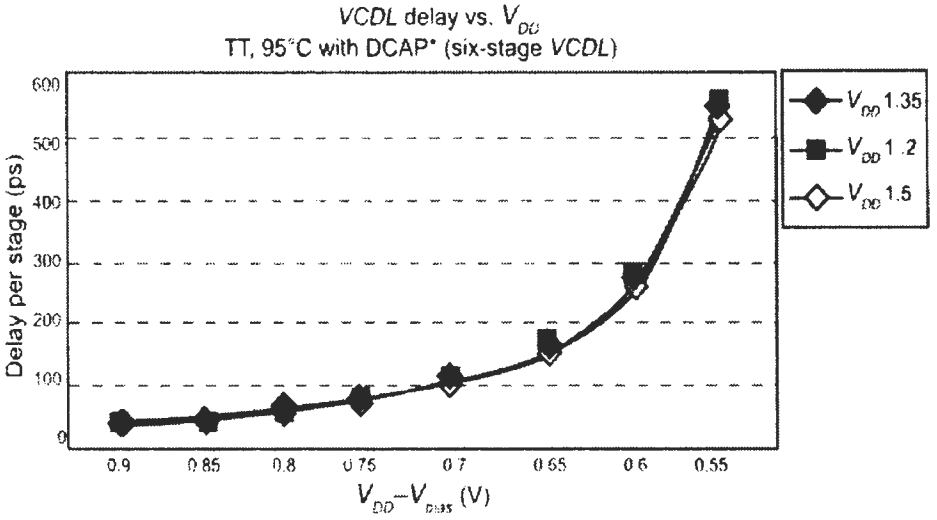
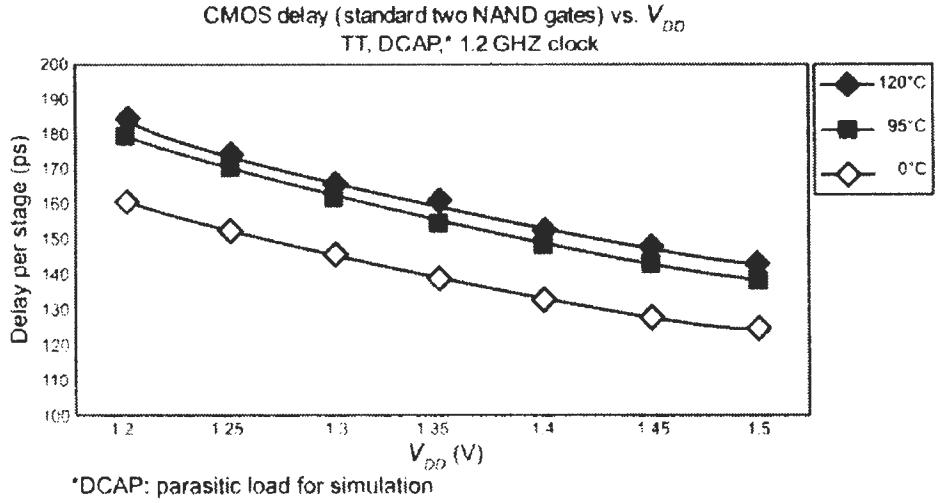


Figure 11.20 Unit delay versus control voltage (V_{ctrl}) at different corners.

A comparison between digital and analog unit delays versus supply variation is shown in Figure 11.21. CML-based analog delays have much better supply noise rejection than its digital counterpart. Analog delays can also be biased to achieve the same delay across different process corners, that is, to achieve consistent latency control.



a)



b)

Figure 11.21 Supply noise sensitivity for both a) analog and b) digital delays.

11.3.1.2 Gain and lock range The gain for the self-bias differential delay line is given as [3]

$$G = \frac{n \cdot C_n}{2 \cdot I_{tail}} = \frac{n \cdot C_n}{k \cdot (V_{DD} - V_{ctrl} - V_{TP})^2} \tag{11.16}$$

where n is the number of delay stages and I_{tail} is the bias current. The output from each delay stage can swing from V_{ctrl} to V_{DD}. The system may not be stable when the V_{ctrl} is approaching V_{DD}–V_{TP} (minimum swing at the output). Furthermore, V_{ctrl} must not exceed a specific threshold determined by the process and operating conditions in order to keep the differential input transistors in the saturation region.

The lock range of the voltage-controlled delay line (*VCDL*) is $n \cdot t_D$, where t_D is the unit delay defined in Equation (11.15). Although the number of stages (n) for analog delay line is fixed, there is a minimum requirement to keep the delay per stage (t_D) within the linear region. If the transition time is comparable to the bit time (half cycle), the output may not reach full swing, and the signal could get attenuated so badly that no valid output would be generated. The rule of thumb is to pick the n greater than eight. Locking to a different clock phase can be used to eliminate this problem, just like the input phase selection used in digital delay line. Instead of locking to 360-degree phase (a full t_{CK}) of the *VCDL*, the differential feedback signals can be swapped to achieve 180-degree (0.5 t_{CK}) locking.

With the same clock frequency, the effective delay per stage is 50% less, and the transition time is always less than the bit time, which provides better noise margin. Figure 11.22 shows two lock regions under different configurations: an eight-stage *VCDL* locking to 360° and a six-stage *VCDL* locking to a 180° phase. It is obvious that the 180° lock has better noise margin and more symmetric transitions ($t_{PLH} = t_{PHL}$). However, duty-cycle distortion from the input must be addressed to achieve precise phase alignment in this case.

There is also an upper limit for the *VCDL* stages. Lock range under different PVT corners generally requires a longer delay line at the cost of large area and more power. A comparison between eight- and sixteen-stage delay lines is plotted in Figure 11.23 with regard to slow simulation corners and linear operating region. The lock range can vary from 425~980 MHz (eight stages) to 220~500 MHz (16 stages). Fewer delay stages are desirable for high-speed operation. To keep the gain constant, an adaptive scheme can be applied based on the frequency and PVT corners. The minimum *VCDL*

delay is compared to the full-speed clock to determine the internal clock speed. The *VCDL* delay can be specified with a scaling factor m, which can be controlled by a programmable clock divider. The phase shift per stage is adjustable based on the ratio of m/n.

$$n \cdot t_D = m \cdot t_{CK} \tag{11.17}$$

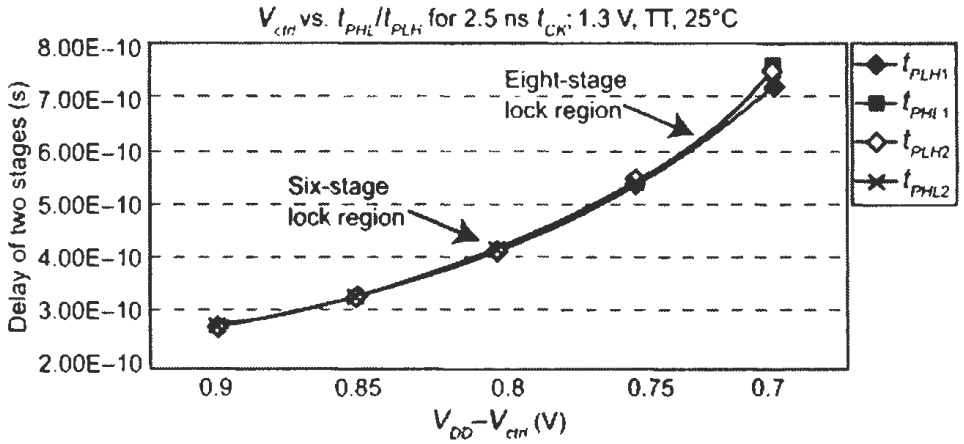


Figure 11.22 Lock range and delay versus control voltage for two different *VCDLs*.

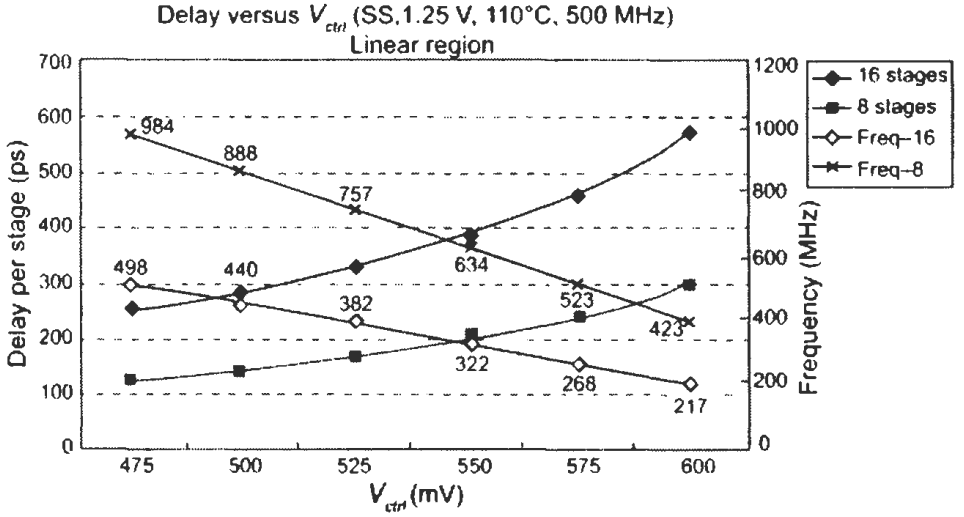


Figure 11.23 *VCDL* delay versus V_{ctrl} at different conditions.

11.3.1.3 Noise and jitter The *VCDL* with current-mode logic (CML) has better supply noise rejection than the digital delay line with CMOS logic. With 150 mV lower supply voltage (–10%), the CML delay line (five stages with loads) has delay variation only +2.1%, while the CMOS inverter

chain (four inverters with loads) has +12.9% variation. Under the same conditions, the level translation from CML (differential) to CMOS (single-ended, CML2CMOS) has 10% delay variation. Depending on the output swing, common-mode range and PVT corners, the CML2CMOS may generate duty-cycle distortion at the output. This is not an issue when only the rising edges of the multiple phases from the *VCDL* are used for clocking. For an all-digital implementation, supply noise contributes 76% of total timing variation (Table 11.2). A timing path with extensive CML delay elements is a good alternative for high-speed DRAM design.

To improve the jitter performance and noise immunity, several design and layout techniques for the *VCDL* are addressed as follows.

- Be careful when selecting the delay element. A device with longer channel length (L) has better matching and noise performance at the cost of speed. Current loads and sinks usually use longer L devices. Devices with lower threshold voltage offer reduced phase noise and improved voltage margin.
- Remember that large tail currents (I_{tail}) improve speed and reduce noise and gain (Equation (11.16)) at the cost of power. Trade-offs must be made between speed, lock range, gain, power, tracking bandwidth, and the number of delay stages.
- Match the layout. Differential inputs use an interdigitated or common centroid layout [8]. Minimize parasitic capacitance and interconnects. Group all identical circuits into a cell for better matching and LVS (layout versus schematic). Keep every delay cell oriented the same, and use the same layer of interconnects. Identical circuits can be used for load and voltage-level matching at the cost of power.
- Isolate and shield all analog signals, i.e., V_{ctrl}, etc. Shield all high-speed matched clocks. Put guard rings around critical analog blocks (*VCDL*, bias generator, etc.). Use quiet and isolated power buses for critical analog circuits. Avoid digital routes across sensitive analog blocks.
- Placement is another key. Keep the clock output buffers (digital, high-speed) far away from the CML delay line (*VCDL*) and level translators (CML2CMOS). Identify and partition the circuits into high-speed clocks, digital control logic, and sensitive analog circuits. Dedicated power pins for analog circuits are imperative for better performance.

11.3.2 Charge-Pump Phase Detector (CPPD)

To generate an analog bias voltage for the *VCDL*, a charge-pump phase detector (CPPD) is generally used along with a loop filter, shown in

Figure 11.24. Based on the outputs of the phase detector, a current (I_{pump}) can be pumped into or out of the loop filter (LF), which is a simple capacitor for the DLL. When the loop is locked, there is no net change in charge within the LF, and V_{ctrl} remains relatively constant. The CPPD shown in Figure 11.24 is prone to charge-sharing problems. A fully differential charge-pump with a unit-gain source follower [4] is useful and shown in Figure 11.25. A current-steering technique [6] can also be used to reduce ripple at the LF output and improve jitter performance. To avoid a "dead zone" [3–4] when a phase is close to lock, keep the up-and-down pulses from the PD the same, with the same duration, depending on PVT variations. The best resolution available from a CPPD depends on the control voltage ripple, loop capacitance, matching etc., and is usually 10–20 ps for all practical purposes.

Phase comparisons can be made on both rising and falling edges to reduce lock time given there is little duty-cycle distortion. The lock time for the CPPD is usually longer compared to the digital DLL, depending on the I_{pump} and C_{LF}. A large loop capacitor usually improves loop stability (a first-order system with a pole provided by the LF) but slows down the locking procedure. Based on initial conditions and clock frequency, a typical lock time for an analog DLL is approximately 50~100 cycles. If no clock is running during power-saving modes, there is a "run-away" problem for the bias control signal (V_{ctrl}), which is generally floated. A digital-to-analog converter (DAC) can be used to store the bias information and reduce the relock time after exiting power-saving modes. The extra complexity, power, and layout area must be traded against performance gains.

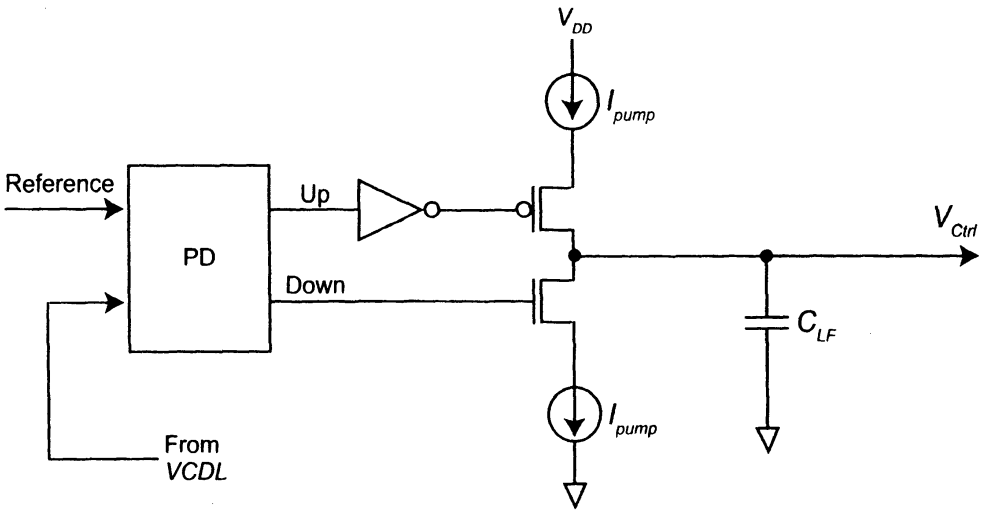


Figure 11.24 Block diagram of a charge-pump PD (CPPD).

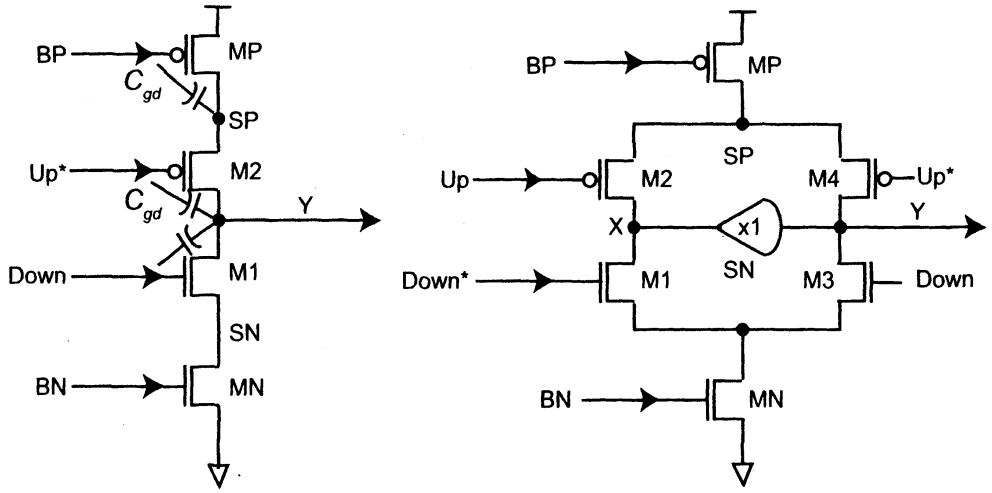


Figure 11.25 Charge sharing and a potential solution for CPPD.

A potential "false lock" problem can exist with the *VCDL*, in which the total *VCDL* delay equals a multiple of clock cycles. False-lock detection is needed for the CPPD to provide correct phase generation. Initializing the *VCDL* with minimum delay is another way to prevent false lock at the cost of longer lock times under certain conditions. "Fast-down" mode is useful to speed up the initial phase acquisition by ignoring the CPPD until a certain phase relationship is reached. Loop stability must be ensured with regard to lock time. When a digital DLL (DDLL, for synchronization) and an analog phase generator (APG, for edge recovery) are both used in a mix-mode clock synchronization system, a lock signal can be generated to start the DDLL synchronization when the phase difference of the APG is within a certain timing window. The overall lock time for the mixed-mode DLL is longer compared to each individual DLL.

The loop response time for the APG is approximately one cycle because the feedback is taken from the output of the last delay stage. Except for the CML delay line, the single-ended to differential converter, CML2CMOS logic and clock buffers remain subject to supply noise. Voltage and temperature variations are tracked quickly, and tracking time is usually within two cycles.

For the mixed-mode DLL in Figure 11.16, any clock manipulation in the DDLL, like input phase selection and MCD locking, impacts the input signal to the APG, and, therefore, changes the loop dynamics of the APG. To ensure smooth transitions and stable operations, the CPPD is blocked from any control voltage changes, and transitions are controlled by the digital loop. The "block" function is also good for test and debug, when an open-loop operation is required to manually adjust the control signal and characterize the *VCDL*.

Finally, a layout presentation of a mixed-mode DLL is shown in Figure 11.26. Notice that the APG is approximately one third as large as the DDLL in this example. The total area is roughly 400 x 320 μm^2.

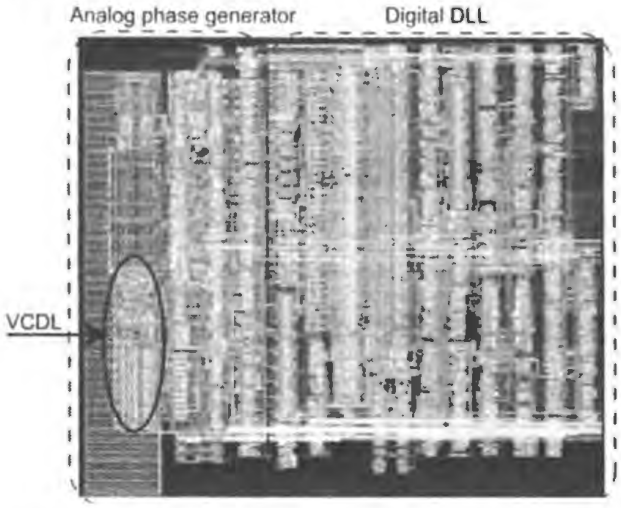


Figure 11.26 Mixed-mode DLL layout example.

11.3.3 Dual-Loop Analog DLL

If only the analog delay line is used for clock synchronization, some restrictions may occur due to a limited tuning range provided by the fixed number of delay stages. From Table 11.2, intrinsic delay variation through the *I/O* path and clock tree can still be several nanoseconds depending on voltage and temperature. A semi-digital, dual-loop DLL is reported in [9], where the coarse adjustment is performed by phase selection, and fine adjustment is achieved by phase interpolation. An analog phase generator (APG) generates multiple phases based on a clock input. Phase selection and interpolation are controlled digitally by a finite stage machine (FSM). The VT tracking is performed by walking through different phases generated by the APG and interpolating between two adjacent phases. A current can be mirrored equally through different legs and selected by a digital code (i.e., thermometer code, 11111000). Although there may still be an uneven phase shift through phase interpolation, especially across the boundary, the perturbation can be small (in the range of 10 ps, for example) compared to other timing variations (i.e., *I/O* model mismatch). The tracking time is comparable to the digital solution, which is proportional to the step size and loop time constant.

If the internal clock speed is slowed down by a clock divider for better clock distribution, the dual-loop structure can still be used with a slave delay line, as shown in Figure 11.27. The analog phase generator (APG) is locked first to provide bias information to the analog DLL. The slave *VCDL*

in the analog DLL has the same structure as the main *VCDL* in the APG. The phase control logic can compare the feedback signal with a reference signal (*CLKIn*) and generate control signals for phase selection and interpolation. Compared to an all-digital implementation, the analog dual-loop DLL has less supply sensitivity due to the CML delay elements. If the global clock distribution can also be implemented with CML buffers, the supply noise sensitivity can be further reduced. Intrinsic delay can also be decreased with CML buffers, which help to reduce loop response time and latency. A similar scheme has also been chosen for high-speed data capturing and per-pin de-skewing for gigahertz links (receiver and transmitter designs) [10].

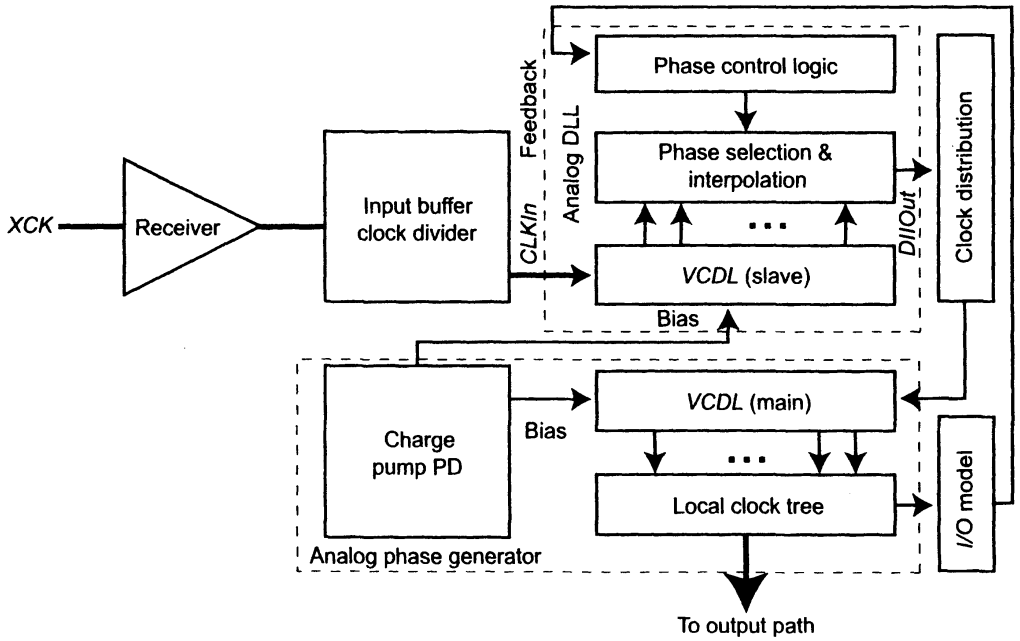


Figure 11.27 Block diagram of a dual-loop analog DLL.

For input timing in high-speed DRAM, the external system clock, or data strobes, are usually center-aligned with command, address (*C/A*), or data, respectively, at input pads. It is critical to match the *C/A* and data paths with the clock/strobe paths for *C/A* and data capturing. Because received clocks must be delivered to different locations for capturing, a clock tree is inevitable and introduces latency for input timing. To match the timing of the clock distribution, dummy buffers must be inserted into each command, address (*C/A*), and data path to maintain the timing relationship. Extra power, area, and delay are added with the matching scheme. Another concern is the Write latency, which defines when the data is available in clock cycles after issuing a Write command.

From the external system and channel, the timing variation (t_{DQSS}) of one quarter of the cycle exists between the system clock and the data strobe.

Internal mismatch between the clock and strobe distribution paths degrades the timing even further. Known relationships between a decoded Write command and captured data and a decoded Read command and output data must be established by clock synchronization circuits. Read-Write latency control circuits can be timed by the respective sync circuits. Clock domain crossing, i.e., command clock to data strobe or command clock to output (Read) clock, must be circumvented for various PVTF corners.

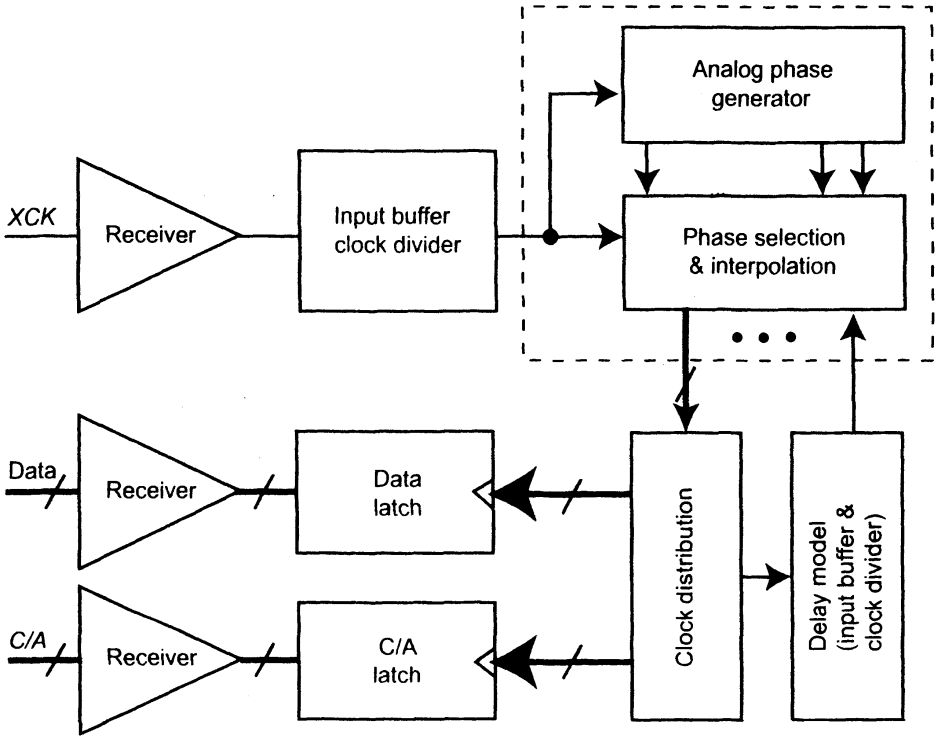


Figure 11.28 Dual-loop DLL for command, address (*C/A*) and data capture.

An analog DLL can be used to hide clock distribution delay and generate quadrature clocks for data capturing, if needed. In Figure 11.28, internal clocks are center-aligned with received data by phase selection and interpolation. *C/A* and data are delivered directly to capture latches after they are received. A training sequence may be required to properly align the data with internally generated clocks. Loop dynamics like tracking, filtering, settling time, etc., must be carefully analyzed. For a feedback system, instantaneous jitter or skew cannot be corrected by a DLL; therefore, jitter caused by the supply noise has the biggest short-term timing impact. Voltage regulation may be helpful for a sensitive timing path. Differential signaling is a plus for better supply noise rejection. As the data rate keeps cranking up, very precise de-skewing for both input and output may be inevitable. A mixed-mode DLL is a good basis for high-speed clock synchronization design.

11.3.4 Mixed-Mode DLL and Its Applications

A comparison between a dual-loop digital DLL (DDLL) and an analog DLL (ADLL) is illustrated in Table 11.4. For high-speed DRAM, an ADLL takes a lead over a DDLL in several categories: lock range, supply noise rejection, duty-cycle correction, and multi-phase generation. Longer lock time and added complexity make it unattractive for mainstream DRAM. A mixed-mode DLL (MDLL) can be configured with a DDLL for clock synchronization and a single-loop analog DLL configured for multiple phase generation. This combination makes multiple phase generation possible when the internal clock is divided down. An MDLL is a good compromise between speed and complexity in digital and analog implementations.

One of the most appropriate applications for an MDLL is high-speed clock distribution. In Figure 11.29, a clock divider slows down the internal clock (*CLKIn*) for propagation through the digital DLL (DDLL) and the main trunk of clock tree (inverters with loads representing routes) for better clock and power delivery. A single low-speed clock (*DllOut*) from the DDLL is distributed over a long distance with less attenuation, power, and mismatch. An analog phase generator (APG) is used for edge recovery and placed close to local clock trees, which make matching much easier. Fewer gain stages (i.e., inverters) are required to drive a low-speed clock over a long distance. The additional APG is justified by reducing the power and area for the main clock distribution and digital DLL.

Table 11.4 A comparison between dual-loop DDLL and ADLL.

	DDLL	ADLL
Lock range	Limited by delay stages	Unlimited
Lock time	Short	Moderate
Dual-loop architecture	Yes	Yes
Resolution	~20 ps	~10 ps
Phase interpolation	Option (digital control)	Yes (digital control)
Layout area	Large	Small
Intrinsic delay	Large	Small
Supply noise rejection	Poor	Good
Duty cycle correction	Extra loop required	Inherent
Multiple phase generation	No	Yes (analog)
Stability	Unconditional stable	1st order, stable
Test and debug	Easy with experience	Difficult
Portability	Good	Moderate
Process scalability	Good	Marginal

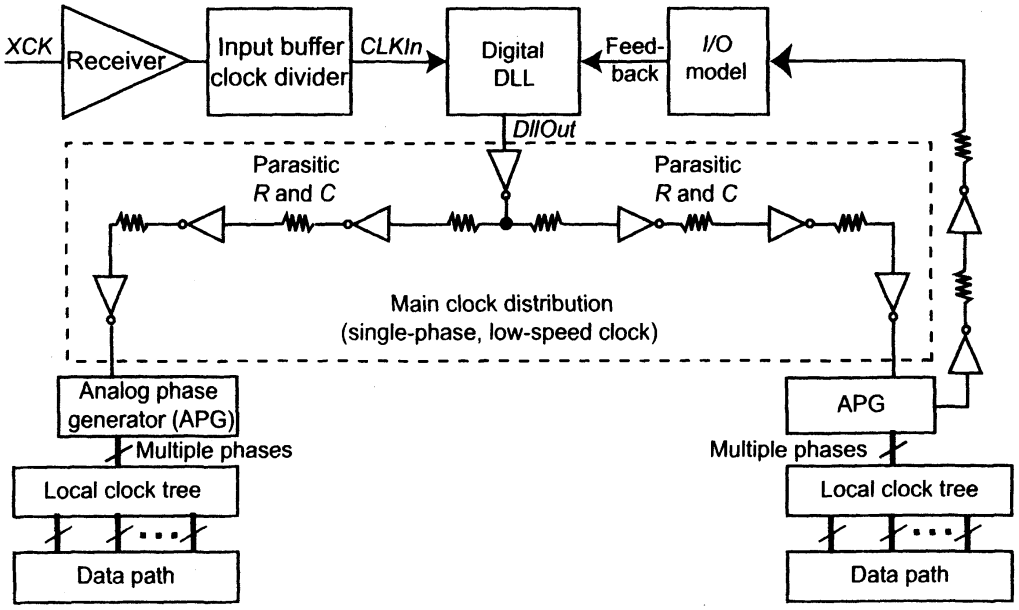


Figure 11.29 A mixed-mode DLL for clock distribution.

Simulation data for a DDLL and an MDLL are compared in Table 11.5 with a typical process, 1.5V supply voltage, and 85°C temperature. The external clock is at 800 MHz. A full-speed clock runs in the digital DLL without an APG, and a half-speed clock with two APGs was used in the mixed-mode DLL. Even with two analog phase generators (APG) in the MDLL clock trunk, the power is less than that of the DDLL with full-speed clocks. For the digital DLL itself, 40% power saving is achieved. However, lock time for the MDLL increases significantly due to analog phase locking.

Table 11.5 Lock time and power consumption for DDLL and MDLL at 1.5 V, TT, 85°C.

	DLL Current (mA)	Clock Trunk (mA)	Local Clock Trees (mA)	Lock Time (ns)
Digital DLL	14	13.3	50	56
Mixed-mode DLL	8.2	13.1	50	146

If the part is down binned to a lower speed grade, full-speed clocking can resume and one of the analog phase generators (APG) can be used as an analog duty-cycle corrector (DCC). Only two phases are generated by the APG: clock and the 180° phase-shifted clock. Compared to a digital DCC (another DDLL, 70% area), an APG is small (30%) and has less intrinsic delay. Better supply noise rejection and quick tracking make an APG a good DCC for clock frequencies anywhere between 400 and 800 MHz.

Finally, a 90° phase-shifted data strobe can be generated by the MDLL. The strobe can be center aligned with the output data rather than edge aligned, which makes it easy to capture data at the controller site. The APG should provide a 45° phase shift for a divide-by-two clock or a 90° phase shift for a full-speed clock. If an MDLL is also used for input timing, an internal 90° phase-shifted clock can be generated for data capture.

11.4 WHAT'S NEXT FOR TIMING

In this chapter, we explored different timing circuits, from a simple digital DLL to a more complicated mixed-mode DLL. We also investigated various design issues associated with different architectures and presented examples of circuit implementations. Based on the application and *I/O* specs, good circuit design can reduce timing uncertainty, minimize clock skew, and improve *I/O* performance. However, timing circuits also add complexity to *I/O* design, increase power, grow layout area, and lengthen debug time.

Moving forward, as DRAM processes scale down into the deep submicron regime, designing high-speed, low-power *I/O* interfaces becomes a

huge challenge. High-performance phase-locked loops (PLLs) may replace DLLs as alternative timing solutions. While clock data recovery (CDR) using a PLL provides lower pin counts and strobe-less *I/O* timing, it puts more constraints on data patterning. Slower transistors and a limited number of metal layers make the design even harder. Mixed-signal design might be a key for the future *I/O* circuitry, in which analog core (i.e., voltage-controlled oscillator, receiver, and transmitter) and digital support circuits (i.e., calibration, interpolation, and tuning) co-exist. Circuit partitions, power bussing, and verification are important for achieving better performance.

On the other hand, extensive training and calibration from the memory controller may be necessary to offload some of the burden of the timing circuit design from memory devices. At the same time, minimizing on-die timing variations is critical for this scheme to work. Clock distribution, clock domain crossings, and latency control remain important issues for high-performance memory data path design. In Chapter 12 we try to address some of these issues via clever logic design.

An advanced memory buffer (AMB) used in a fully-buffered, dual inline memory module (FB-DIMM) provides another possibility in that timing circuits and high-speed *I/O* are moved into the AMB. The AMB, fabricated with logic processes, serves as an intermediate buffer between the memory controller and DRAM devices. An AMB supports a narrow point-to-point channel enabling high-speed communication between the host controller and the AMB while also supporting high bandwidth, wide channels between the AMB and multiple memory devices. Obviously complication is the price paid for higher performance.

REFERENCES

[1] A. Hatakeyama, H. Mochizuki, T. Aikawa, M. Takita, Y. Ishii, H. Tsuboi, S. Fujioka, S. Yamaguchi, M. Koga, Y. Serizawa, K. Nishimura, K. Kawabata, Y. Okajima, M. Kawano, H. Kojima, K. Mizutani, T. Anezaki, M. Hasegawa, and M. Taguchi, "A 256-Mb SDRAM using a register-controlled digital DLL," *IEEE Journal of Solid-State Circuits*, vol. 32, pp. 1728–1732, November 1997.

[2] F. Lin, J. Miller, A. Schoenfeld, M. Ma, and R. J. Baker, "A register-controlled symmetrical DLL for double-data-rate DRAM," *IEEE Journal of Solid-State Circuits*, vol. 34, pp. 565–568, April 1999.

[3] J. Maneatis, "Low-jitter process-independent DLL and PLL based on self-biased techniques," *IEEE Journal of Solid-State Circuits*, vol. 31, November 1996.

[4] A. Efendovich, Y. Afek, C. Sella, and Z. Bikowsky, "Multifrequency zero-jitter delay-locked loop," *IEEE Journal of Solid-State Circuits*, vol. 29, pp. 67–70, January 1994.

[5] T. Saeki, Y. Nakaoka, M. Fujita, A. Tanaka, K. Nagata, K. Sakakibara, T. Matano, Y. Hoshino, K. Miyano, S. Isa, S. Nakazawa, E. Kakehashi, J. M. Drynan, M. Komuro, T. Fukase, H. Iwasaki, M. Takenaka, J. Sekine, M. Igeta, N. Nakanishi, T. Itani, K. Yoshida, H. Yoshino, S. Hashimoto, T. Yoshii, M. Ichinose, T. Imura, M. Uziie, S. Kikuchi, K. Koyama, Y. Fukuzo, and T. Okuda, "A 2.5-ns clock access, 250-MHz, 256-Mb SDRAM with synchronous mirror delay," *IEEE Journal of Solid-State Circuits*, vol. 31, pp. 1656–1665, November 1996.

[6] K. Wong, E. Fayneh, E. Knoll, R. Law, C. Lim, R. Parker, F. Wang, and C. Zhao, "Cascaded PLL design for a 90nm CMOS high-performance microprocessor," *ISSCC Digest of Technical Papers,* pp. 422–424, February 2003.

[7] F. Lin, "Research and design of low jitter, wide locking-range all-digital phase-locked and delay-locked loops," dissertation, March 2000, University of Idaho.

[8] R. J. Baker, *CMOS Circuit Design, Layout and Simulation*, 2nd ed., Hoboken, NJ: IEEE Press, 2005.

[9] S. Sidiropoulos and M. A. Horowitz, "A semi-digital dual delay-locked loop," *IEEE Journal of Solid-State Circuits*, vol. 32, pp. 1683–1692, November 1997.

[10] E. Yeung and M. A. Horowitz, "A 2.4 Gb/s/pin simultaneous bidirectional parallel link with per-pin skew compensation," *IEEE Journal of Solid-State Circuits*, vol. 35, November 2000.

[11] B. Razavi, *Monolithic Phase-locked Loops and Clock Recovery Circuits: Theory and Design*. Piscataway, NJ: IEEE Press, 1996.

[12] B. Garlepp, K. S. Donnelly, K. Jun, P. S. Chau, J. L. Zerbe, C. Huang, C. V.

Chapter
12

Control Logic Design

12.1 INTRODUCTION

The control logic design for a DRAM could be considered the most neglected area of DRAM design in the literature. One obvious reason for such a lack of attention is that there are probably as many ways to implement the DRAM control logic as there are logic designers working on the problem. We define the peripheral logic interface as the set of circuits that interfaces the array access signaling protocol given in the data sheet with the peripheral array logic described in Chapter 4. This logic must translate command and address information coming from the input pins of the DRAM device and provide control signals to the array peripheral logic. We consider the peripheral logic interface to be the hub for controlling interactions between the data sheet specifications and three distinct areas of the DRAM die covered in previous chapters: the input circuit path, the output circuit path, and the array peripheral logic. Figure 12.1 illustrates the relationship between each of these three areas of the DRAM highlighting that the peripheral logic interface acts as the liaison between the peripheral array logic and the DRAM *I/O*.

An important note about the peripheral logic interface is that the row control is minimal out of this area. Wordline accesses or, what is being referred to here as row accesses, are considered more a part of the peripheral array logic since this type of control does not depend on the *I/O* circuit path logic for performance and generally does not require the same sort of treatment that data accesses require for a high-performance part. For this reason, we do not consider the wordline control, also commonly known as the *RAS chain* circuitry in this chapter. Please refer to Chapter 2 for more details about row access control and timing circuitry.

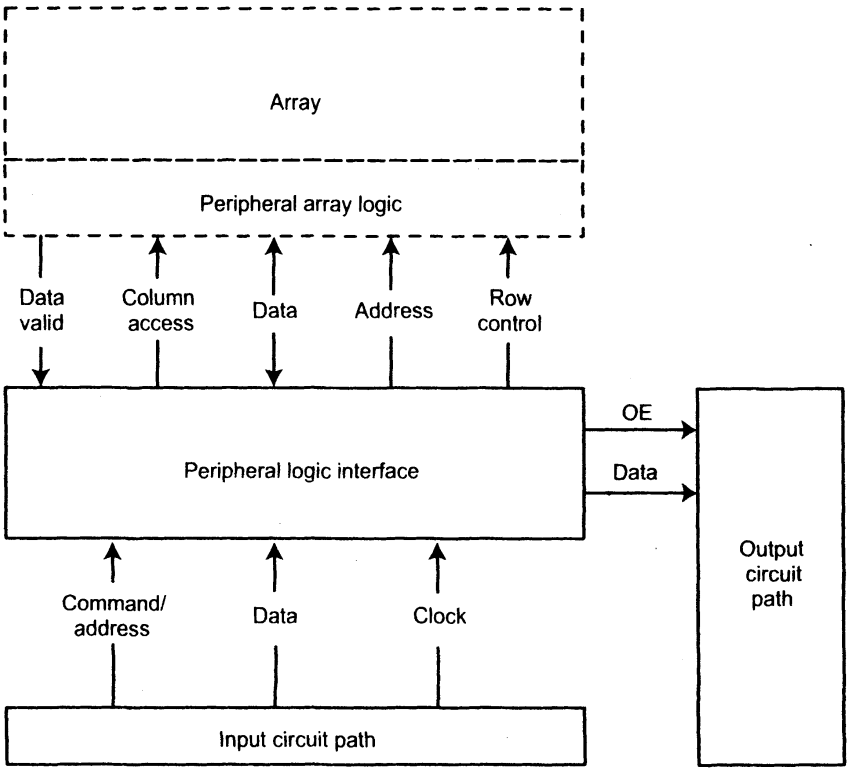


Figure 12.1 DRAM logic partitions.

With generational changes to the DRAM specifications come changes in the input and output circuit paths. The *I/O* circuit paths must be tailored to higher data rates and sometimes to changes in the command, data, and address protocols. This is contrasted with the relative consistency in array access signaling requirements for interfacing with the array peripheral logic as discussed in Chapter 8. For commodity DRAM, i.e., DDR, DDR2, etc., the column cycle times shown in Table 8.1 (repeated in Table 12.1) have remained relatively constant through several generations. Any benefits from process scaling are counter balanced by increases in memory density, which results in a relatively constant die size. As process scales, voltage scaling and interconnect delay become the overriding factors limiting array performance. For the GDDR, high-performance devices listed at the bottom of the table, column cycle time is reduced because of array and package ballout differences, which are outlined in Chapter 8, resulting in higher bandwidth (8.1). The control logic circuits bridge the gap between the bandwidth requirements of the synchronous *I/O* and the asynchronous timing of the array while, simultaneously, maintaining access delay independent of variations in clock period. As high-performance DRAM requirements demand reduced column cycle times, the challenge for the peripheral logic interface design is to be transparent to frequency changes. The goal is to ensure that

array access delay characteristics of the DRAM always limit device performance.

Table 12.1 DRAM technology and speed relationships.

Technology	Burst Length	Min Column Cycle Time (t_{CCD})	Peak Bandwidth
SDRAM	1	6 ns	167Mbps/pin
DDR	2	5 ns	400Mbps/pin
DDR2	4	5 ns	800Mbps/pin
DDR3	8	5 ns	1.6Gbps/pin
GDDR3	4	2.5 ns	1.6Gbps/pin
GDDR4	8	3.2 ns	2.5Gbps/pin

Notice in Table 12.1 that burst length increases as bandwidth increases resulting in a constant column cycle time. With more data delivered on each column access, we can increase the *I/O* bandwidth without major changes to the operation or architecture of the DRAM array. The amount of data transferred for each column access, expressed relative to each data *I/O*, is often referred to as *data Prefetch*. For example, a device specified to deliver a burst of eight data bits for each column access is said to have an 8n Prefetch. If the same device has a data bus width of 32 bits and an 8n Prefetch, then each column access reads or writes 256 bits of data. This concept is further illustrated by Equation (8.1).

To understand how DRAM logic design differs from more traditional logic design, we examine logic style choices that may aid performance and reliability while working within the confines of an inexpensive DRAM process. In the remainder of this chapter we discuss performance considerations and design choices for the logical interfaces shown in Figure 12.1. We first look at command and address capture and processing and how choices in logic implementation can affect DRAM performance. We then examine Write data capture and demultiplexing and how Write data protocols for current high-performance graphics memory complicate logic and timing circuit implementation. Finally, we discuss the Read data path and how higher bandwidth demands have resulted in increased complexity for the Read data pipeline and serialization circuits.

12.2 DRAM LOGIC STYLES

Before we consider details of the control and data path logic design, we must understand some of the difficulties that the DRAM designer faces when making decisions regarding logic style and circuit topologies. Making a choice in logic style primarily becomes choosing between using static logic gates with clocked flip-flops or adopting a dynamic or domino logic style employing latches and multi-phase clocks to gain clock skew tolerance. The use of dynamic logic and latches implies a departure from a more common logic design flow involving circuit synthesis tools using hardware description languages such as Verilog® or VHDL. As we progress through this section, we see that the choice of logic methodology is driven by three fundamental limitations:

1. Low-cost process technology motivated by the commodity status of most varieties of DRAM products.
2. Delay variation for array column operations across process, voltage and temperature corners.
3. Performance and protocol requirements defined by industry standards and published in product data sheets.

These limitations tend to push us to a more custom logic design flow rather than a more automated, abstracted approach. The simplicity of DRAM operation makes gate count manageable for a full-custom flow. However, this simplicity in operation is complicated by the protocol for interfacing to the DRAM from the outside world and complicates timing interactions between global circuit blocks on the DRAM die. Even though synchronous-protocol interfaces dominate current DRAM technology, the operation of the DRAM array has not changed, and the DRAM remains, fundamentally, an asynchronous device.

12.2.1 Process Limitations

Design process technology used for DRAM has historically served as an industry-wide development vehicle for process feature scaling. Due to the commodity nature of the DRAM business, the research and development costs related to process scaling have paid dividends because of increased device yields per unit area of processed silicon. The process technology lead DRAM once enjoyed has eroded in recent years relative to process development for processors and logic process nodes developed by contract fabrication companies. The rush to develop finer process geometries is somewhat tempered by the changes to MOS devices and interconnect brought on by process scaling [3]. Some of these changes can have adverse effects on DRAM operation, which can lead to yield loss.

Logic design for a monolithic DRAM is made difficult because of low-cost process requirements. Some of the "exotic" materials and process steps often used by microprocessor manufacturers are economically out of reach for the DRAM logic designer. To maintain a reasonable cost profile, the number of mask steps must be kept to a minimum.

There are two primary barriers to maximum logic circuit performance as a result of the DRAM process and process scaling. First, high transistor threshold voltage (V_{TH}) is necessary to minimize DRAM leakage current. This is an intentional process characteristic for maximizing data retention in the array. This process characteristic bleeds over to the logic areas of the device and limits performance of the logic gates. Because of the increased V_{TH}s, gate delay increases, which is illustrated by the alpha power law model [4] where the gate delay is partially expressed by:

$$t_d \propto \frac{C_L V_{DD}}{(V_{DD} - V_{TH})^{\alpha}} \tag{12.1}$$

where C_L is the load capacitance, V_{DD} is the supply voltage, and α is the velocity saturation index, which is close to one for a short-channel process.

In some cases, the problem of high V_{TH}s has been overcome by developing process technology with multiple gate oxide thicknesses and doping profiles. By providing an option for thinner gate oxide devices in the DRAM process, performance improvements have been realized for *I/O* circuits and peripheral logic interface circuits. Even with process improvements, more aggressive logic styles, such as dynamic logic may be necessary to overcome the process delay limitations caused by relatively high V_{TH} devices. Thinning gate oxides and more aggressive logic styles have helped opened the door to high-performance DRAM devices. But device scaling has not lead to the performance gains seen with process scaling of logic processes for modern microprocessors [6].

Because of process scaling, changes in interconnect characteristics is a second major difficulty faced by the DRAM logic designer. Figure 12.2 shows how scaling has resulted in a change in the aspect ratio of the interconnect metals on a single layer. Qualitatively, we must consider that the capacitance is proportional to the dielectric constant, κ, wire height, H and space, S. Using low-k dielectrics to reduce sidewall capacitance is sometimes an option for more expensive logic processes, but this has not been a cost-effective option for DRAM at this time. Please refer to [7] and [8] for a quantitative treatment of this topic. Wire scaling results in increased capacitive coupling, which in turn increases the potential for noise. Under worst-

case conditions, when neighboring interconnect transitions occur in opposite directions, the capacitance is close to double the statically calculated capacitance. For example, the transitions on signals *A1* and *A2* in Figure 12.2 "attack" the "victim," *V*. This results in an increase in capacitive coupling current and causes a longer delay on signal *V*. This scenario must be considered when determining worst-case delay on a signal bus. The designer is faced with a trade-off when designing signal busses. One example would be the design of the peripheral data bus. Because a high-performance DRAM must have larger data Prefetch depths for each data *I/O*, the designer is tempted to take advantage of the increased wire density to keep die size low and still provide the necessary data to maintain *I/O* throughput. But because of capacitive coupling and loading, much of the potential interconnect density is used for increased space and/or static shields to help mitigate the effects scaling has on interconnect [23].

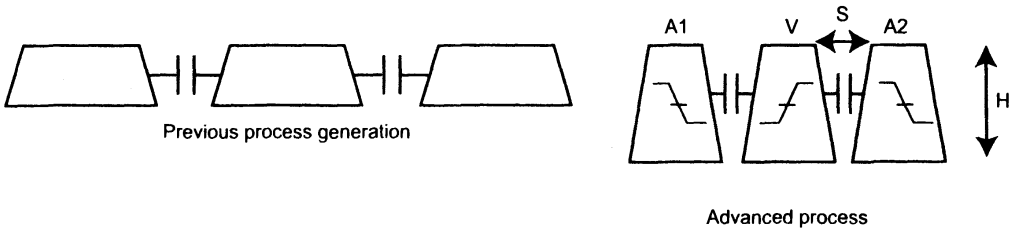


Figure 12.2 Effects of scaling on interconnect capacitance.

$$C \propto \kappa \frac{H}{S} \tag{12.2}$$

There are other more indirect effects of scaling that impact logic circuit design. When process scales, voltage is often scaled proportionally. But the V_{THS} of the DRAM process have not changed proportionally with scaling and has therefore lead to a higher ratio of threshold voltage to supply voltage. Referring to Equation (12.1), when V_{DD} is reduced there is an increase in delay, t_D. Voltage scaling can affect noise immunity for dynamic circuits; but this vulnerability is often offset by the relatively high V_{TH}s of the DRAM process. We see that characteristics of a DRAM process are a hindrance to high-performance logic. But we must also consider how DRAM performance specifications are dictated by the access delays of the DRAM array. Next, we will consider specifications for array operation and how these specifications affect the circuit topology decisions for DRAM logic design.

12.2.2 Array Operation

Chapter 1 presents some of the generational changes to the command and address signaling protocol that has occurred over the life of the commodity DRAM. Early DRAM protocol called for a broadside address bus, where both row and column addresses are simultaneously applied for each access. Second-generation DRAMs introduced the multiplexed address bus, where the *RAS** and *CAS** signals latch the row and column address from the same address signal inputs, respectively. Other developments that significantly affect the design of the peripheral logic interface include the idea of segmenting the memory arrays into banks and developments of alternative modes of operation. These alternative modes of operation are primarily focused on data Prefetch modes for column access commands.

The synchronous interface was the most significant development for third-generation DRAM. This was the beginning of what could be called "high-performance" DRAM. The addition of a synchronous interface increases the complexity of the peripheral logic, but simplifies DRAM protocol by allowing some of the timing parameters to be expressed with reference to a global clock. Now, instead of a combination of signals validated by the CS* signal, we have a continuous clock signal sequencing command and address inputs.

The point to take away from this discussion is that throughout the evolution of the DRAM interface protocol, the fundamental operation and circuit topology of the DRAM array has changed very little. Access delay for opening a wordline or for column data accesses are very dependent on process, voltage, and temperature (PVT) conditions. This point is illustrated in Figure 12.3 where we see Read cycle timing diagrams for both an extended data out (EDO) DRAM and a SDRAM. Notice the timing parameter for t_{CAC} in the EDO DRAM data sheet relative to the value $t_{CL} = 2*t_{CK}$ in the SDRAM data sheet (t_{CK} is the clock period). The EDO data sheet expresses a maximum value for $t_{CAC} = 13$ns, which is the data access delay measured from the falling edge of *CAS**. The parameter t_{CAC} is expressed as a maximum because valid data may be output from the device any time before t_{CAC}, depending on operational PVT. We also see that the *CAS** signal looks similar to a clock: it toggles for each data access that is driven from the device for the EDO-Page-Mode operation. However, the EDO DRAM is not a pipelined device. That is why the cycle time of the *CAS** signal, $t_{PC} = 25$ns, is greater than the maximum access time, $t_{CAC} = 13$ns, for each Read access. This guarantees that successive Column accesses cannot occur before all data is driven from any previous Read.

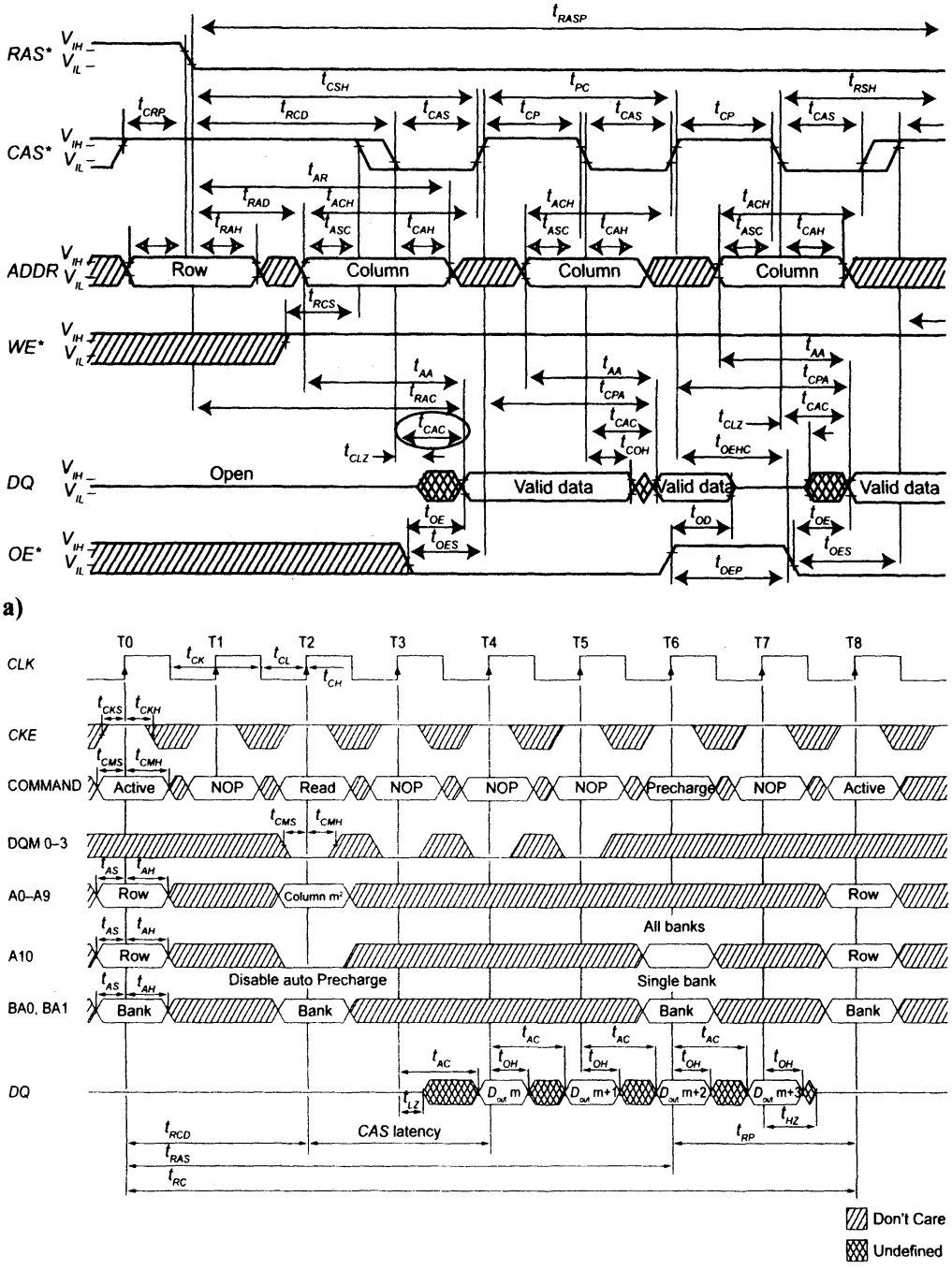


Figure 12.3 a) EDO Page mode and b) SDRAM Read cycles.

Early-generation SDRAM had limited pipelining and often operated at clock cycle times that matched the requirements of the underlying asynchronous DRAM. For instance, let's look at the SDRAM side of Figure 12.3. If a device is specified to operate at a 133 MHz clock frequency with a *CAS** latency (t_{CL}) of 2 and a data access time (t_{AC}) of 5.4ns, then the total

maximum access time for the device is 12.9ns. This is expressed as a maximum because the first valid data out of the device in the SDRAM can still come any time before the maximum specified access time. The corresponding timing parameter, t_{CAC}=13ns, for the EDO device reflects approximately the same access delay as is shown for the SDRAM device.

When we look at a column Read operation for a high-performance DRAM, such as the data sheet for a GDDR3 device shown in Figure 12.4, we see that column access latency has remained relatively constant over the generational changes of the DRAM protocol. The clock period shown in Figure 12.4 is 1.25ns; and, with a *CAS* latency of 10, the maximum column access time is 12.5 ns. The lack of improved access time is partially because the transistor speed benefits of process scaling are often tempered by voltage scaling and increased interconnect coupling. The sum of the effects of interconnect scaling with simultaneous increases in area because of increasing array density has caused most DRAM timing parameters to remain constant.

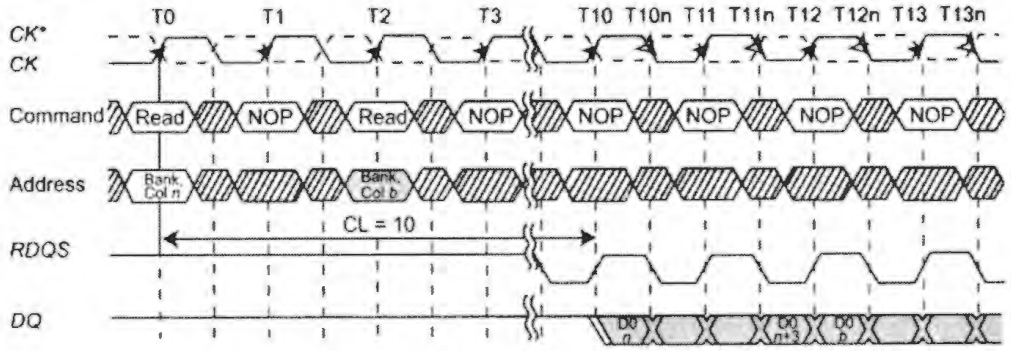


Figure 12.4 GDDR3 Read burst timing diagram.

Another observation from Figure 12.4 is that the short burst length (BL), which is translated into column cycle time (t_{CCD}) by Equation (12.3) for a double-data rate (DDR) device,

(12.3)

$$t_{CCD} = \frac{BL}{2} t_{CK}$$

allows several Read commands to accumulate before the first data burst is output from the device. In Figure 12.4, with a *CAS* latency of 10 and a Read command occurring every second clock cycle, the result is 5 possible Read commands pipelined in the device before the first Read command completes. This is in contrast to early SDRAM designs where each data access stood alone because the C/A clock frequency was chosen to match the characteristic delays of the memory device. Later in this chapter we discuss how

large delay variations for array column accesses due to PVT variation can influence the logic design, especially in the Read data path.

12.2.3 Performance Requirements

In the previous section we saw that the characteristic access delays of the DRAM have remained relatively constant even though data rate requirements have changed drastically between Page-Mode DRAM and high-performance DRAM. The data rate requirements for high-performance DRAM have meant significant changes for the peripheral logic interface as well as the data *I/O* circuits and analog signaling protocols as compared to earlier generations of DRAM. To help achieve these higher data rate requirements, some improvements in peripheral logic transistors have occurred (Section 12.2.1).

One area where transistor performance significantly improves overall performance is in the *I/O* circuitry. This is especially true for the receiver logic and the Read data serialization logic. Also, the interface between the parallel data bus from the array and the serialization logic greatly benefits from transistor performance improvements. Later in this chapter, we examine how Read data is presented to the data serialization logic introduced in Chapter 10. Transistor performance improvements alone, however, will not enable high-performance DRAM operation in the peripheral logic interface circuits. When implementing designs with DRAM process technology, we must also consider logic circuit style and topology as well as greater parallelism to complete the transition to higher clock frequencies.

Parallelism itself is a natural consequence of clock frequency division. Clock frequency division, discussed in Chapter 11, also creates multiple clock phases, which leads to an opportunity for parallel logic paths. For example, if we are receiving commands in the peripheral logic interface with a clock running at 1.25 GHz, and we divide the internally distributed clock frequency by two, we now have a clock frequency of 612.5 MHz. To maintain the correct command bandwidth established by the external clock frequency of 1.25 GHz, we must generate a two-phase clock running at 612.5 MHz; with the clock phases shifted a relative 180 degrees. Such a situation is shown in Figure 12.5. Now we can generate parallel logic paths based on the two-phase clock signal but at the expense of complications in the timing circuits and increased gate counts. However, with the process limitations just outlined and the advantages of clock frequency division discussed in Chapter 11, the complications in both logic implementation and silicon area prove to be worthwhile compromises for achieving high performance.

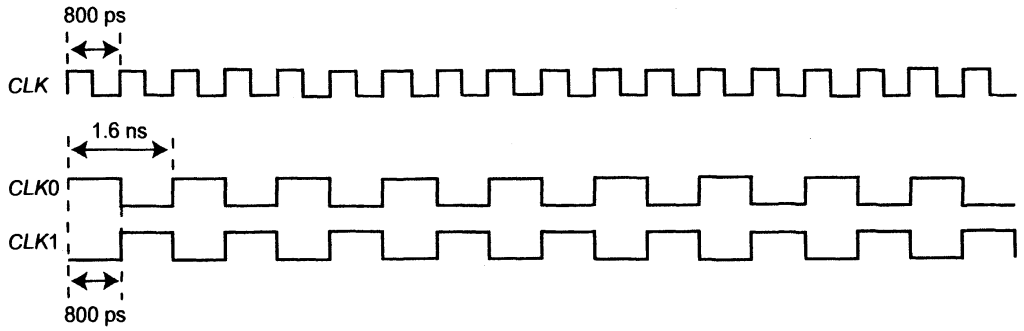


Figure 12.5 Clock division with resulting multi-phase *CLK*.

When we look at clock frequency specifications for high-performance DRAM, we find that there is often a demand for a wide range of clock and data frequencies. One reason for this is that it is more cost effective for a single device to cover a range of power-performance points. Of course, this has an impact on the logic design. For instance, Table 12.2 and Table 12.3 show the clock frequency specifications for a combination GDDR3/ GDDR4 device.

Table 12.2 GDDR3 frequency and latency range.

Speed Grade	Clock Rate			
	CL - 10	CL = 9	CL = 9	CL = 8
–12	800 MHz			
–14		700 Hz		
–15			650 MHz	
–16				600 MHz

Table 12.3 GDDR4 frequency and latency range.

Speed Grade	Clock Rate				
	CL = 14	CL = 13	CL = 12	CL = 11	CL = 10
–08	1.25 GHz				
–09		1.1 GHz			
–1			1.0 GHz		
–12					800 MHz

To get the best performance over a wide range of operating frequencies, we must consider how the peripheral logic interface operates relative to the clock frequency. Along with covering a wide range of operating frequencies, we must also consider that the logic is robust enough that the device can be speed graded or *binned* according to the access delays of the array. What this means is that the logic must not limit the performance of the array access delays with any dependencies on clock period. In the next two sections, we briefly examine how logic style can influence the performance of the DRAM when viewed at the *I/O* pins of the device. In the remainder of the chapter, as we discuss more specifically how the peripheral logic interface is implemented, we examine how the chosen logic style must bridge the gap between the synchronous environment in which the DRAM operates and the asynchronous DRAM array.

12.2.4 Delay-Chain Logic Style

One style of logic that has been a natural fit for the asynchronous nature of the DRAM array, and is not widely accepted outside of DRAM design, is a style we refer to here as *delay-chain* logic. This logic style is comprised of a mixture of combinatorial gates, one-shots (monostable multivibrators), and latches. The reason we define this style of logic as "delay-chain" is because the logic is usually implemented with a series of delays such that a sequence of functions or events can be executed without a global timing reference (clock). In other words, the logic consists of a cascade of events and functions. We will look at an example of how this style is applied as part of a column access command to the array.

Figure 12.6 is an example of delay-chain logic used for enabling the column decoder and helper flip-flop logic (HFF, Section 8.2) circuits for a column access. At the input to the circuit is a signal named *colCyc**. This signal starts the delay chain and initiates the sequence of events necessary to enable the column decoders and access data from the array. There are several more delay chains cascaded with this circuit that perform other sequential operations. We are not going into the details about this circuit's operation as it is only presented here as a simplified example of a delay-chain logic circuit. However, we will examine the implementation of the one-shot that follows *colCyc**. As a side note, keep in mind that an array access for both Reads and Writes is identical. Think of the array as a Read-only device. A Write occurs when a Read access is overdriven by Write driver circuits, through the *I/O* transistors (Section 1.2) and onto the datalines where the sense amplifier restores a full-rail state to the cell.

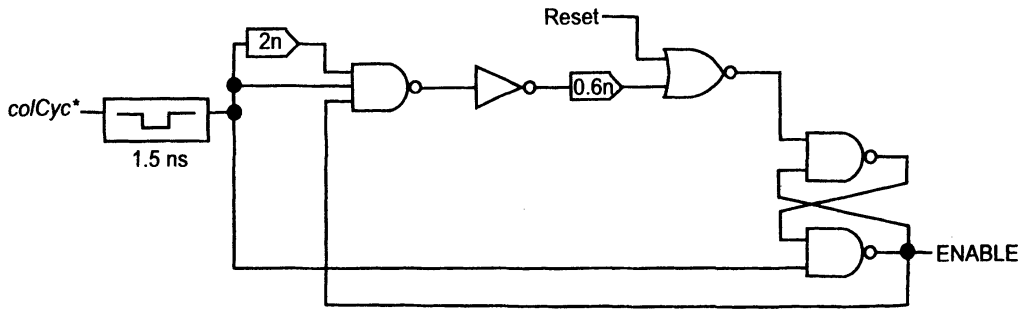


Figure 12.6 Delay-chain logic example.

We see in Figure 12.6, that when *colCyc** fires LOW, a one-shot of 1.5 ns duration is triggered. The design of this one-shot circuit is shown in Figure 12.7. The circuit has an equilibrium state where input A is HIGH with node C LOW, which forces node B HIGH. Looking at Figure 12.7, whenever input signal A transitions LOW, node C transitions HIGH, which causes output Y to fire LOW. The delay cell labeled "1n" in Figure 12.7 only delays rising edge inputs. Such a delay element is implemented through capacitive loading in a string of inverters. After the rising edge from gate X0 is delayed and inverted through the inverted delay path, a LOW appears on node B, which forces Y back HIGH completing the approximately 1.5 ns pulse.

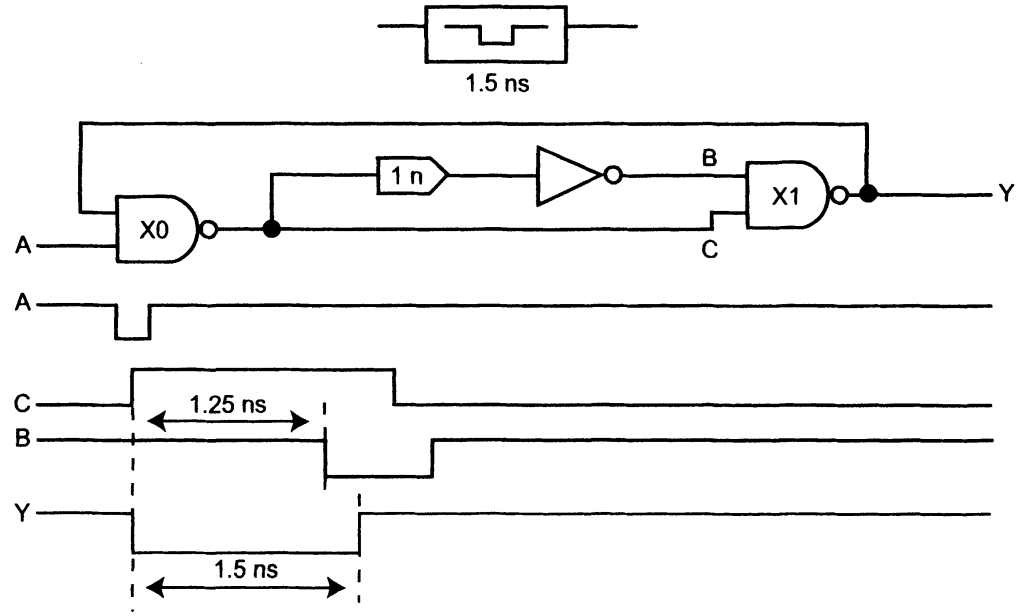


Figure 12.7 One-shot design.

When we consider PVT variation, the pulse width out of the circuit in Figure 12.7 can vary by 100% over the full operating PVT range. How does the delay-chain logic style fit in with high-performance DRAMs? The

answer is that this logic style is still widely used in the peripheral array logic region. As we stated at the beginning of this chapter, when synchronous DRAM came into existence, the synchronous interface became a wrapper around the existing DRAM logic. Thus, we have defined the peripheral logic interface as the high-performance, synchronous section of logic that interfaces to the original DRAM peripheral logic. Many DRAM designs still use some form of delay-chain logic in the peripheral interface region of the DRAM die. This method can prove effective when directly interfacing to the array, but as performance demands grow and process continues to scale, this style of logic has limitations that must be considered.

Some risks when using delay-chain logic in the peripheral logic interface region include questionable delay tracking reliability when operating over a wide range of clock frequencies as well as increased process variability as a result of scaling. The latter results in reliability issues related to statistical delay variation relative to anticipated delay values. Another important consideration when using delay-chain logic is the fact that the heuristic nature of the design approach for this style of logic often leads to disjointed circuitry with a lack of procedural clarity for the design process flow. What often happens is that delay-chain logic designs are passed between different designers and the logic is patched to the point that the original functionality is difficult to discern. When delay-chain logic is used in the peripheral logic interface area, the circuits must be "tuned" to operate for a given process, voltage, and frequency. Being forced to tune these circuits limits the reusability of this style of logic. This is not to say that this style of logic is not without merit when applied judiciously in the peripheral array logic region. The greatest strength of using this logic in the array peripheral region is that the delay chains exhibit some correlation to PVT variation between the array control signals and the delay characteristics of the array itself. Another advantage to this style of logic is the ability to adjust the delay stages using metal mask changes so that post-silicon timing adjustments can be easily made.

Next, we will consider a dynamic logic methodology. This method solves some of the problems of delay variation and is well suited to the asynchronous timing specifications of DRAM array accesses. Two more important advantages for applying dynamic logic in the peripheral logic interface is that first, this method provides a rigorous design procedure for implementing logic functions and second, the logic implementation is only limited by gate delay and is robust when properly designed.

12.2.5 Domino Logic

Early DRAMs used NMOS logic for implementing protocol operations, address decoding, and even output pad drivers. P-channel devices have long been practical for the implementation of full CMOS gates, yet p-channel devices still pose a limitation to logic circuit performance, especially for a DRAM process where cost concerns limit process improvements. Another way to overcome limitations of the p-channel device is by using the domino logic style [10]. Domino logic is interesting in that each gate requires a clock signal, yet logic functions can be designed that are not necessarily synchronous to a global clock. Instead, logic functions can be implemented that result from a cascade of events, where each gate output has the potential of generating a clocking function for subsequent gates in a chain of logic. This description may sound similar to the description of the delay-chain logic style where logic functions are cascaded and separated by chains of delays that ensure events follow a known sequence. This concept can be extended to the use of domino logic for implementing many of the necessary logic functions in the peripheral logic interface. In this section, we examine some of the advantages of domino logic and how the application of this technique is well suited to the synchronous interface of the DRAM and the preservation of asynchronous array performance.

Figure 12.8 illustrates the operation of a *footed* domino gate that implements an AND function. The term "footed" refers to the transistor MN2 in the n-channel stack. This transistor guarantees that the current path through the n-channel stack is cut off during evaluation. When the clock signal, *CLK*, is LOW, the node DYN is precharged to V_{CC} through the p-channel device, MP0. This state is referred to as the precharge phase of operation. When *CLK* transitions HIGH, the state of A and B is evaluated, which is referred to as the *evaluation* phase. If both A and B are HIGH, DYN is discharged to ground through MN0–MN2, forcing the output of the inverter HIGH. The device MP1 is used as a keeper device to counter device leakage when the *CLK* is HIGH and DYN is not discharged. MP1 is sized with a long channel to reduce crowbar current and performance degradation during the evaluation phase.

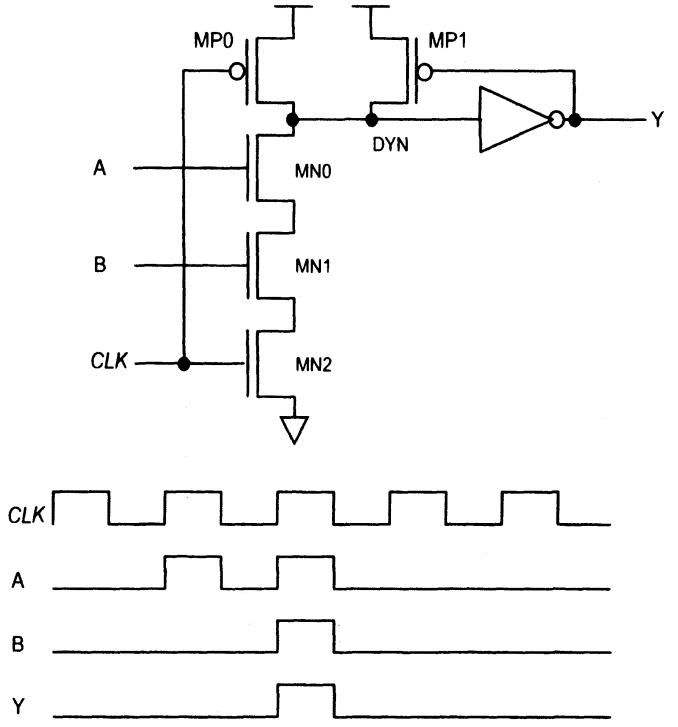


Figure 12.8 Domino AND function.

Many designers are unfamiliar with the application of domino logic because most logic design tools and methodologies focus on synchronous implementations using hardware description languages, such as VHDL. This level of abstraction is not as prevalent in DRAM logic design, so the designer has broader latitude when choosing logic styles. However, this freedom comes with a caveat. For instance, we must be aware of some of the risks involved when implementing domino-style logic to achieve a robust implementation. These risks include interconnect capacitive coupling, leakage, charge sharing, noise tolerance, minority carrier charge injection, and other issues related to reliability. These topics, though very important, are not covered in this work. A discussion of these issues can be found in [11] and [12]. The reader would be well served in researching these issues before forging ahead with a domino logic design. We will instead focus on some of the advantages for using domino logic when interfacing the synchronous DRAM interface with the asynchronous DRAM array.

Perhaps the greatest advantage of using domino-style logic in the peripheral logic interface is the monotonic signal transitions for each term of a logic function. We define a monotonic signal, with regard to the domino-style logic, as a signal that can only transition from LOW to HIGH during the evaluation phase. The static gate (the inverter in Figure 12.8) that forms the output stage of the domino gate enforces this regiment by invert-

ing the output of the dynamic gate and driving a LOW-to-HIGH transition to the input of the next domino gate. Monotonicity must be enforced between levels of logic for reliable timing and robust operation.

One problem we face in the DRAM interface is that signals coming from the input circuit path are not monotonic, as we have defined here. Signals such as *CAS** and *RAS** transition at the command/address clock (*C/A* clock) frequency and can transition both HIGH or LOW between each evaluation phase of the clock. If we are to use domino-style logic in the peripheral logic interface, we must first convert the control signals, such as *CAS** and *RAS**, from a static format referenced to the C/A clock to monotonic signals compatible with the domino logic style.

Figure 12.9 shows a circuit that converts signals from the static logic domain to the dynamic logic domain [26]. If we look at the input latch circuit examples from Chapter 9, we see that the output from these circuits is generally valid for an entire clock cycle. The circuit in Figure 12.9 can be used to convert from the full-cycle static domain into the monotonic, half-cycle valid domain. This circuit operates by creating a pulse from the clock edge through the inverter string driven by the clock signal. When *CLK* is LOW, the outputs *Q* and *Qb* are precharged to a low value through the p-channel devices. When *CLK* transitions HIGH, the data input is evaluated during the pulse created by the overlap of the HIGH time of the clock signal and the delayed transition of the clock through the inverter chain. An important aspect of the static-to-dynamic converter circuit is that the output is considered *dual-rail*. What this means is that if the input data to the converter, D, is LOW, then the output signal *Qb* will transition HIGH for one half of a clock cycle. Likewise, if the input *D* is HIGH during evaluation, then the output *Q* will transition HIGH while the output *Qb* remains in the Precharge state (LOW). Domino circuits cannot be cascaded to perform combinatorial logic because the outputs of domino gates are not functionally complete. Static hazards can occur where, for instance, a non-monotonic transition from HIGH to LOW could be skewed relative to the Precharge clock such that the gate evaluates to the incorrect state. Once the dynamic logic gate incorrectly evaluates, it cannot recover until the next evaluation cycle. The dual-rail output from the circuit in Figure 12.9 generates both true and complementary outputs that are both monotonically rising. This allows us to exploit the skew-tolerance [1] of domino logic and secure functional completeness for implementing the command decoder logic.

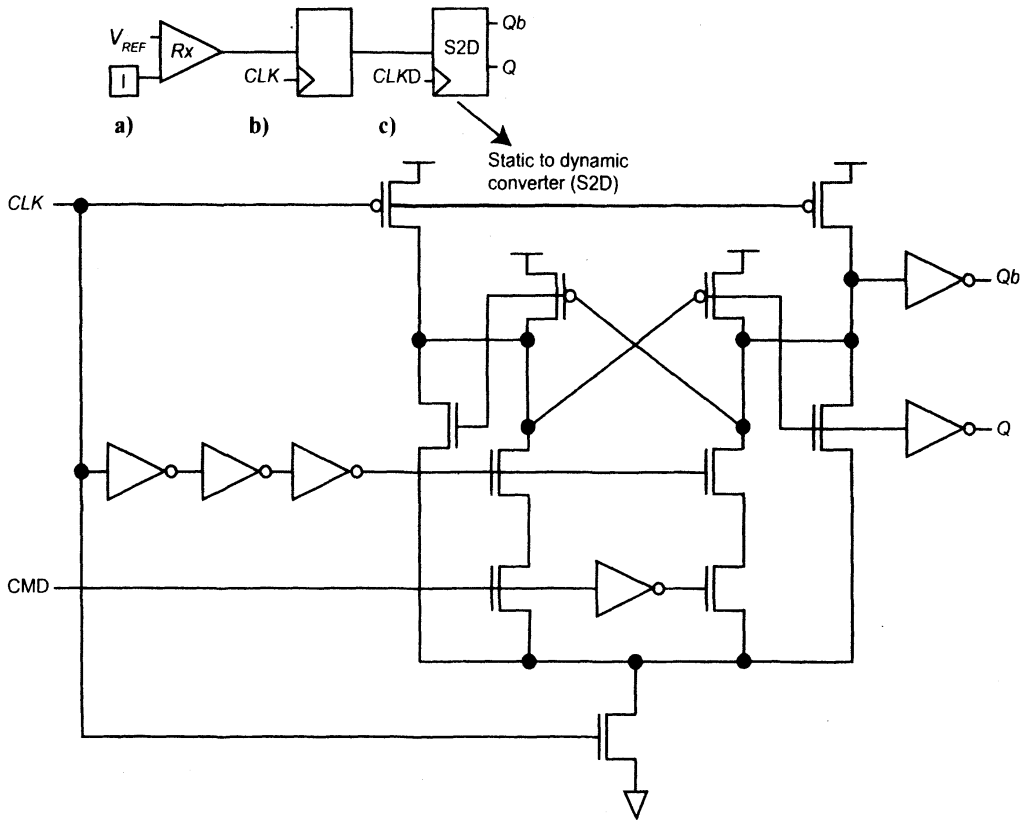


Figure 12.9 Dual rail static-to-dynamic conversion circuit.

The inset of Figure 12.9 serves as a point of reference for using the static-to-dynamic circuit in the DRAM input path. Figure 12.9a shows the connections of the single-ended input receiver circuit used to detect the incoming signal, followed by Figure 12.9b, the input capture latch, and then followed by Figure 12.9c, the static-to-dynamic converter circuit. This example would be common for the command and control input signals such as *RAS** and *CAS**.

At this point we should consider the question: Why use dynamic logic in the command input path of a high-performance DRAM? Part of the answer to that question is related to the process limitations discussed at the beginning of this chapter. Remember that DRAM processes implement as few metal layers as possible, making global clock distribution very difficult. Not only is it difficult in a DRAM process to distribute a well-matched clock, but using a global clock results in associated cycle delay dependencies on array access time, thereby limiting the usefulness of a globally distributed clock. Recall that an important timing parameter for DRAM performance is column access latency, which we have also referred to as *CAS* latency (t_{CL}). Because column access latency is constant across the range of operating frequencies for a device, we must not allow the clock

period to become part of the array access latency equation. Figure 12.10 is a diagram showing how internally synchronous operation would detrimentally allow the clock period to become part of the access latency. By synchronizing the command and address logic, we could potentially force a violation of the specified array access time. In Figure 12.10, the command and address are captured by the input capture latches (Chapter 9). If the following clock edge is used to align a partially decoded command (assuming that the total command decode plus synchronization overhead exceeds a clock cycle), the decode delay will be increased by the difference in clock period delay between the two operating frequencies shown. In this example, a device is specified with an operating frequency range of 800 MHz to 1.25 GHz. If we allow a clock cycle to occur between the command capture clock edge and one stage of the partial command decode, a t_{CL} delay difference of 900 ps will be added between the minimum frequency and maximum frequency operating points. Because of this difference, if devices are binned with this frequency dependency, many potentially high-performance devices may not meet the latency specifications.

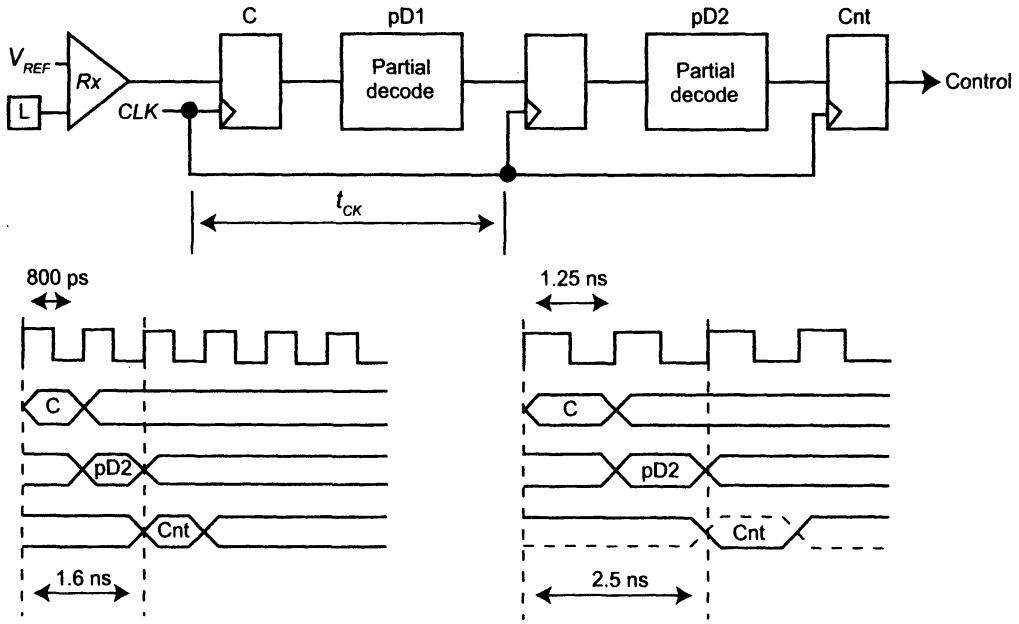


Figure 12.10 Example of a synchronous decode path.

One obvious solution for maintaining constant *CAS* latency across the operating frequency range is to use a combinatorial circuit for the decode operation and delay the input clock to a second set of retiming latches. The second set of retiming latches are required for realigning the decoder output to account for timing skew and hazards through the input and decode path; this configuration is illustrated in Figure 12.11. Without realignment of the decoder output, signal skew can cause false decoder transitions to appear in

the command and timing logic paths and possibly lead to data corruption in the DRAM array. One problem with this solution is that the clock must be delayed based on the maximum skew and delay through the decode path. If the delayed clock does not correlate with the delay through the input and decode paths, then, potentially, a high-performance device may not meet latency requirements at the highest operating frequency. For example, a device might meet timing requirements through the decode path but, because the clock is delayed based on worst-case timing expectations (with timing margin built in, of course), then any performance gains in the decode path may be masked by the delayed clock. This would be a greater concern for those devices operating right at the margin, since the delayed clock may prevent the device from binning at a higher performance point.

Skew between decoded signals is not the only problem with using combinatorial decode circuits for command and timing control. For high-performance devices with extremely low clock periods, the decode path could also have multi-cycle delay paths at higher clock frequencies but still satisfy single-cycle timing at lower clock frequencies. This problem gets back to the design and use of the same DRAM device over a wide range of clock frequencies. In this situation, false decodes could occur if the incorrect clock edge were to trigger a false decode. And, finally, using the simplistic method of a combinatorial decoder causes greater difficulty for final timing verification of the design. This is true because the global or, in some cases, multiple local clocks that are used for retiming the decoder output must be comprehensively verified for all combinations of decoder outputs.

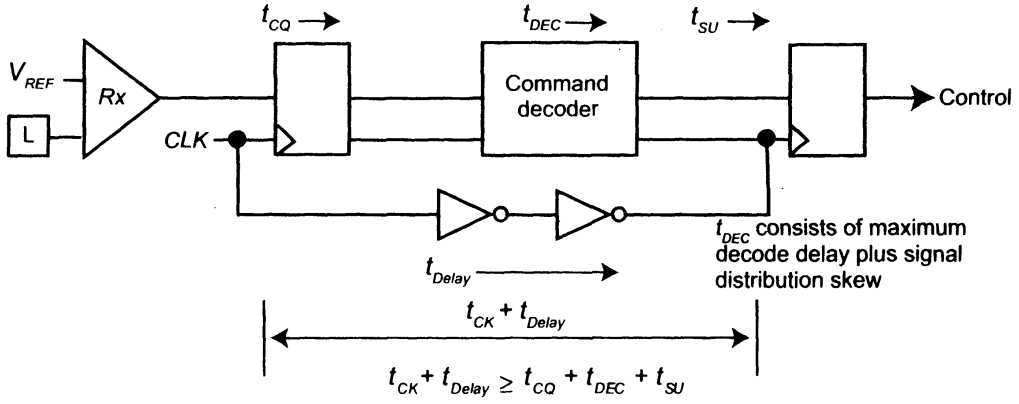


Figure 12.11 Example of a gate delayed clock decode path.

12.2.6 Testability

One disadvantage of using domino logic is the issue of testability. Because domino logic is dynamic, there are possibilities that at extremely low frequencies, such as the clock rates used during array testing early in the manufacturing process, charge could leak from the dynamic nodes and

cause incorrect operation. Because the design of monolithic DRAM largely follows a full-custom design flow, the implementations of scan chains for fault checking and automatic test pattern generation are disregarded. The ratio of the number of transistors used for the logic in the timing and control of the DRAM compared to the number of array transistors is so small that logic testing has largely been ignored in the test flow. Although, as package technology has advanced, the requirement for boundary scan chains, primarily intended for continuity checks during system level manufacturing, has become necessary. The issue of testability is a very broad issue and is only briefly treated in this section.

One very simple yet brute-force method for implementing logic that allows low-frequency array testing is to implement a parallel static logic interface for directly accessing the array peripheral logic. This method allows the device to be treated as a Page-Mode DRAM similar to the devices described in the early chapters of this book. This implementation does not test the high-performance peripheral logic interface, but it does allow low frequency testing and asynchronously controlled array timing stress to find weak array bits. When implementing a source-synchronous, matched-input-path methodology, there are often opportunities to generate static signals directly from the input logic paths. These static signals are often branched off of dummy buffer loads, which are used for load matching in the matched delay paths (Chapter 9).

Another method for low-frequency testing is to place back-end latches on the dynamic gates to isolate the dynamic node and provide static outputs. Figure 12.12 shows two examples of how static signals are generated from the dynamic gates. In case (a), the output of the domino gate is given a full weak inverter latch [1]. This differs from the usual p-channel keeper (Figure 12.8). The full latch is used on the first gate in a chain of domino logic to maintain both a HIGH and LOW state on the output of the gate. All subsequent gates sharing the clock phase can be designed with only the p-channel keeper, thereby maintaining the performance advantages of the domino logic. The output of this circuit follows the monotonically rising signal protocol previously discussed.

Figure 12.12b shows what is commonly referred to as an *N2Latch* [24]. Both the monotonic output and the static output from this gate can be used. The static portion is routed to test logic in the array and the monotonically rising outputs are ignored during low frequency testing. The static output of this circuit remains valid for a full cycle, unlike a monotonically rising signal output from Figure 12.12b. A disadvantage to this method is that the extra loading on the dynamic gates potentially increases gate delay.

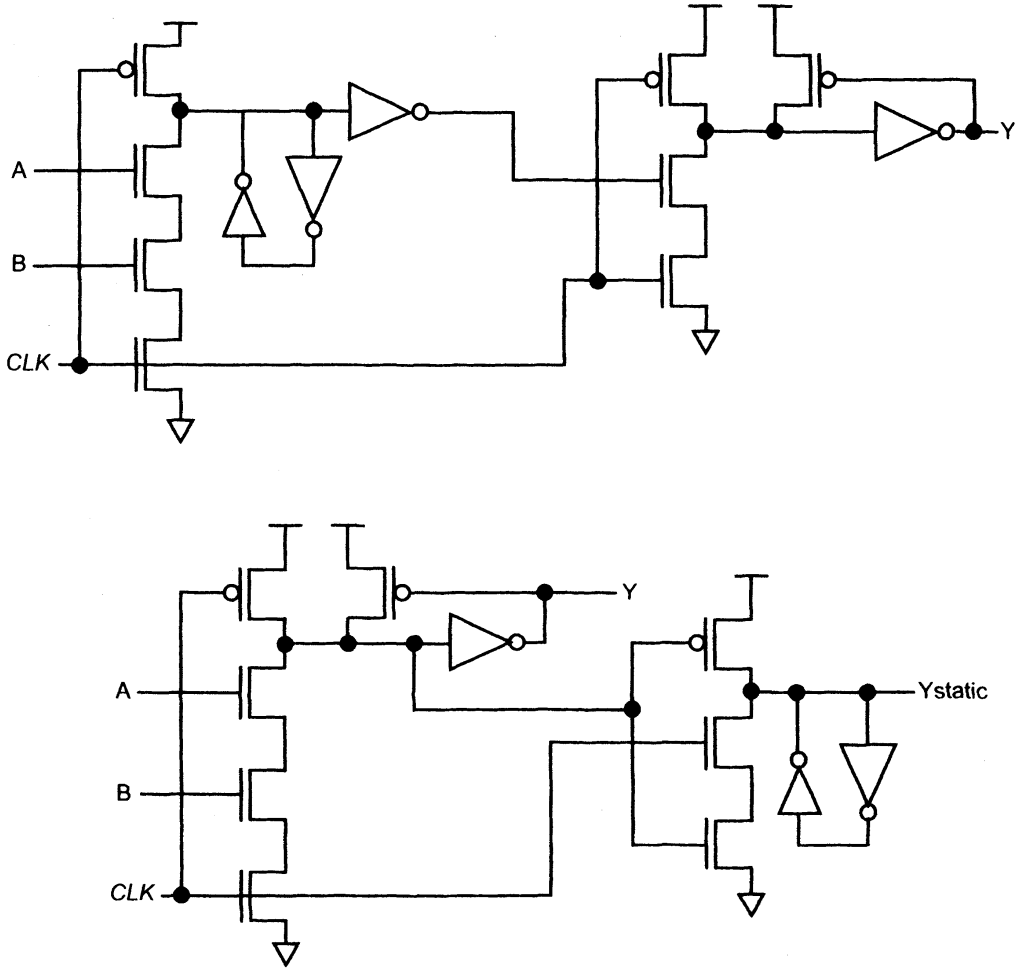


Figure 12.12 Static outputs from dynamic gates.

Now that we have examined some of the reasons for utilizing domino logic for portions of the timing and control circuits in the peripheral logic interface, we more specifically look at some of the issues related to command and timing control of array accesses and why we might want to make particular choices in logic implementations. As stated at the beginning of this chapter, these solutions are only one method to accomplish the task of meeting the industry standard specifications for a high-performance DRAM. The specifications are the thread of commonality that ties together the operation of DRAM between multiple vendors. The descriptions that follow are not meant as a design guide, but rather an illustration of the problems posed by the synchronous interface surrounding what has essentially remained an unchanged asynchronous DRAM. In the next section, we examine how domino-style logic can be implemented in the command decode and address register logic. Following that, we look at the data side of the DRAM and examine the Read and Write data paths and how the vari-

able latency of array accesses are made to fit within a deterministic timing framework established by the global clock.

12.3 COMMAND AND ADDRESS CONTROL

At the center of the peripheral logic interface is the command and address control logic. This area of logic is the interface between the command and address input circuit paths discussed in Chapter 9 and the timing and data *I/O* interface logic that we will examine in the sections to follow. Figure 12.13 is a composition of the primary protocol implementation for the peripheral logic interface with what we consider to be the control and address logic highlighted for reference. The functional blocks shown in Figure 12.13 will be the subjects of the remainder of this chapter.

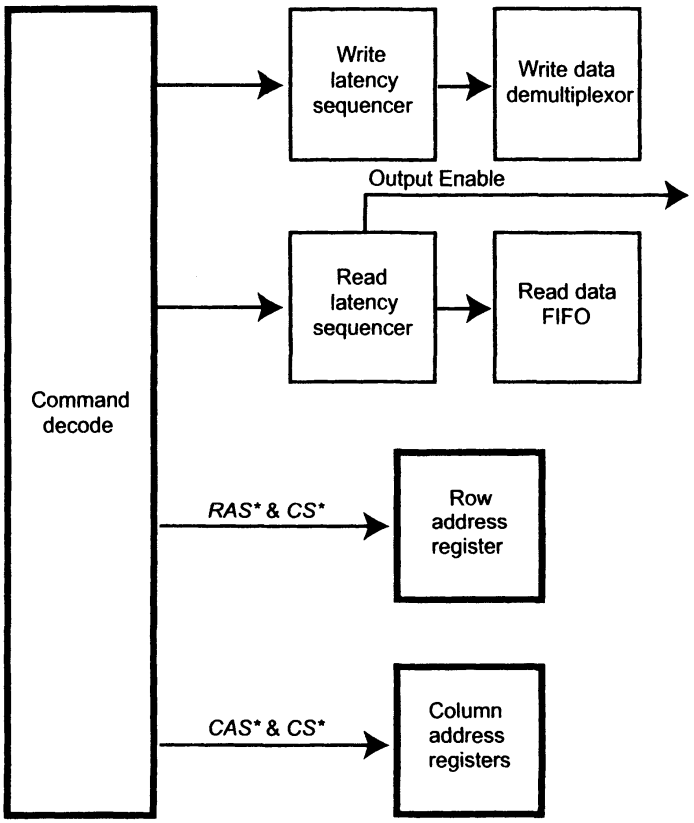


Figure 12.13 Primary functional blocks of peripheral logic interface.

Let us consider the context of each of the highlighted functional blocks of Figure 12.13. The command decode circuit combines control signals such as *RAS** and *CAS** in the context of the system clock and determines an action to take based on the command protocol. Depending on the command that is decoded, address inputs may need to be captured from the address input path. The address information for an array access is stored in registers based on whether a row or column access is decoded. The outputs

of the address registers are routed to the peripheral array logic and are decoded by the row and column decoders. The outputs of the address registers are not synchronized to a global clock but are instead bundled with signals that enable the peripheral array logic circuits. The control and timing signals are bundled and carefully routed in conjunction with the address signals. This enables correct timing when activating the row and column decoder circuits in the peripheral array logic. In particular, row address registers are often an extension of the peripheral array logic, with the style of logic resembling the delay-chain style described earlier in this chapter. The outputs of the row address register circuits are often referred to as *bank control logic* because each bank is determined by the division of row address decoders to sections of the DRAM array that can be independently accessed. We will not focus on bank control logic as its implementation depends on array architecture and is largely unchanged for high-performance DRAM, except for the case of reduced latency DRAM described in Chapter 8.2. Also, see Chapters 2, 3, and 4 for details on how row and column decode and control logic operate in the peripheral array logic area of the DRAM die.

The remainder of this section describes our choices for command decoder logic style and operation for a high-performance DRAM. We also discuss some architectural decisions that must be made with regard to the column address registers that are important to achieving low column access latency and how increased *I/O* bandwidth requirements affect the way that column addresses are handled.

12.3.1 Command Decoder

The command decoder is the control center for all DRAM operations. Control signals originate from this circuitry for every array access operation defined for the device. For high-performance synchronous DRAMs, the command decoder has taken a greater role in sequencing commands in accordance with the clock cycle time. This implies that the command decoder can be synchronous for some of its operations. The command decoder operation is becoming more important as the command and address protocol change due to increased memory density and interest in limiting the number of pins for each DRAM in a system. In some of the most recent DRAM command-address protocols, the command is not necessarily aligned with the complete address on the same clock cycle. Pin count can be reduced by splitting the address between multiple clock cycles to avoid having a wide, parallel address bus. Figure 12.14 shows different access modes for command and address protocols.

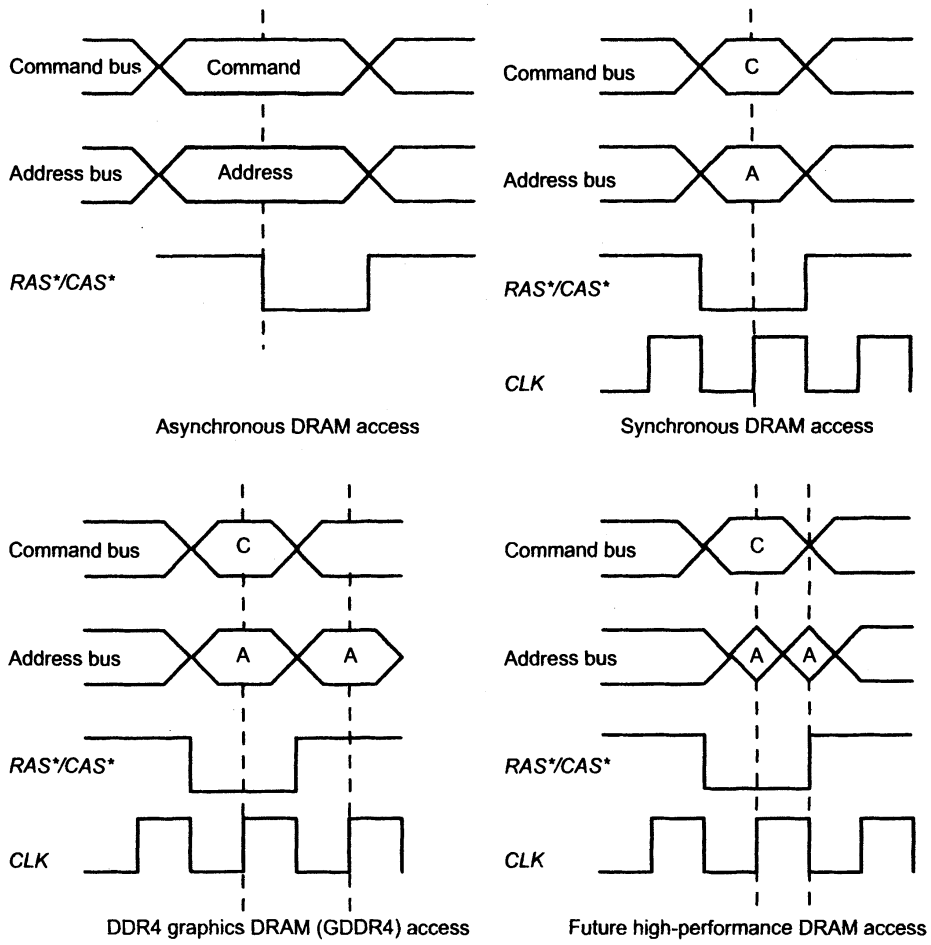


Figure 12.14 Command and address protocols.

Before synchronous DRAM, column accesses initiated through the command decoder were timed using a logic style that was the same or similar to the delay-chain logic style discussed earlier in this chapter. The command decoder was fully implemented with combinatorial logic, and a signal or signals that emulate the worst-case delay through the decode terms was generated similar to the scheme shown in Figure 12.11. These delayed signals were implemented as ENABLEs on combinatorial decode terms or as clocks on flip-flops or latches for retiming the decode output terms to avoid false decodes.

Following the command decoder is a block of logic that generates and sequences the control signals of the DC sense amps (DCSA) or helper flip-flops (HFF), which are described in Chapter 5. Of course, this logic is still implemented in the peripheral array logic of high-performance DRAM. This group of logic circuits is often referred to as the *CAS chain*. With synchronous DRAM, data accesses are sequenced by the system clock, which establishes the conditions that allow the command decoder to be sequenced

by clock-driven signals. This is because the data sheet specification dictates the cycle delay between commands.

Our discussion of the command decoder synchronization began in Section 12.2.5. In that section, we discuss the advantages of domino logic and how this style of logic can lead to *gate-delayed* operation. What we mean by "gate-delayed" in this context is that the logic operations occur independent of a global clock signal and, therefore, independent of clock period delay as well as of the synchronous overhead of setup, hold, and *CLK*-to-*DATA* delay associated with synchronous circuits. Remember that input logic must be independent of clock period because column access latency is constant regardless of clock frequency.

Figure 12.15 is an example of a gate-delayed logic path implemented with domino logic. This logic path generates a signal to indicate that the device has received a Read command. The command inputs *RAS**, *CAS**, *WE**, and *CS** are distributed through the input circuit path and latched in the input capture latches. In this example, because the delay through the capture latch is slightly greater than the minimum clock period, the clock signal is delayed by buffer delay to meet the timing requirements of the dual-rail static-to-dynamic conversion circuit (Figure 12.9). This delay is inserted following the critically matched signals of the input capture path (Chapter 9). The Read command is indicated by *CAS**=0, *RAS**=1, *WE**=1, and *CS**=0. The dual-rail conversion circuit (DE) generates monotonically rising signals from the clock edge for each of the signal conditions. For example, *WE**=1 would generate a rising output on the *Q* signal of the conversion circuit, and the *Qb* signal would remain LOW as a result of the preceding Precharge period. The signals resulting from the conversion stage are routed through successive domino stages that are clocked directly by the dual-rail signals. The domino gate X5 is clocked by the *CASQ** signal with the *CSQ** and *RASQ* signals combined in the dynamic AND gate. The output signal *Column* of this gate indicates that a column command is potentially received. The signal *Column* is then combined with the *WEQ* signal in the dynamic gate X6 to generate the monotonic signal *Read.* Since the clock signal is only used for the first stage of logic, we can consider the logic path following the clocked gates to be gate delayed because the logic delay is based on the gate delay through the path and has no direct timing dependencies on clock period.

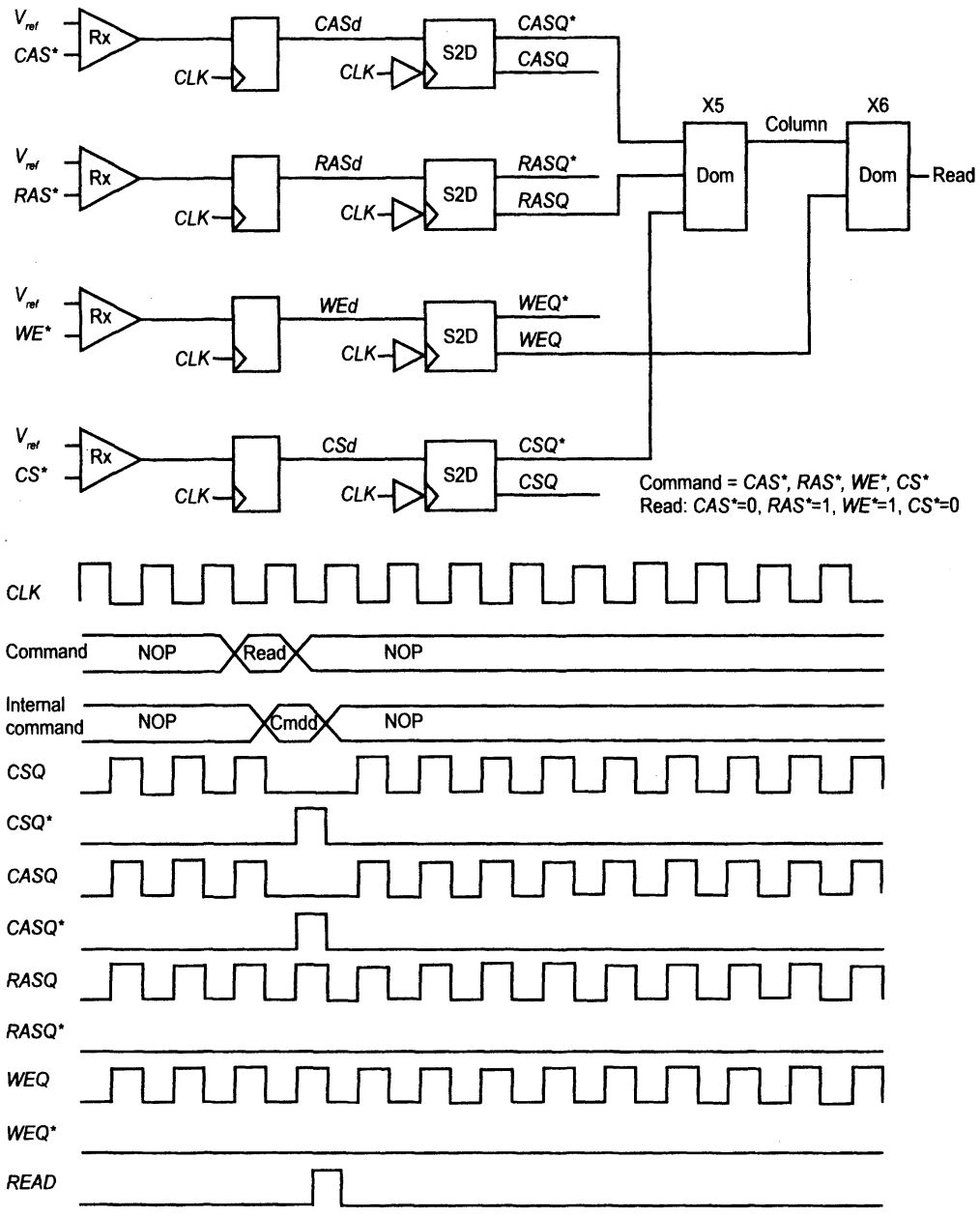


Figure 12.15 Gate-delayed domino logic decode path.

Again referring to Figure 12.15, the timing relationship between domino gates X5 and X6 can take advantage of the footed domino gate so that the *Read* signal has a pulse width of one half of a clock period. If we use the signal *Column* as the clock for gate X6, then the only timing requirement is that the *WEQ** meet the hold time requirement for gate X6. [13]. These timing relationships are shown in Figure 12.16 with t_{DQ} designating the delay through the footed domino gate.

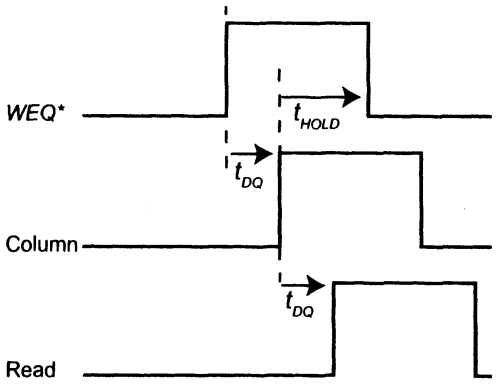


Figure 12.16 Hold time requirement for domino gate.

One of the advantages of using domino logic for generating the decode and timing signals is that if a device suffers logic timing issues, they can usually be overcome, and correct operation is attained, by simply reducing the clock frequency. In Figure 12.16 we see that reducing the clock frequency increases the pulse widths, which results in more hold time for the final decoder logic stage. Reducing clock frequency has advantages for synchronous static logic as well, but recall that the advantages of domino logic include reduced reliance on clock distribution networks, skew tolerance, and a reduction in timing overhead.

The example just described is for simple, single-phase clock implementation. The pulse width of the signal *Read* is approximately one half of a clock period wide. For extremely high frequency, we may be forced to divide the clock frequency to get wider pulse widths on the decoder output signals. A wider pulse width on the decoder outputs helps ensure better signal integrity when distributing the signal over long distances. For this situation, each gate would require two phases of dynamic logic with the last static gate in the logic path implementing a merge function so that a single decode signal would be generated with a full-cycle pulse width. clock frequency division is only possible when the array access cycle times are multiple command clock cycles. For instance, if the burst length (BL) specification were a minimum of four for a DDR device, then, given that the command clock and data strobe frequencies are equal, the column cycle time, as determined by Equation (12.3) would be two clock cycles. Clock frequency division under these conditions would be possible. A frequency divided decode logic path and timing diagram is shown in Figure 12.17.

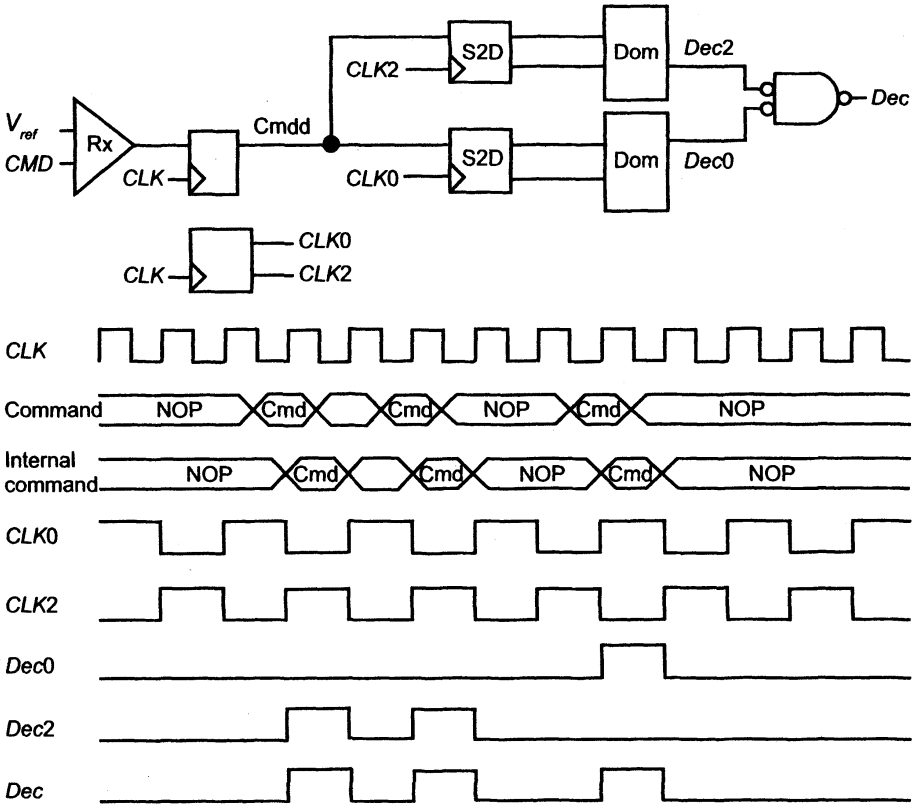


Figure 12.17 Frequency divided gate-delayed domino logic decode path.

Depending on the circuits used in the input circuit path, we may be able to take advantage of the receiver circuits for generating monotonic signals. The single-ended signaling protocol is the most common signaling protocol for monolithic DDR DRAM. This signaling protocol applies a reference voltage on one input of the receiver differential pair, with the target signal connected to the opposite input (Figure 12.17). An example of this topology is shown in Figure 12.18 (a). The latch is separated from the receiver, which allows matched delay interconnect distribution for *CLK* and *DATA*. For high-performance DRAM, a receiver sense-amp latch (SA latch), shown in Figure 12.18b, can be used as a combination input detection circuit and latch. The SA latch circuit tightly couples the analog detection circuit with a static latch. The detection circuit of the SA latch naturally outputs a dual-rail, monotonic signal to toggle the inputs of a set-reset latch. If these signals are buffered out of the SA circuit, both the latch stage and single-ended to double-ended conversion circuits (shown in Figure 12.9) are bypassed. As a side note, the SA latch can also be used for differential signaling. This signaling technique has disadvantages when applied to DRAM because of increased pin count. For differential signaling, channel bandwidth must be

increased to achieve equal bandwidth-channel product compared with single-ended, wide bus *C/A* paths.

Once a partial decode is detected, enough information is available to determine if an address must be loaded into the address register logic. The address register logic is a secondary register set that reflects the contents of the input capture latches when a valid column or row command is received. The input capture latches are updated on every cycle of the clock, and the secondary registers are loaded only if a valid *RAS** or *CAS** combined with a valid chip select is received. In the next section, we examine some issues related to implementing the secondary address register, focusing on column address operations.

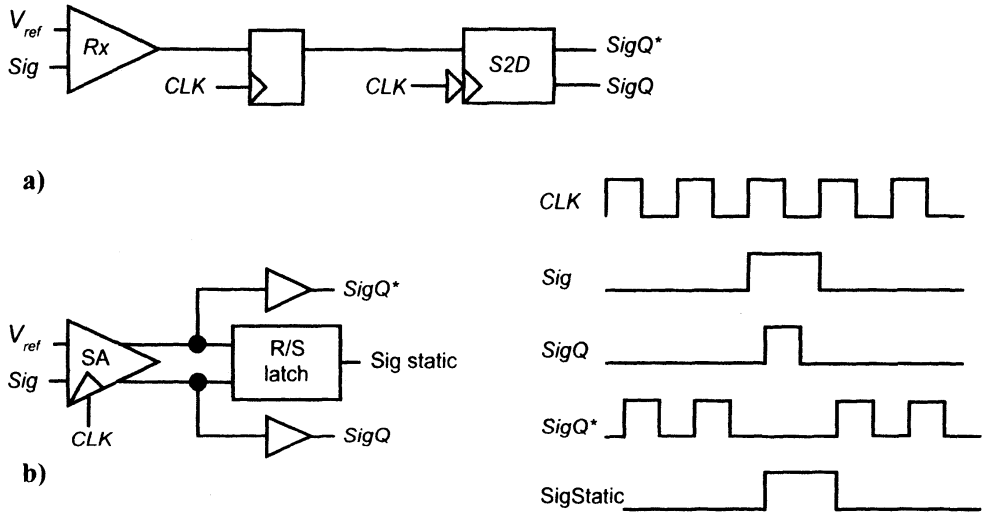


Figure 12.18 SA Latch dual-rail conversion.

12.3.2 Read and Write Data Address Registers

In early synchronous DRAM, the minimum command and address setup times to the clock were long enough that DRAM designers would often cheat and use the *CAS** setup time to allow address signals to start to propagate into the device through transparent capture latches. Doing this allowed the address signals to propagate into the device and achieve faster column access time for Read operations. This essentially made the synchronous DRAM look internally like an asynchronous DRAM device. For our high-performance example, we look at an address topology that incorporates a register that is secondary to the address capture latches. These secondary latches are controlled and timed from the command decoder circuit. The functional relationship between the command decoder and the secondary address registers is shown in Figure 12.13.

For array Read accesses, we want to retrieve data from the array with minimum latency. We can determine the delay for array accesses by break-

ing the delay into two components: array access latency and command decode latency. Array access latency is the delay measured from the enabling of the column address decoder to the arrival of valid Read data from the helper flip-flops (Chapter 5). We designate this delay as t_{CAA}. The command decode latency is a measure of the sum of the input circuit delay, the command decoder delay, and the address register logic. We designate the total command decode delay as t_{CMD}. Considering these definitions, in order to minimize array access latency from the perspective of the control logic, we must distribute the address to the peripheral array logic and enable the column address decoder with the minimum possible delay.

By partially decoding the command, a monotonic signal that has a pulse width of one half of the valid address period is generated. This allows the use of transparent latches for the address register. The latches are transparent when the *CASQ** signal, qualified by the chip-select, is HIGH. The timing of the latch signal aligns with the address transitions from the input capture latches so that the address signals start to transition on the address distribution path following the t_{DQ} (data to output) delay of the latch circuit. The timing relationship between the latch signal and the captured addresses is advantageous since the latch signal is one half of an address cycle wide while the address signals are one clock cycle wide; thus, the critical timing parameter is t_{CP} as shown in Figure 12.19.

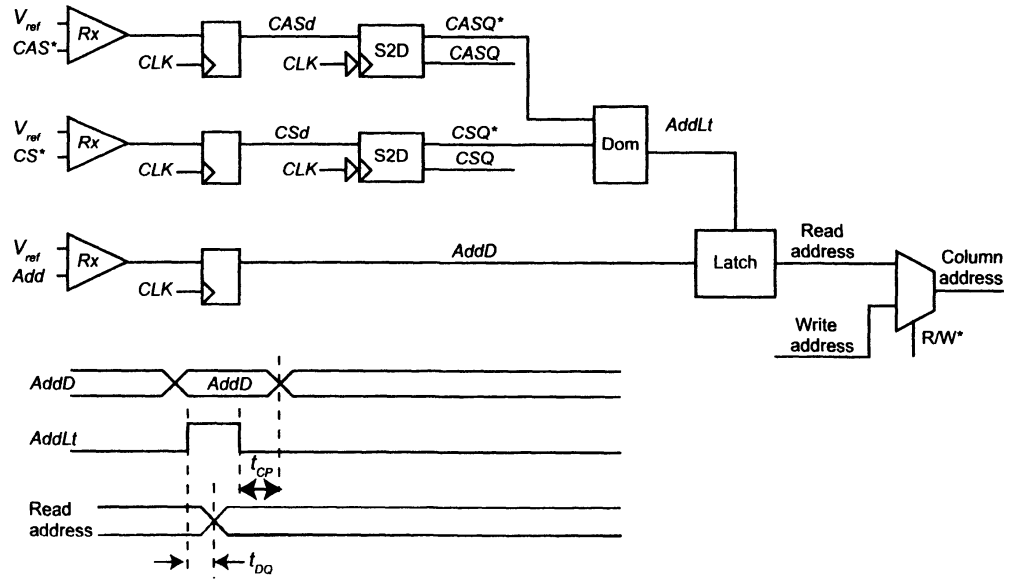


Figure 12.19 Read column address register timing for transparent latches.

The column address for Writes must be handled differently from the Read column address. For Write commands we must wait until the Write data is available before the column address decoder can be enabled. Figure 12.20 is a timing diagram showing how the arrival of Write data is delayed

by the Write latency specification, t_{WL}, and the burst length, t_{BL}, for data arrival at the Write drivers in the peripheral array logic. Notice that in this example, data is not available to be retired to the array before the end of the first burst of four data bits. While the device waits to receive all of the Write data, Write commands and addresses continue to be received. This differs from the column address for Reads where the address is retired as quickly as possible to achieve minimum array access latency. That is why there is a mux in the address path shown in Figure 12.19 to illustrate that the Write address originates from a different register than the Read address.

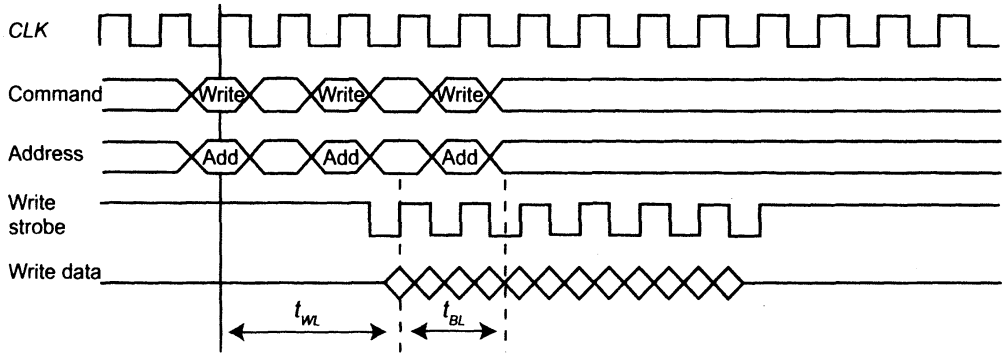


Figure 12.20 Write data timing relationship to Write command and address.

For our high-performance DRAM logic example, the Write addresses are loaded into a first-in first-out (FIFO) circuit. There are several design alternatives for FIFO circuits. The first consideration is determining the maximum Write address FIFO depth required. Knowing the FIFO depth can help select the type of FIFO circuit best suited to the design specifications. Again referring to Figure 12.20, we see that the first address must remain available until the Write data associated with that address has arrived. For the situation in Figure 12.20, we see that the maximum FIFO depth is determined by the maximum specified Write data latency and the minimum allowed burst length. Remember that column cycle time (this is also column address cycle time) can be expressed as a function of burst length (Equation (12.3)). The burst length is equal to the Prefetch depth for each data *I/O* and in our example t_{BL} is equal to column cycle time. If t_{WL} and t_{BL} are expressed as a function of discrete command clock periods, then we can determine the maximum depth of the FIFO. Equation (12.4) is an expression for Write address FIFO depth where FD is the FIFO depth, WL_{max} is the Write latency expressed as an integer number of clock cycles, and t_{WT} is Write data transfer overhead for transferring the Write data and column address to the peripheral array logic. The FIFO depth must be rounded up to the nearest integer for fractional results.

$$FD = \frac{\frac{BL_{Min}}{2} + WL_{Max}}{\frac{BL_{Min}}{2}} + \frac{t_{WT}}{t_{CK}} = 1 + \frac{2WL_{Max}}{BL_{Min}} + \frac{t_{WL}}{t_{CK}} \quad \textbf{(12.4)}$$

Ignoring the transfer overhead, t_{WL}, we find that for our example in Figure 12.20, FD=3. Of course, we can not ignore the transfer overhead, which is partially dependent on the address FIFO design chosen.

One possible Write address FIFO design is a simple asynchronous FIFO [14] [15]. The advantage of using an asynchronous pipeline design is that there is very little overhead in the circuit implementation and, if the design is carefully chosen, the forward latency through the pipeline can be minimized. Another advantage to the asynchronous pipeline design is compatibility with the signaling protocol for decoded addresses as examined earlier in this section. If we generate a decoded Write command similar to the decoded Read command in Figure 12.15, we can use the generated WRITE signal to load the Write address FIFO directly from the column address register. Figure 12.21 is a top-level block diagram showing this configuration. The timing parameter, t_{FL} is the forward latency through the asynchronous address FIFO. This is the sum of delay for each latch controller stage in the FIFO. This delay, summed with the address distribution delay, must be less than the minimum Write data arrival time shown in Figure 12.20.

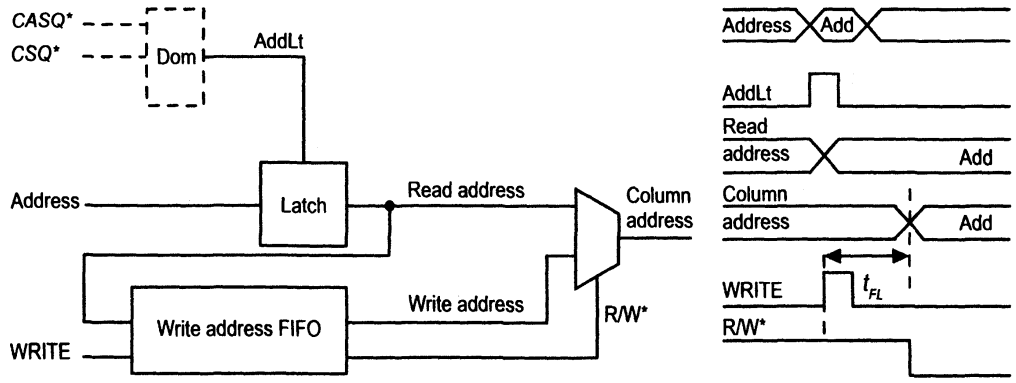


Figure 12.21 Column address register and MUX control.

There are many details surrounding the design of the Write address FIFO. If an asynchronous FIFO is used, the throughput must be considered, especially when the FIFO is at capacity. Also, the asynchronous channel communication between latch stages must be considered for compatibility with the logic-signaling environment in which the FIFO is placed. A careful examination of the references provided should edify the reader on these

issues with details of the FIFO design and implementation that are beyond the scope of this text. Primarily, we must be aware of how the DRAM specification affects the address distribution logic in the peripheral logic interface area of the DRAM device.

In the next section, we examine one method for enabling the data access control logic once the command and address are fully received. This method involves a two-wire interface with the control signals bundled with the address and data signal busses. We illustrate how the asynchronous nature of the DRAM array affects the Read and Write *I/O* data paths.

12.3.3 Column Access Control

Continuing our example of a DRAM logic interface, we now examine a simple two-wire signaling protocol for controlling timing and cycle time for array accesses. For data accesses, once the command is decoded and the address is loaded into the appropriate address register, a signal is generated and distributed to the DRAM array banks. We call this signal, *colCyc,* and its function is to time the arrival of column addresses at the peripheral array logic. A second signal is generated in the peripheral array logic when Read data is available for transfer into the peripheral logic interface of the DRAM. We refer to this signal as, *dataValid.* It is important that we understand the function of these signals as these are the primary means for communicating data accesses between the peripheral array logic and the peripheral logic interface illustrated at the beginning of the chapter in Figure 12.13.

Figure 12.22 is a timing diagram showing the timing relationship between *colCyc* and the column address for Read and Write addresses as well as the timing relationship between the *dataValid* signal and Read data driven from the HFF circuits in the peripheral array logic. These signals follow what is commonly referred to as a *bundled signal protocol.* Simply stated, the bundled signal protocol is the case of a single timing signal that is driven concurrently with transitions of a signal bus. The bundled signal is meant as an indicator that data on a bus has changed. This may sound like a clock signal but, unlike a clock signal, it is not a continuous signal, and it only transitions when there are transitions on the associated bus.

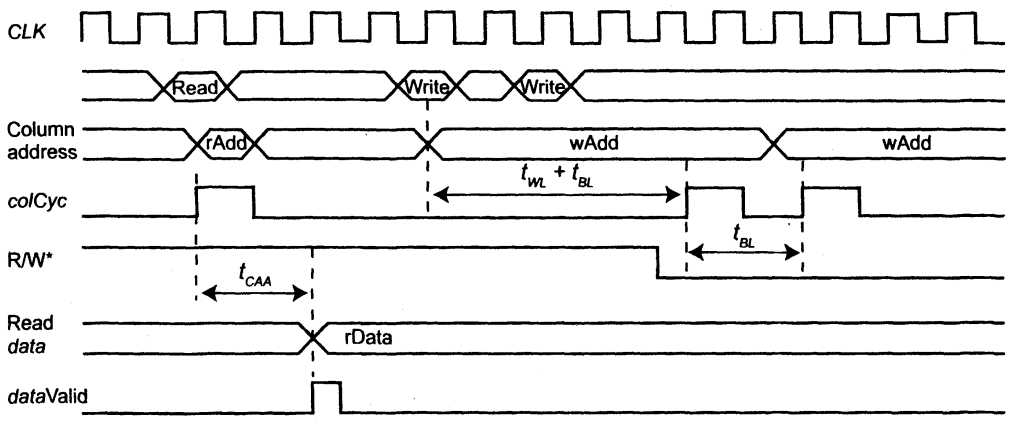


Figure 12.22 Two-wire bundled signal interface for data accesses.

Notice in Figure 12.22 how *colCyc* transitions are timed when there is a Read access versus a Write access. The *R/W** signal is included to indicate when a Read or Write occur. Remember that the DRAM array really should be considered a Read-only device when a column address is enabled. Writes are a result of overdriving the array sense amplifiers through Write driver circuits. For Reads, the access delay is partially determined by the command decode delay and address distribution logic. Writes are partially delayed by the Write data latency and burst length. This is shown in Figure 12.22.

The *dataValid* signal is sent from the peripheral array logic to the peripheral logic interface whenever Read data is driven from the HFF circuits. This signal is very important to the operation of the Read data path in the peripheral logic interface because t_{CAA} will vary with changes in temperature and voltage. This signal must be considered fully asynchronous without reference to any clock signal generated near the array interface. On the other hand, the *colCyc* signal does have an established cycle time based on the command clock. This becomes a more important distinction when we examine the Read data path in Section 12.4.

At this point, we have examined how process can determine which logic style we might choose for implementing DRAM access functions. We have also illustrated an application of domino-style logic and how exploiting the gate-delayed potential of domino logic allows us to maintain constant data access latency across a wide range of clock frequencies. Further, we have examined how the DRAM specification can affect how the column address is treated for both Read and Write accesses. But there are many details that have been left out of our discussion of DRAM peripheral logic implementation. The delay-chain logic style implemented in the peripheral array logic is more often an art form than an academically rigorous logic style. There is also the wordline control logic that we have mostly ignored.

These sections of logic are important but the details do not contribute greatly in our discussion of some of the unique problems encountered in high-performance DRAM logic design. Therefore, the remainder of this chapter is devoted to examining the data *I/O* circuit paths between the peripheral array logic and the data input and output paths. This will include an examination of the Write data serial-to-parallel conversion circuits (demultiplexors) and the Read data FIFO circuits, which are necessary to decouple variations in array access latency versus the synchronous Read latency timing specifications of the DRAM.

12.4 WRITE DATA LATENCY TIMING AND DATA DEMULTIPLEXING

In this section, we consider the Write data demultiplexor circuits or what are more commonly known in DRAM circles as the *write pipes*. This set of circuits follows the data capture latches of the input circuit path discussed in Chapter 9. The purpose of these circuits is to convert the input serial data stream into parallel data with a bus width and cycle time that is compatible with the Prefetch data depth of the array. For example, if the per-bit Prefetch depth of a column access is 8 bits, then each Write pipe will be required to convert 8-bit data bursts into parallel data for each column access.

There are two distinct processes that occur when a Write command is received. First, based on the predetermined Write latency, the input data path is enabled to coincide with the arrival of data and any synchronous reference, such as a data strobe, at the data inputs. (Programmed Write latency is sometimes formulaic to Read latency or discretely programmed in a mode register, as discussed in Chapter 2.) Programmable Write latency is intended to allow time between the receipt of a Write command and the arrival of data at the data inputs to the device when operating at high clock frequencies. The reason for this delay is obvious from our discussion of the command decode path, in that the delay through the command decoder can span multiple clock cycles when operating at very high frequencies. Write latency is defined as the delay between the Write command and the arrival of the first bit of data of a Write data burst. This delay is measured in command clock cycles. An example of the Write latency specification is shown in the timing diagram of Figure 12.23. Included in Figure 12.23 is the parameter t_{DQSS}, which is discussed later in this section.

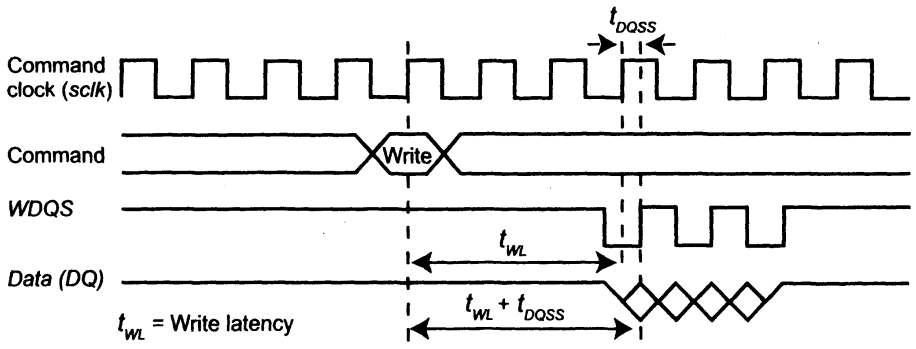


Figure 12.23 Write latency definition.

Following a Write command and data capture, there is a serial-to-parallel data conversion, or what is often referred to as *data demultiplexing*, in the data path. The primary purpose for data demultiplexing is to match the data rate at the input pins with the column cycle time of the array. For example, if the array column cycle time is 3.2 ns, and the burst granularity is 4, the maximum data rate would be 1.25 Gb/s. If we change the burst granularity to 8, then the data rate becomes 2.5 Gb/s. Figure 12.24 is a top-level view of the data demultiplexing operation for reference.

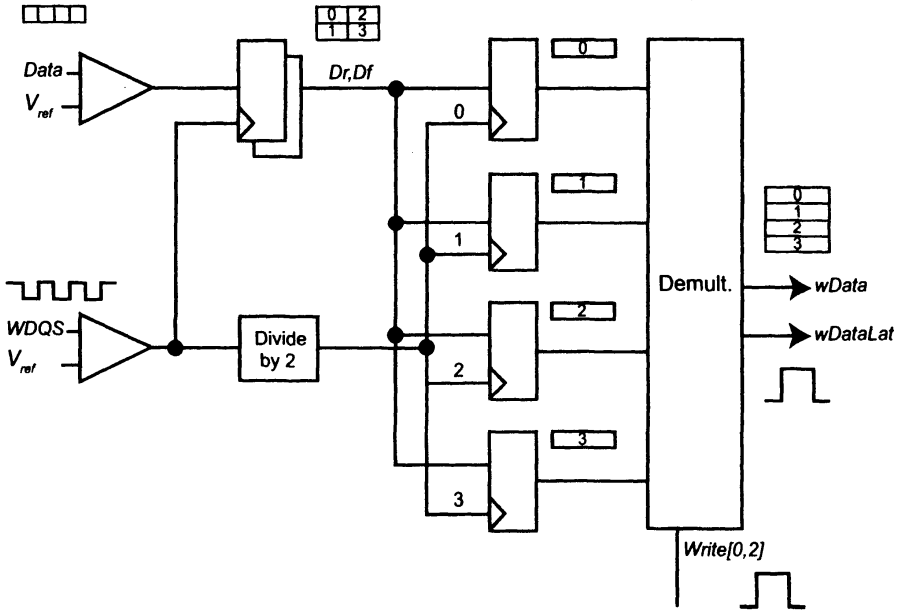


Figure 12.24 Demultiplexor in DRAM Write data path.

In the remainder of this section, we first consider the necessary steps to meet Write latency timing. Predicting Write data arrival based on programmable Write latency values is fundamental enough to DRAM Write data capture that any future specifications implemented in high-performance DRAM will continue to benefit from the current state of the art. However,

some have proposed DRAM with interfaces that are based on data packet protocols [16], which would potentially eliminate the need to track Write data latency. At the present time, a packet-based DRAM interface is intended for niche applications. The packet-based interface is attractive because of narrow input ports and, hence, low pin count. However, the per-pin data rates are necessarily higher in order to match the bandwidth available from alternative devices with wider input ports. There has not been widespread use of packet-based DRAM, so this protocol is not discussed in this text. Following the discussions of Write latency timing, we examine, in greater detail, one method for demultiplexing the data driven from the input circuit path and how data is assembled and eventually driven to the peripheral array logic circuits. The discussion about demultiplexing will also consider issues related to clock frequency division that, for a DRAM, is often advantageous when dealing with data rates greater than 2 Gb/s.

12.4.1 Write Latency Timing

When SDRAM operating clock frequency specifications were based on the column cycle time capabilities of the DRAM, random access was possible and data Prefetch was unnecessary since the specification allowed full random, cycle-to-cycle Read or Write accesses. For SDRAM, there is not a programmable Write latency specification; the Write data arrival time is concurrent with the Write command at the device inputs. Write data can burst for several cycles, but the LSBs of the column address are automatically changed for each data bit and driven with data to the peripheral array logic. For DDR SDRAM, however, data Prefetch is a necessity in order to ensure that the throughput of the data *I/O* circuits do not become data limited because of the array cycle time. As was emphasized earlier in this chapter, the DRAM array architecture and performance has changed very little between early, asynchronous designs and high-performance designs.

Aside from Write latency, another innovation that is part of the DDR DRAM specification is the data strobe. The data strobe is a burst clock that is aligned to the center of the data eye for Write data and nominally aligned to the transition edges of the data eye for Read data (from the point of view of the DRAM). For Write data, the data strobe serves as a source-synchronous [17] signal that relieves some of the internal clock distribution difficulties that result from the SDRAM specification. SDRAM uses the same system clock as a timing reference for both data and commands; whereas, DDR DRAM protocol specifies separate timing references: the data strobe for data and the *C/A* clock for commands. Having a source-synchronous data strobe allows tighter timing for data capture and also allows internally delay-matched data and clock distribution between the data inputs and the

data latches, as described in Chapter 9. For commodity-level DDR DRAM, the data strobe is bi-directional. For high-performance DRAM, such as graphics DDR DRAM, there are separate, unidirectional data strobes that are distinguished based on Read and Write commands.

There are two fundamental timing specifications related to the data strobe. The first timing specification is concerned with alignment of the strobe to the command clock, while the second specification is the data setup and hold time relative to the data strobe. The data strobe transitions are specified to nominally align with transitions on the command clock to help accommodate deterministic latency relative to Read or Write commands. Because of both data-dependent noise and channel imperfections (simultaneous switching operations, clock jitter, inter-symbol interference, and channel cross talk), it is necessary to specify a finite amount of skew between the data strobe and command clock. This skew specification, illustrated in Figure 12.25, is commonly referred to as t_{DQSS}. Skew between the command clock and the data strobe does not alleviate the data setup and hold timing specifications between the data strobe and data signals. The timing parameter t_{DQSS} is a component of Write latency and specifies that valid data must fall within a deterministic timing slot relative to the command clock.

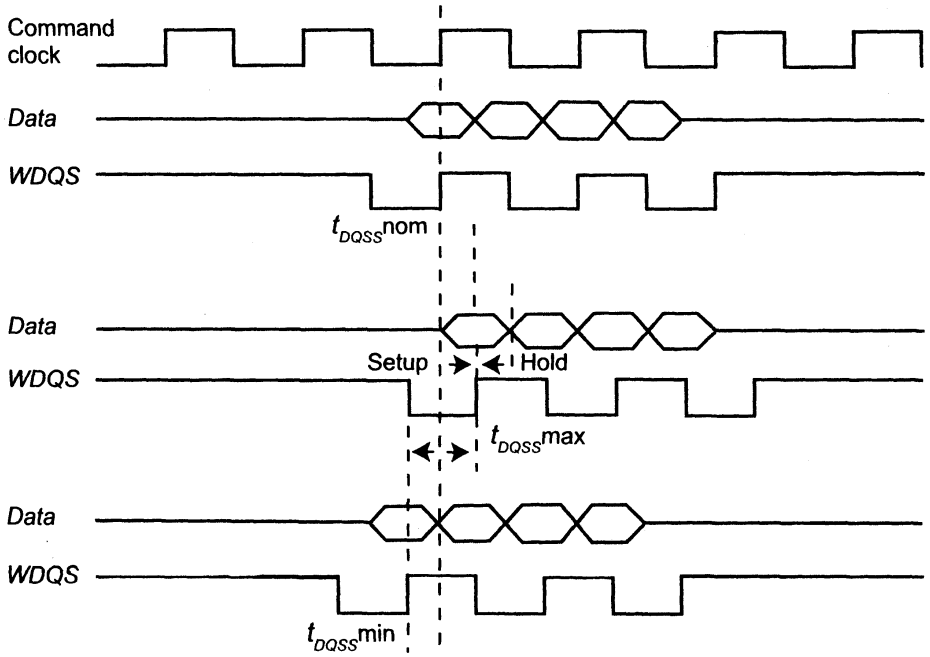


Figure 12.25 Write data strobe and t_{DQSS} definition.

Another purpose for variable Write latency is to allow time to enable and disable the Write data receivers. For high-performance DDR DRAM, more aggressive receiver designs are often necessary. More sensitive

receivers are often not self-biased but instead require a biased front end (Chapter 9). For some generations of DDR DRAM, the data strobe is bi-directional and center terminated (the signal is terminated to the middle of the input signal swing range). Also, if the input data receivers are biased to an active state, the receivers constantly consume power. Write latency allows time between the Write command and data so that the receivers can be enabled only during valid Write data periods. On the other hand, with supply terminated signaling (high-performance GDDR designs implement supply terminated signaling), the receivers can be turned on with little power consumption because if the bus is idle, the termination resistors pull the input signal to the supply voltage. Also, with a supply-terminated data strobe, there is less likelihood of false strobe toggles propagating into the device due to noise occurring around the switching point of the receiver. With a center-terminated bus, there is a very high probability of false toggles on the data strobe if the receiver is enabled during idle bus states, which, depending on how the logic is implemented, could lead to data and logic failures in the capture and demultiplexing logic. Overall, Write latency is a timing mechanism that increases the timing margin between decoding the Write command and the arrival of input Write data, and it also allows time to enable and disable the data strobe and data input receivers for more reliable, lower power operation.

Without getting distracted by the difficulties posed by specification-related problems such as termination or protocol, the fundamental design problems related to Write data latency for high-performance DDR DRAM are associated with the separate clocking domains of the command clock and Write data strobe (*WDQS*). There is some relief in that the *WDQS* is specified to have a bounded skew (t_{DQSS}) relative to the command clock. Referring back to Figure 12.13, we see that connected to the command decoder is a block labeled *Write latency sequencer.* This block essentially represents a counter that counts command clock cycles in order to align timing signals generated from the command clock domain to converge with the arrival of data from the inputs of the device all timed to Write latency. The problem with this configuration is that the data *I/O* are often located a long physical distance from the command decoder, so that sending a signal across the die to synchronize the completion of the demultiplexing function requires very careful, customized design.

Figure 12.26 defines signals that we discuss for the remainder of this section. Signals *Write[0]* and *Write[2]* are driven from the Write sequencer logic. These signals are used for completing the sequence between tracking the deterministic Write latency in the command clock domain and the arrival of data indicated by valid transitions of the *WDQS* signal. We consider the Write latency sequencer to be part of the command decoder cir-

cuitry. The timing diagram in Figure 12.26 shows the relationship between *Write[0,2]* and the assemblage of Write data within the demultiplexor block of Figure 12.24. Also, Figure 12.27 is a diagram showing the physical relationship between the data input circuit paths and the command decode and latency timing circuits.

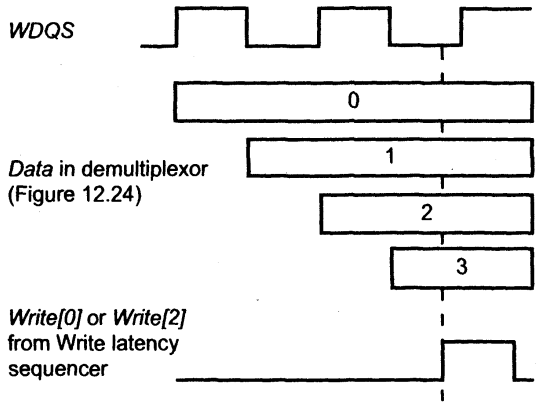


Figure 12.26 *Write[0,2]* definition.

The diagram in Figure 12.27 shows a *matched-mode* timing method for enabling the demultiplexing circuits relative to the arrival of data based on Write latency. We refer to the topology in Figure 12.27 as "matched-mode" because the *CMDCLK*[0:3] signal distribution is delay matched to the divided *WDQS* distribution, which is labeled t2 in Figure 12.27. These two disparate circuit paths can be nominally delay matched because the *CMDCLK*[0:3] signals can be point-to-point routed to the demultiplexor timing circuits while the *WDQS* and *C/A CLK* signals must be branched and matched across a physically wide routing area (see Chapter 9).

There are a few interesting features in Figure 12.27 that explain how a mid-performance device can time Write data latency using a "brute-force," match-mode method. First, notice that as the *C/A CLK* and *WDQS* enter the device, both signals are frequency divided. Frequency division is implemented to gain greater timing margin for logic in the data demultiplexor circuits and for distribution and capture of the *Write[0,2]* timing signals into the demultiplexor sequencing logic. The data capture latches for both command and Write data are clocked by the full-frequency clocks. It is only following the capture latches that frequency division is utilized to clock the downstream logic. In the example shown, the input, *C/A CLK* is divided into four phases (*CMDCLK*[0:3]). This clock division can be accomplished using a phase generator circuit (Chapter 11). The key to achieving accurate Write latency timing is to align the arrival of the divided *WDQS* signal with timing and sequencing signals *Write[0,2]*. These signals are synchronized by *CMDCLK*[0,2] from the Write latency sequencer at the command decoder and distributed directly to the frequency divided *WDQS* domain.

The *Write[0,2]* signals (these signals are labeled 0,2 because they are only generated from phases 0 and 2 of *CMDCLK*[0:3]) are then retimed at the control logic for the demultiplexor logic circuits. The reason *Write[0,2]* is retimed at the demultiplexor circuits with the command domain clock is because *CMDCLK*[0:3] is continuous; whereas, *WDQS* is a burst strobe that only toggles when data is driven into the device (Figure 12.25).

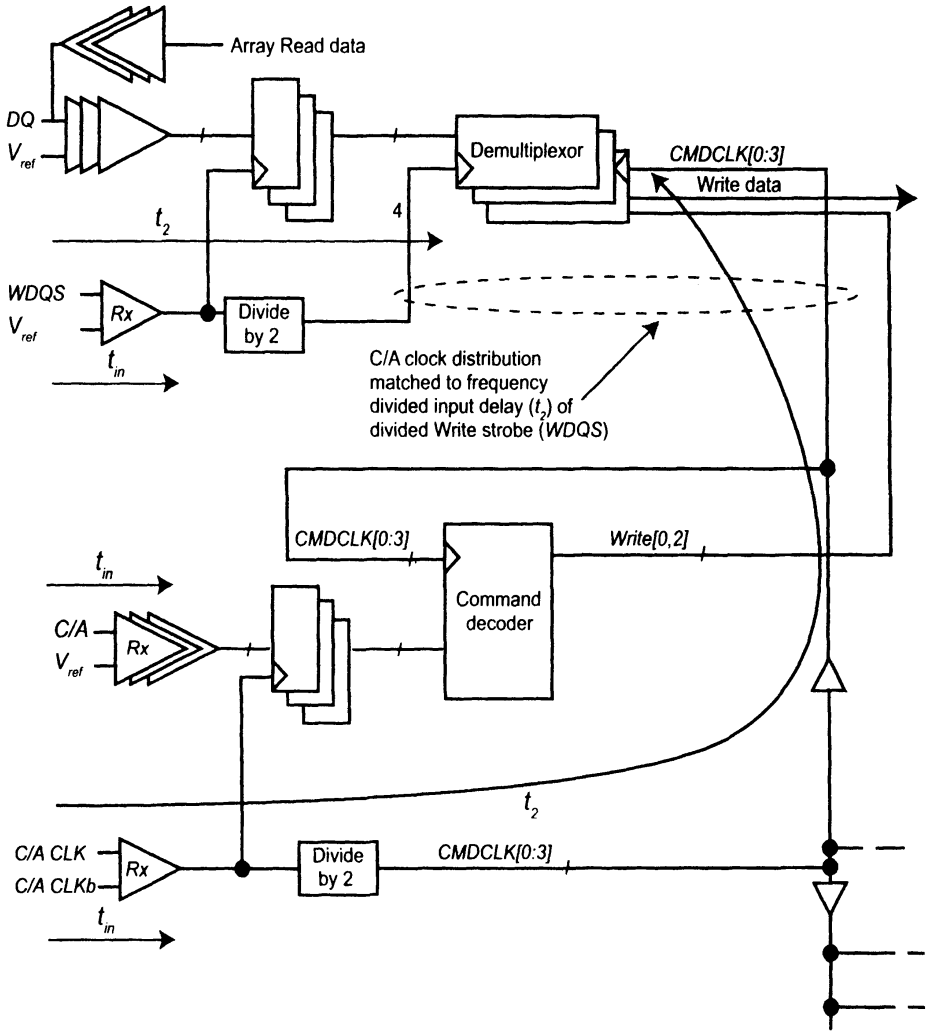


Figure 12.27 Relationship between data input path and command decode path.

What we are trying to achieve through all of this delay matching is to predict the arrival time of data that is captured in the source-synchronous *WDQS* timing domain relative to the Write command received in the *C/A* clock domain based on the predetermined Write latency. One key to the success of this method is delay matching between the *WDQS* capture path, *C/A* capture path and the *C/A CLK*[0:3] distribution path (t_{in}).

An alternative method to consider for retiming the latency timing signals is to reduce our reliance on delay matching of the frequency-divided

distribution paths, indicated by t2 in Figure 12.27. We can do this by including a clock alignment circuit, such as a PLL [20], which uses the input capture path as the reference to the phase detector. This configuration is shown in Figure 12.28. Here, we have less risk in trying to delay match the long interconnects routed from the command clock domain out to the Write data demux circuits with the *WDQS* distribution to the demultiplexor circuits. We choose a PLL for this application because of the higher supply noise rejection ratio when compared to a DLL (Chapter 11). With this configuration, we are still required to delay-match all of the input capture paths but we are not required to try to match the global clock distribution of the command clock with the input delay (labeled t2 in Figure 12.27).

Using a branch of the input capture clock tree as a reference to the phase detector of the PLL forces the branches of the globally distributed command clock tree to align with the frequency divided *WDQS* clock at the demultiplexing circuits. Remember that in the cases we are examining, the *WDQS* is a burst strobe. If *WDQS* were a continuous clock, we could align the Write clock domain with the command clock domain directly. But the scheme shown in Figure 12.28 aligns the distributed command clock with the predicted arrival time of the *WDQS* based on delay matching between all of the input capture paths indicated as t_{in} in Figure 12.28.

The problem with the matched-mode methods just described is that as process scales, device variability becomes a significant problem. Another consideration is that across large physical distances on the die, there can be voltage gradients that will cause further mismatch and clock skew across the die. All of this is exacerbated by higher clock and data frequencies. The static portion of these sources of timing mismatch can be mitigated through training sequences for adjusting Write data timing in the data capture paths and for setting the Write data latency relative to the command decoder timing. These training sequences are mentioned in Chapter 9, but because the training sequences are specific to a particular DRAM specification, it is enough to know that the adjustment of the data capture and data latency timing circuits is primarily based on exchanges of expected data patterns across the data channels and that these timing relationships may need to be retrained periodically. Any necessary retraining of timing relationships causes a great deal of complication in the DRAM specification. However, many training sequences that have been proposed or implemented are only set during device power cycles or reset sequences and do not involve periodic re-training. In either case, clock alignment circuits, such as a PLL, adjust long-term voltage and temperature drifts to help maintain synchronous alignment.

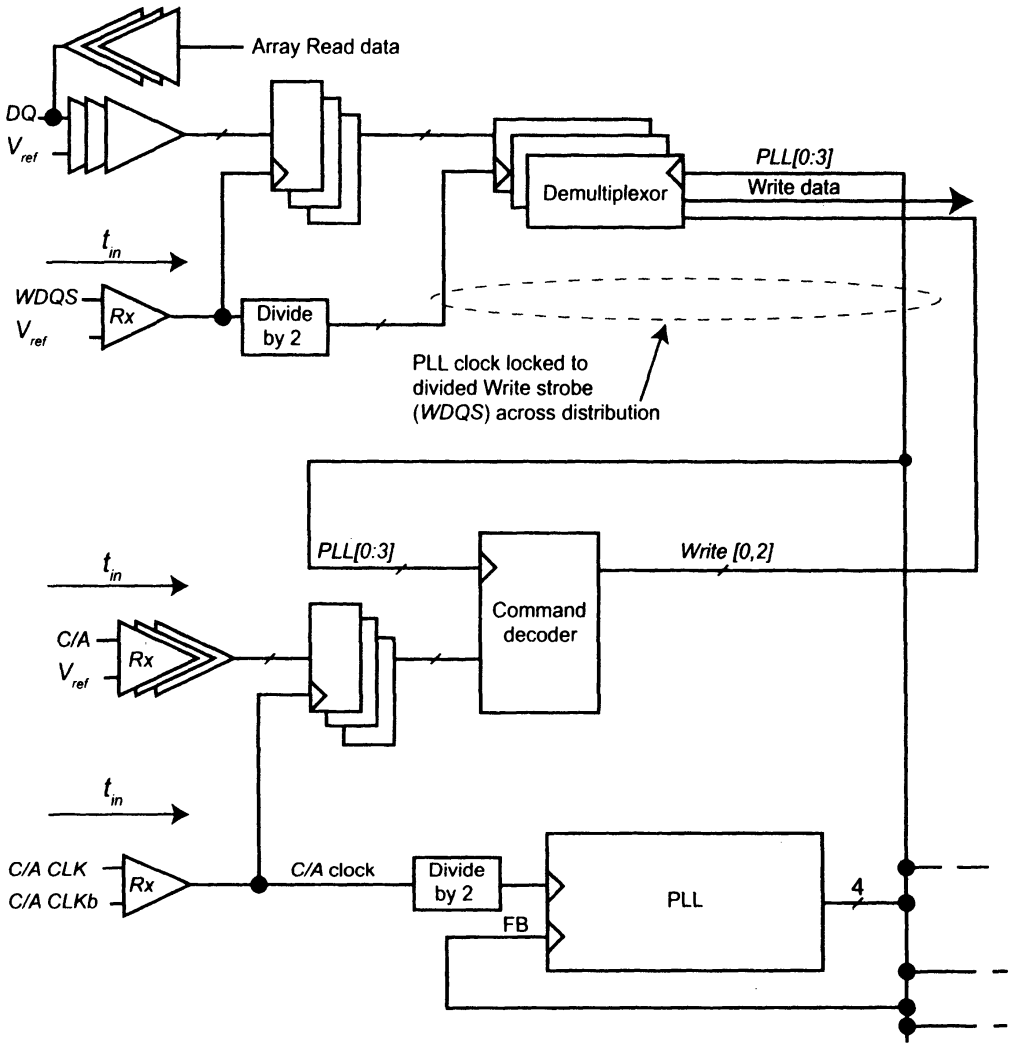


Figure 12.28 PLL solution for Write latency retiming.

12.4.2 Write Data Demultiplexing

Figure 12.24 (shown in the previous section and repeated in Figure 12.29) is a top-level diagram for a single bit slice showing how the Write latency timing signals, *Write[0,2]* converge with the arrival of the serial data stream at the demultiplexing circuits. The *Write[0,2]* signals are used to ensure that data captured in the critical, high-speed capture path is correctly retired to the DRAM array. In this figure, we see that data enters the device as a serial stream at each data *I/O*. Because we are looking at a high-performance DDR data path, the capture latch is implemented as a double-edge latch consisting of two capture latches: one latch for rising strobe edge data, *Dr*, and one latch for falling strobe edge data, *Df*. Following high-speed data capture, each of the data outputs of the capture latches is now

one half of the input data rate. In our example data path, there is a frequency-divided, four-phase clock. This clock is distributed to all of the data bits associated with the WDQS signal and operates at one quarter of the data rate. Each phase of the clock is used to capture either the rising edge data from the capture latches (Phases 0 and 2) or the falling edge data from the capture latches (Phases 1 and 3). At this set of latches, the data rate is down to one quarter of the of the original data rate at the DRAM inputs. The demultiplexing circuit then assembles the bits into a parallel set of data that is driven, along with the *wDatLat* signal, to the peripheral array logic, where it is latched and held until the data is driven on the *I/O* lines of the DRAM array logic.

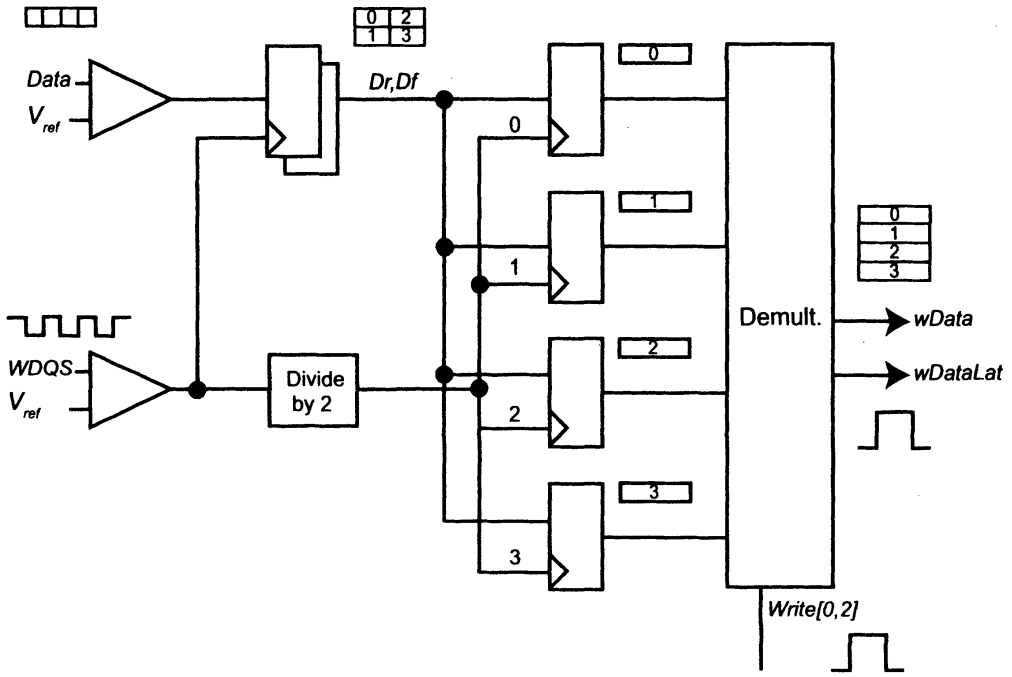


Figure 12.29 Input data bit slice.

Figure 12.29 is a simplification of some of the design choices that must be made in order to implement a robust solution for capturing data across a physically wide data path. Fortunately, one provision that is part of the specification for high-performance DRAM is that multiple data strobes are available for wide data paths. For instance, a 16-bit wide device would have two data strobes: one for each byte [18]. The data strobes are generally placed at the center of the data word. Depending on the pad spacing, there is a relatively long routing distance from the center of a group of data pads out to the end of the data path pad row. It is this routing distance, combined with the maximum operational data rate, that can determine the circuit topology for the control signals used to time the demultiplexing circuits.

There are two primary choices when determining the optimum circuit topology for demultiplexing across a data word. Figure 12.30 illustrates these two choices. The first example is shown in Figure 12.30a. Here, the control and timing logic is centered within the data byte with the timing and control signals globally distributed to each nibble on either side of the control logic block. If the distribution delay of the timing and data select signals (*DSel*) is less than the column cycle time, then this is a viable configuration. Notice that the latency timing signals (*Write[0,2]*), which are inputs to the control logic, are timed to coincide with the input *WDQS*. The input data is clocked into each of the demultiplexor circuits by the distributed *WDQS* signal. For each burst, the control logic at the center of the data byte drives control and timing signals to each of the demultiplexor circuits concurrently. For this case, the demultiplexor circuits operate independently when capturing data but the timing of the signals distributed from the centralized control logic block retimes the data driven from each of the demultiplexor circuits.

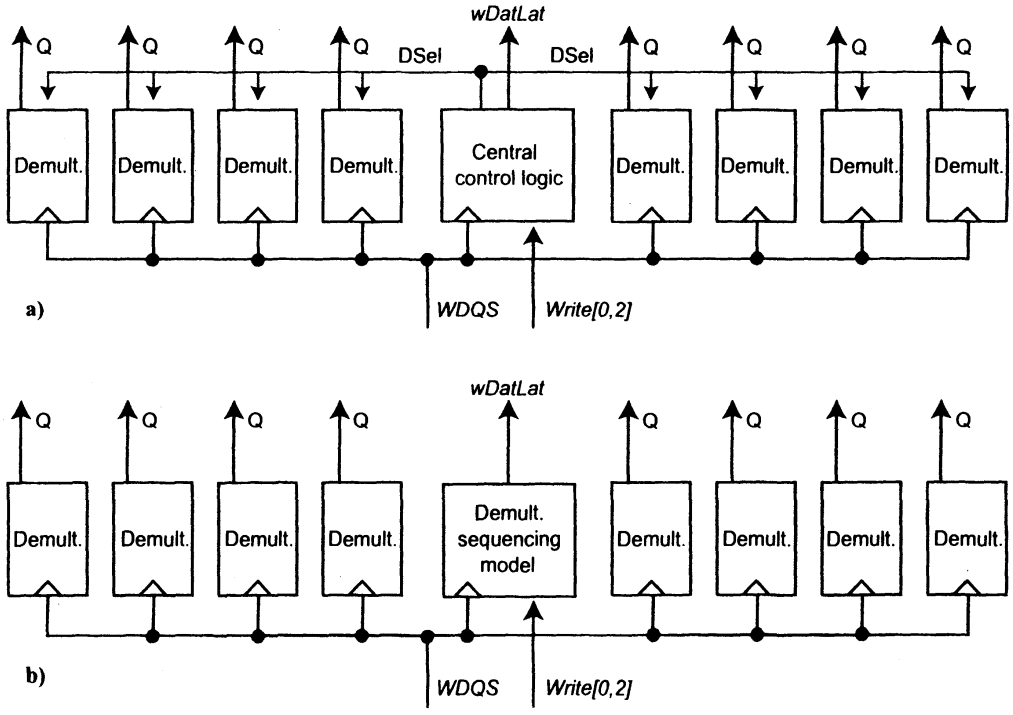


Figure 12.30 Circuit topology choices for Write data de-serialization.

Alternatively, in Figure 12.30b, the control logic is localized for each bit in the byte. There is a central block of logic that times the *wDatLat* signal, which is bundled with the Write data and used for latching and retiming the Write data in the peripheral array logic. The centralized logic in this case is an additional demultiplexing circuit that is used as a model of the other demultiplexors in order to time the arrival of data. There is delay margin

built into the output timing of *wDatLat* from this circuit to compensate for any timing skew between each bit within the byte. Part of the compensation must account for t_{DQSS}. Typically, t_{DQSS} is +/–0.5 of a bit time (often referred to as *unit interval*). If t_{DQSS} violates this specification, then the latency timing circuits may align to the incorrect starting bit of a burst. The t_{DQSS} compensation is handled by the timing domain crossing that is indicated by the convergence of the *Write[0,2]* signals with the data strobe distribution generated in the *WDQS* timing domain. By tracking the status of the demultiplexing circuits relative to the arrival of the *Write[0,2]* signals, t_{DQSS} can be compensated within the demultiplexor tracking circuits. The primary difference between the circuit topologies in Figure 12.30 is that the centralized logic in Figure 12.30a generates all of the timing and control signals for each of the demultiplexor circuits and also generates the bundled latching signal *wDatLat*. Whereas, in Figure 12.30 (b), the control and timing logic is repeated within each of the demultiplexor circuits with only the latency control logic and *wDatLat* logic contained in the centralized circuit block. The circuit of Figure 12.30b can operate at a higher frequency because there are no globally distributed control signals, but this circuit suffers from increased power and area relative to the circuit of Figure 12.30a.

Now that we have examined some of the details of Write latency tracking and the global data capture topology of the demultiplexor circuits, we turn our attention to the design of an individual demultiplexor circuit. To illustrate the demultiplexor operation, a simple flip-flop based circuit is shown in Figure 12.31. The circuit in Figure 12.31 is an example of only part of what is required for a full demultiplexor solution. We see from the timing diagram in the figure that the purpose of the demultiplexor circuit is to successively expand the data valid window at each pipeline stage until the data can be safely transferred to the array logic. Because the contents of the flip-flops are over-written on successive Write cycles, this solution is only viable if data is transferred to a retiming circuit within three-bit times (between WDS[3] and WDS[2] in this example) using the *wDatLat* signal. Remember from the previous discussion that *wDatLat* results from the convergence of the Write latency timing signals sent from the command decode clock domain, with the timing signals generated by the central control circuit in the Write strobe timing domain.

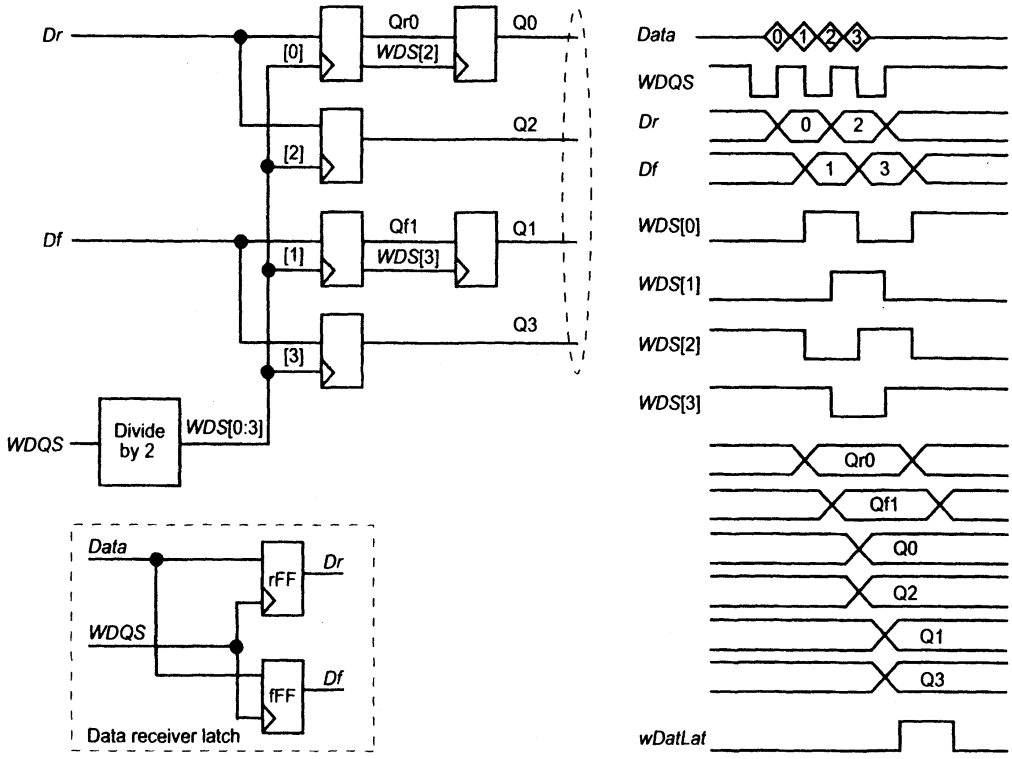


Figure 12.31 Flip-flop-based Write data demultiplexor.

One problem that is not obvious from Figure 12.31 is that with a divided Write strobe, there are two possible relationships between the receipt of a Write command and the output state of the Write strobe frequency divider. The timing diagram in Figure 12.32 illustrates this case. This diagram is taken from a GDDR DRAM data sheet and shows the case where two Write commands are issued with an extra command cycle inserted between the minimum possible Write cycle time. This Write command sequence is often referred to as a *nonconsecutive Write operation.* First, consider that the alignment between the WDS[0:3] divided strobes is the same as that shown in Figure 12.31. Then, when the extra command cycle shown in Figure 12.32 is inserted between the Write commands, an extra toggle occurs on the *WDQS* signal, and invalid data is loaded into the data capture latches.

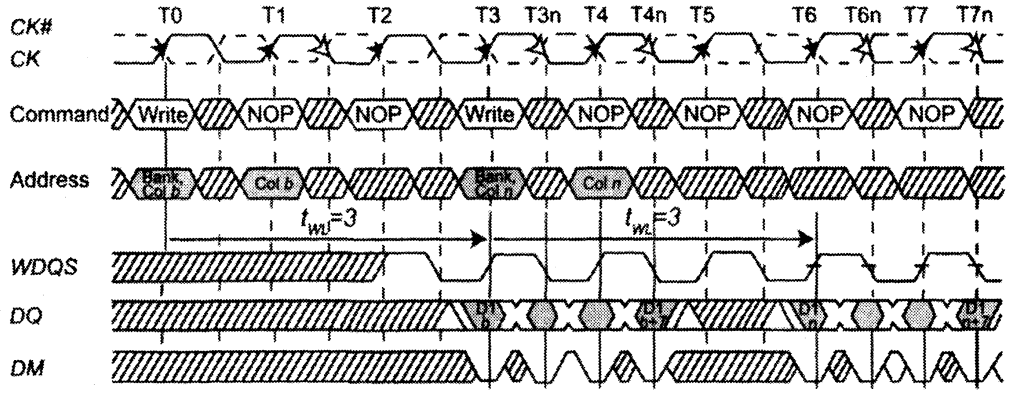


Figure 12.32 Nonconsecutive Write commands.

Figure 12.33 shows how the divided strobe alignment changes between the first set of Write data and the second set of Write data. Focusing on the WDS[0:3] signals, for the first Write, WDS[0] aligns with the first rising edge of *WDQS* for the data burst (burst of four, for this example). Because of the extra cycle inserted between the Write commands, WDS[2] aligns with the first rising edge of the *WDQS* for the second data burst. Because the divided strobe signals are operating at one half of the frequency of command clock, the command to *WDQS* alignment can assume two different states. In order to correctly track Write data latency, and drive the correct data out of the demultiplexor circuit, we need a method to detect the two possible command-to-data alignment states for a divided *WDQS*. This is the reason that the Write latency timing signals are paired as *Write[0,2]*.

Following a system reset or power cycle, the *WDQS* divider and the Write latency control logic in the command decoder are reset to a known condition. Depending on the logic or clocking implementation at the Write decoder and demultiplexor logic, one of the latency timing signals fires for the first Write command and establishes an alignment relationship between the command decoder logic and the output of the *WDQS* divider for the first Write command. For all subsequent Writes, the correct *Write[0,2]* signal will align accordingly with the WDS[0,2] signals to correctly time the output of the demultiplexor circuit. We should note that if toggles occur on *WDQS* that do not correspond to a Write command, for instance, because of overshoot or undershoot on *WDQS* due to signal integrity issues, then untracked toggles may occur on WDS[0:3]. Under these conditions, the demultiplexor circuits must be designed to track the correct *Write[0,2]* to WDS[0,2] alignment, regardless of starting conditions. This is a specific problem the designer must evaluate, yet solutions to this problem are not fundamental to this discussion.

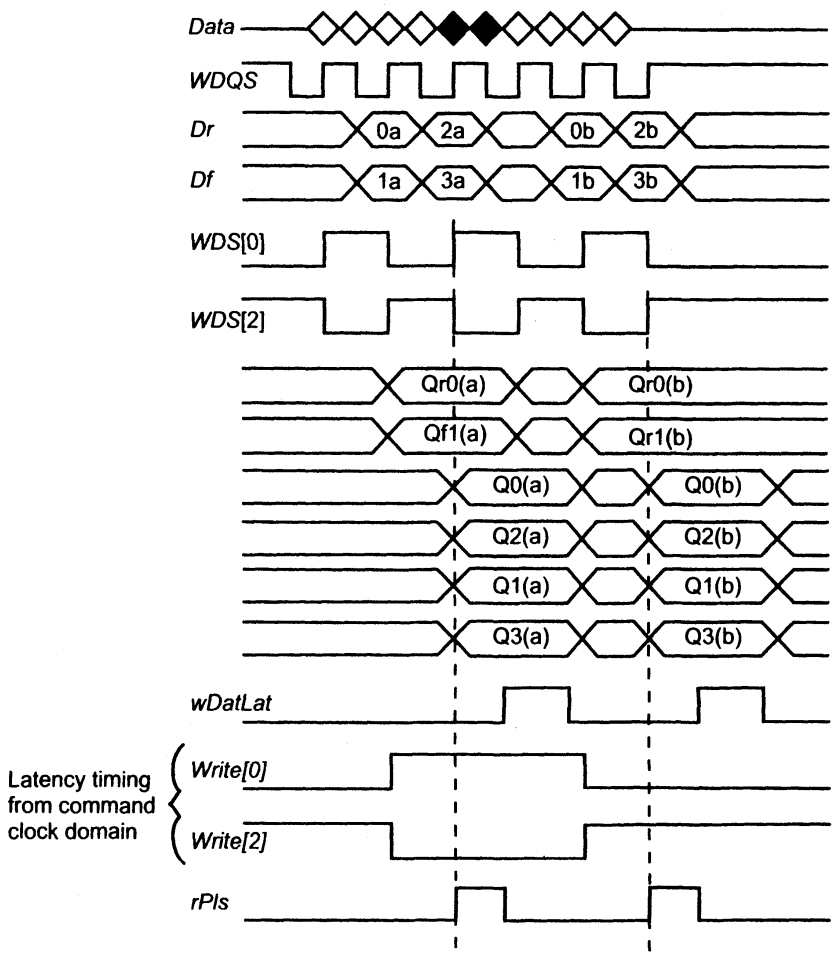


Figure 12.33 Divided Write strobe alignment for nonconsecutive Write commands.

Additional signals are illustrated in Figure 12.33 that help us examine a simplified solution to demultiplexing for the divided *WDQS*. This discussion is further aided by the circuit diagram of Figure 12.34 [19]. The first feature to notice is the inset to Figure 12.34. Here, we show a modified data latch circuit that includes both rising edge latches (*rFF*) and falling edge latches (*fFF*). Of course, the representation as flip-flops is only for illustrative purposes; in practice, high-performance latch circuit designs are used (Chapter 9). The outputs of this latch design are represented as *Dr* and *Df* in the timing diagram of Figure 12.34. The performance of the latch circuit must allow a data transfer to occur between the rising edge and falling edge of *WDQS* at the highest data rate and with the maximum allowed *WDQS* duty-cycle distortion. The resulting output of the latch circuit in the inset of Figure 12.34 is alignment of *Dr* and *Df* following each falling edge of *WDQS*. Continuing with the example of Figure 12.34, the operation of the demultiplexor circuit is similar to that of the circuit in Figure 12.31 except, in this example, the *Write[0,2]* signals are used to track the alignment of

Write commands with the WDS[0,2] signals to mux the correct data into the demultiplexor circuit path. The block labeled "Pls" represents centralized logic to control the timing of the outputs Q0-3 of the circuit to align the data for transfer to the peripheral array logic. The *wDatLat* signal is used to accommodate the data transfer to the peripheral array logic. By examining the timing diagram in Figure 12.33, we see that the second set of Write data aligns to the complementary *WDS* signals relative to the first Write command because of the nonconsecutive Write condition.

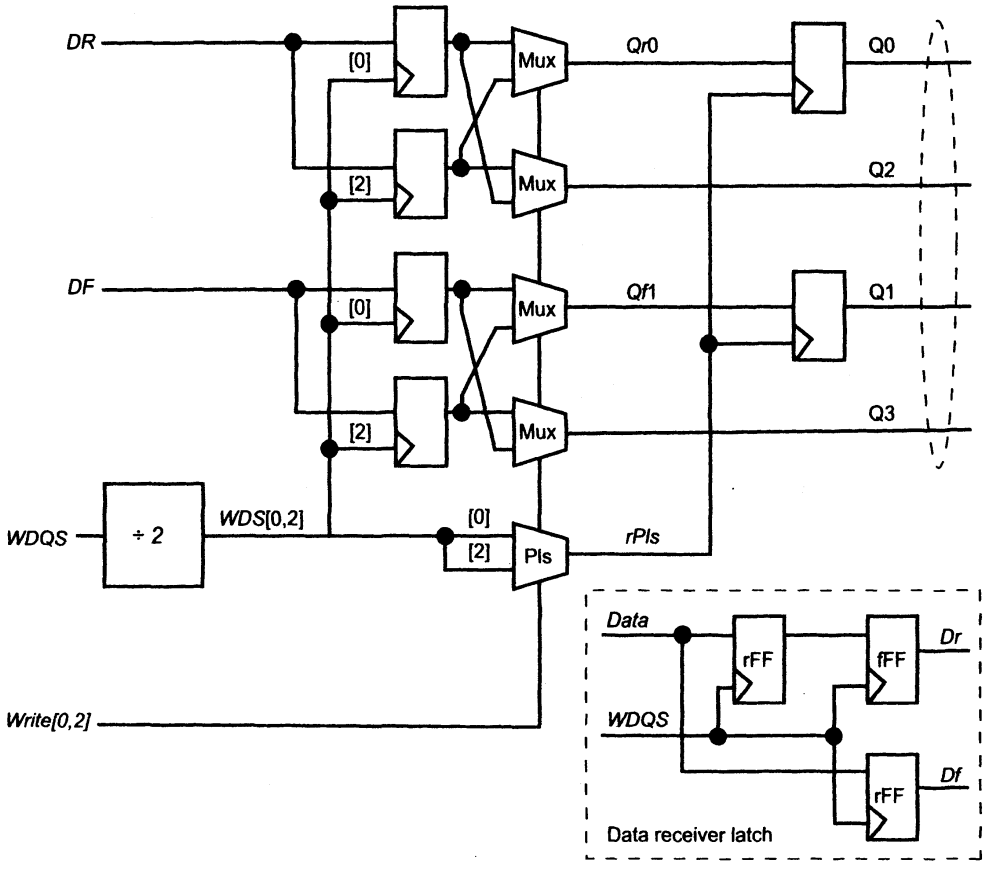


Figure 12.34 Circuit diagram for divided WDQS demultiplexor.

In this section, we have examined some of the fundamental problems for tracking latency and demultiplexing Write data. Two fundamental problems emerged in this discussion. First, that Write latency is specified relative to the command clock domain, and, as such, requires designers to employ a synchronization method to transfer associated timing information between the command clock and *WDQS* timing domains. And, second, that the serial Write data on the wide *I/O* data bus must be demultiplexed and retimed for each individual bit slice before it is transferred to the peripheral array logic.

Keep in mind that much of the preceding discussion may be oversimplified in some cases. This is necessary so as not to obfuscate the fundamental

problems encountered by the DRAM logic designer when executing a full Write data capture design solution. As was stated at the beginning of this chapter, DRAM peripheral logic design is as much an art form as it is a scientific exercise. In the next section, we visit the Read data logic path. The problems encountered in the Read data path have many similarities to those of the Write data path. Similar to Write data, Read data access delay is specified with a deterministic latency relative to the command clock, and there is also a data format conversion (parallel-to-serial, in this case) necessary for compatibility with the array data path and output data rates.

12.5 READ DATA LATENCY TIMING AND DATA MULTIPLEXING

Read data accesses, similar to Write data accesses, are composed of two major logical operations: latency timing and data formatting. We saw that Write latency timing is controlled by transferring timing and synchronization signals from the command clock domain to the *WDQS* timing domain. Write data is also reformatted from a serial data stream to parallel data through the demultiplexor circuits. Write data reformatting is necessary so that the *I/O* data rate is bandwidth matched to accommodate the array column cycle time. Stated another way, a 2 Gb/s serial data stream matched to a 4 ns column cycle time would require an array data bus width of 8 bits per data *I/O*. This same bandwidth matching occurs for Read data where Read data on the array bus cycles at 250 MHz with an 8n Prefetch providing enough data bandwidth to feed *I/O* circuits operating at 2 Gb/s.

Read data is managed much the same way as Write data except that with Read data the ordering of the array access and *I/O* transfer are reversed. For Read commands, the array is immediately accessed and data is transferred from the HFFs, in the peripheral array logic, to the Read data path of the peripheral logic interface. On the other hand, for Write commands, Write data is captured in the input circuit path, then assembled in the demultiplexer circuits. Write data is then retired to the array with an array access. In the Read data path, there is a multiplexor circuit that is the converse of the demultiplexor circuit in the Write data path. Just as the Write demultiplexor is often referred to as the *Write pipes*, the Read multiplexor circuits are often referred to as *Read pipes*. Refer to Figure 12.35 for a simple block diagram showing the relationship of the peripheral logic interface sandwiched between the array logic and the input and output circuit paths.

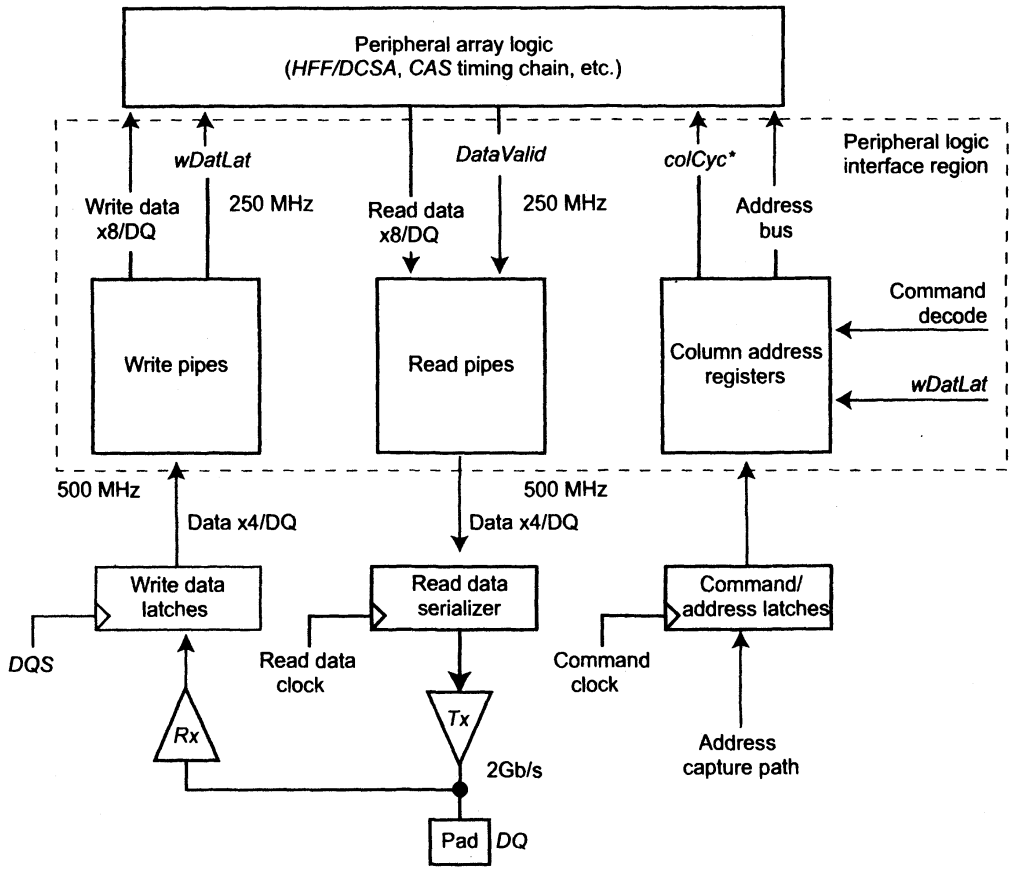


Figure 12.35 Peripheral logic interface Read and Write data paths.

Figure 12.35 illustrates the relationship between the Read and Write pipes and how the data rate of the *I/O* circuit paths translate to array column cycle time. Another feature shown in this figure is separated Read and Write busses in the peripheral logic interface region. The separated Read/Write busses allow concurrency for Reads and Writes, which may be necessary for tight Read-to-Write or Write-to-Read data timing. Also consider that the Read data array accesses are controlled by the timing of the command clock domain, while Write data array accesses are timed to the *WDQS* timing domain. If we consider t_{DQSS} combined with variations in address access delay for Reads (t_{AA}), there could be overlap on a shared Read data and Write data bus in the peripheral logic interface region. There will be a timing diagram later in this discussion that will better exemplify this possibility. Another notable feature of Figure 12.35 is the address bus with the bundled signal, *colCyc**. Whenever a Read command is received, the address and *colCyc** signal are immediately driven to the peripheral array logic to get the lowest possible array access latency. For Write data, the *colCyc** signal is fired based on the *wDatLat* signal that comes at the end of the data demultiplexor operation (Section 12.4). Recall that Read data accesses

are initiated as quickly as possible to get the lowest possible array access latency, while Write data accesses require that the column address is held until all of the data of a burst is fully captured in the demultiplexor circuits.

Let us look at an example of a Write-to-Read access to see how the array column cycle time is maintained for data accesses. Figure 12.36 is a timing diagram showing a series of column accesses. In this example, there is a Write command followed by a Read command, which results in a minimum array column address cycle time at the peripheral array logic. We need to be aware that a *colCyc** initiated by the *wDatLat* signal must have timing similar to a *colCyc** initiated for a Read command originating from the command decoder. By having similar timing relative to the launching event, we can maintain constant column cycle time between Read and Write data accesses. In the timing diagram, Read latency (CL) is four command clock periods, while Write latency (WL) is CL–1=3. At the end of the Write data burst, the *wDatLat* signal transitions. As outlined in Section 12.4, the *wDatLat* serves two functions. The first is to retime data in a set of latches in the peripheral array logic. The second is to enable the firing of the *colCyc** signal to cycle data accesses from the array. Write data is then overdriven through the sense amps and, ultimately, to the Mbit.

Again referring to Figure 12.36, a Read command is issued at the end of Write data. For this example, the Read command occurs two command clock periods, which is one column cycle period, from the end of the Write data. We see that the tight timing of the Read command relative to the end of Write data results in close to a minimum column cycle period transition on the *colCyc** signal between the Write and Read accesses (labeled t_{WTR} in the figure). The chip architect must decide between a shared data bus or two unidirectional busses for interconnect between the peripheral logic interface region and the peripheral array logic for Read and Write data. Variables such as bus loading, interconnect routing, column cycle time, data throughput, Read and Write access concurrency, die area, and any other performance specification-related issues must be considered when making this decision.

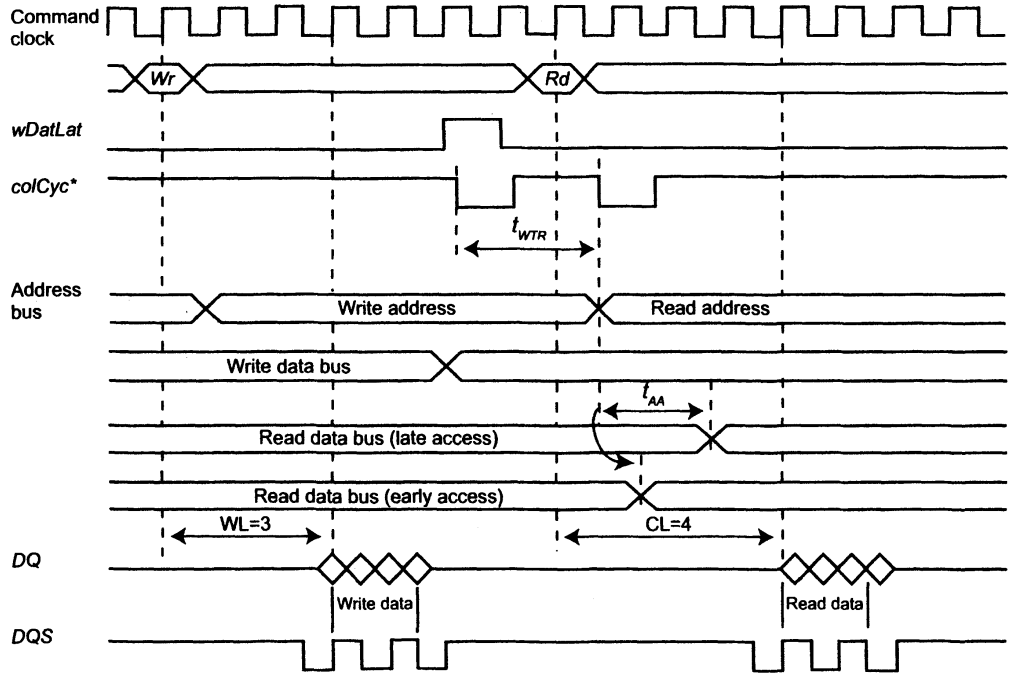


Figure 12.36 Timing diagram for internal data bus utilization.

Another important timing parameter for the peripheral logic interface design is *address access latency,* labeled t_{AA} in Figure 12.36. This delay is measured from the firing of *colCyc** to the transition of data at the output of the peripheral array logic. There are two sets of Read data shown in Figure 12.36. The first set of data, labeled "Read data bus (late access)" shows address access timing where data arrives late in the access cycle but early enough to satisfy the programmed value for CL. The second set of data, labeled "Read data bus (early access)" shows address access timing that occurs early in the access cycle, well ahead of the programmed CL delay. Variations in t_{AA} are fundamental to the operation of the DRAM array. Array access delay, measured from the initiation of a change in column address, depends solely on timing models formed by delay chains that are used to reflect variations in on-die process, voltage, and temperature conditions. For this reason, array access delay varies by a wide range and, because it can vary by multiple clock cycles, Read data cannot be synchronized in the peripheral array logic but is instead synchronized in the output circuit path. This means that the Read data path design must take into account the minimum possible access delay as well as the maximum possible access delay in order to maintain continuous data throughput. The implication is that for minimum access delay, any data driven into the data path must be held until it is driven through the synchronous portion of the output circuit path, based on the programmed CL value. At the same time, for max-

imum array access delay, the data must arrive at the synchronization circuits ahead of the expiration of the programmed CL value.

For the remainder of this section, we will examine the two main components of Read data timing managed within the peripheral logic interface. First, we look at the purpose of the Read pipes as well as the performance characteristics necessary for the Read pipe circuit block. After looking at the Read pipe circuitry, we examine synchronous Read latency tracking for a high-performance DRAM and look at an example circuit used for tracking Read latency over various operating conditions.

12.5.1 Read Data FIFO

The Read pipe is functionally a FIFO. Its purpose is to absorb array timing variation (t_{AA}) and maintain data throughput between the array data path and the synchronous portion of the output circuit path. For the remainder of this chapter, we will refer to the Read pipe as the *Read data FIFO* or *Read FIFO*. Looking back at Figure 12.36, we see how variations in t_{AA}, relative to CL, cause the Read data to have an indeterminable timing relationship to the synchronous Read timing control. There are several possible design variations for the Read data FIFO. For mid-to-low frequency DRAM, there is very little depth required for the Read data FIFO. Often, the CL value is low and the burst granularity is high to ensure that there is sufficient array timing margin to achieve high yield for commodity level devices. However, for high-performance DRAM, the Read data FIFO requires much more attention. With lower column cycle times and higher data rates, the storage and timing performance of the Read data FIFO is more important. In this section, we look at a design example for a Read data FIFO. This examination considers aspects of the DRAM specification to determine the performance parameters necessary for a successful design. We do not study a specific circuit design for the FIFO but instead outline the performance issues associated with the FIFO design and suggest some alternatives for the FIFO architecture.

Figure 12.37 is a block diagram showing the position of the Read data FIFO between the peripheral array logic and the output data serializer. In this example, a four-phase clock drives the output serializer. This clock is frequency-divided and quarter-aligned and is discussed in Chapter 11. Each rising edge of the four-phase clock is separated by a single bit time so that output data transitions correspond to each rising edge of the output data clock. We will see that using a divided, four-phase clock in the output circuit path is an important consideration for the Read data FIFO design.

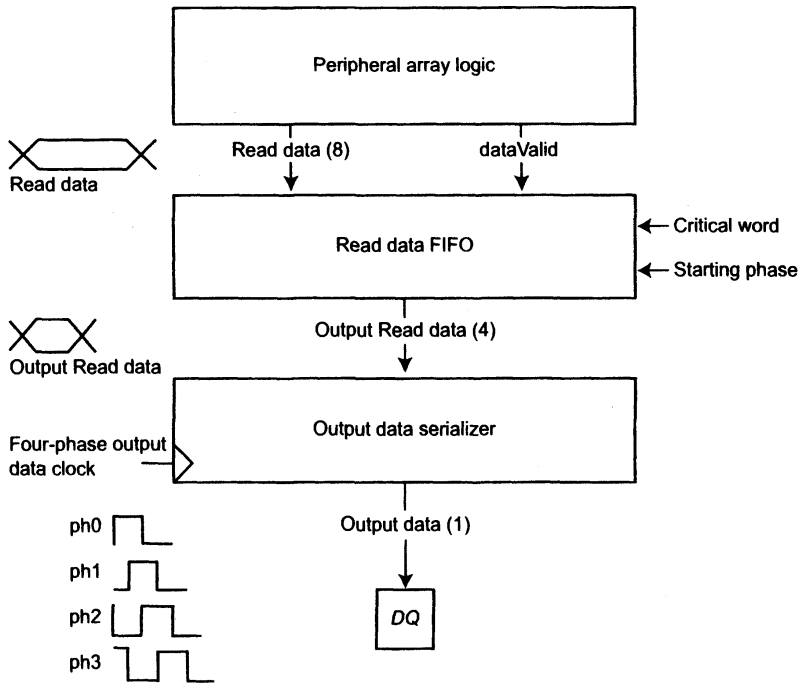


Figure 12.37 Read data FIFO reference.

In our example, the array data path delivers an eight-bit data Prefetch for each DQ. The eight-bit Prefetch is delivered to the Read data FIFO bundled with the *dataValid* signal, which is discussed in Section 12.3.3 and is shown in Figure 12.22. The data is then reduced to a four-bit width through the Read data FIFO and delivered to the data serializer. The output data serializer reduces the data from four bits down to singular output data as well as retimes the data to the output data clock. There are a few functional details about the FIFO that need to be clarified. One is that the Read data FIFO must be capable of multiplexing and sequencing the correct four bits of data through the output data path. For some DRAM specifications, there is what is called *critical word first*. This feature uses a lower column address bit to indicate which nibble is output first in an 8n burst (a choice between 0–3 or 4–7). This feature is a part of the DDR SDRAM specification and is sometimes referred to as *burst ordering*.

There is another multiplexing function necessary in the Read data FIFO. Most device specifications call for commands to be referenced to the positive transition of the command clock. When the clock is frequency divided, there are two phases of the output data clock referenced by the positive edge of the command clock. For our example, ph0 and ph2 are considered proxies for a positive transition on the command clock. When data is routed from the peripheral array logic to the Read data FIFO, there is no output clock phase information included. A combination of CL and Read

command alignment to the four-phase output data clock results in two possible data-to-clock alignments. Figure 12.38 shows an example of a nonconsecutive Read access where the first Read forces the first data bit of an access to align to output data clock, ph2 and the second Read forces the first data bit to align to output data clock, ph0. The remaining bits are also multiplexed to the correct clock phase so that the phase order for the first access is 2,3,0,1 and the second output phase order is 0,1,2,3.

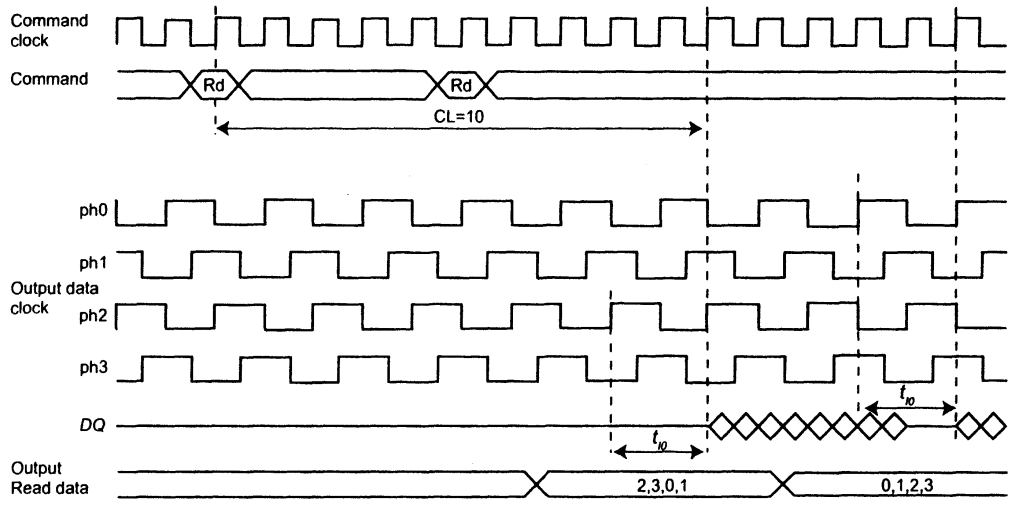


Figure 12.38 Nonconsecutive Read access and multi-phase Read clock.

The starting phase alignment information originates from the Read command latency circuits. These circuits are an extension of the command decoder circuits and are discussed later in this section. The starting phase information is determined by the programmed CL value and the relationship between the command clock and the multi-phase output data clock. Because the Read data FIFO only carries array data, another FIFO is necessary to feed the starting phase information to the multiplexing circuits in the Read data FIFO. Figure 12.39 shows the secondary FIFO labeled "orthogonal" FIFO. It is so named to aid in visualizing this circuit as providing a perpendicular control term to the Read data FIFO.

Another notable feature in Figure 12.38 is the timing parameter, t_{IO}. Remember from previous discussions in this text that high-performance DRAM commonly employ clock alignment circuits that are used to align the output data and strobe to the external system clock. To accomplish this feat, an *I/O* delay model is included in the feedback of the clock alignment circuit to advance the output data clock by the delay of the input and output circuit paths (Chapter 5 and Chapter 11). Because the data output clock is advanced relative to the command clock by the asynchronous *I/O* delay, once data is retimed and driven through the data output path, the data appears nominally aligned to the command clock at the device *I/O* pins. This configuration is necessary to simplify system design so that there is a

common timing reference for all DRAMs in a system. The side effects of advancing the timing of the output clock for on-die Read latency tracking will become more apparent later in the discussion about Read latency (CL) tracking.

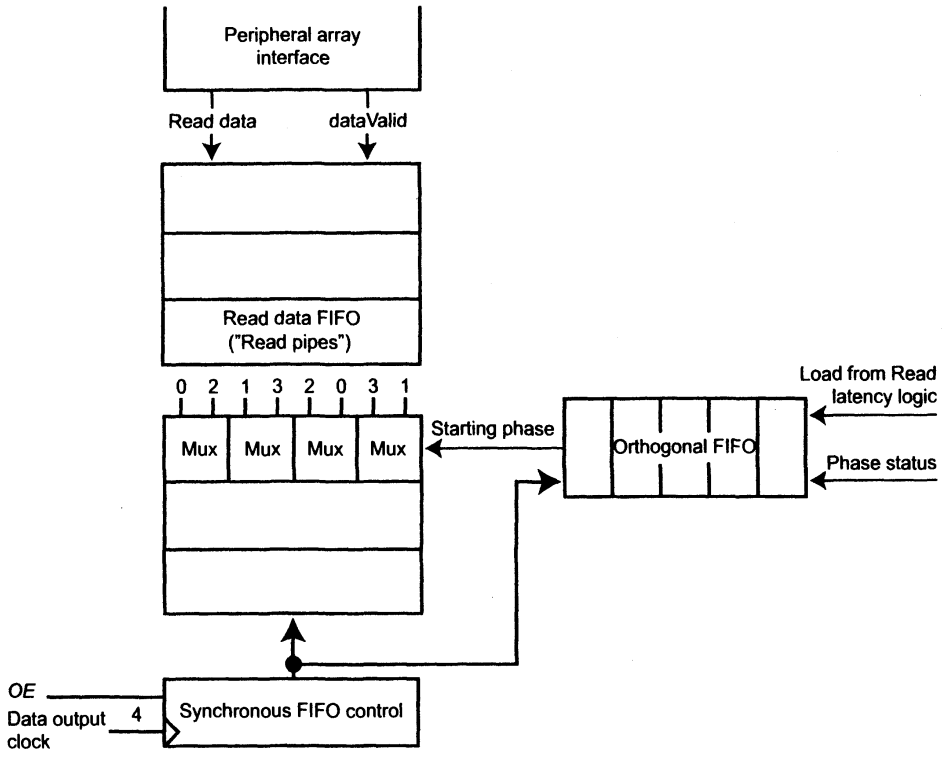


Figure 12.39 Data alignment to multi-phase output clock through the Read data FIFO.

To get a better idea of how the FIFO must perform in the DRAM Read data path, let's examine the diagram in Figure 12.40. Here, we see a continuous stream of consecutive Read commands. The boxes represent relative percentages of the overall CL time for each operation of a Read command. Notice that for the "Fast" operating condition, the time allotted for the FIFO is the majority of the CL delay. Because the CL specification is based on worst-case timing conditions, if the device happens to perform better than the data sheet specification indicated for CL, there is no way for the system to take advantage of the improved array performance. The reason for this is that there may be several DRAMs in the system that perform worse or better. Therefore, the FIFO must match the throughput and latency of the array accesses with the expected data arrival based on the programmed CL value for all of the devices. The device data sheet specifies the minimum and maximum value for CL. The value of CL is based on the acceptable worst-case performance of any one DRAM in the system. For the "Slow" operating condition, the FIFO period is much less than t_{AA} and, for this case, we are concerned with the length of delay through the FIFO circuitry (t_{FL}).

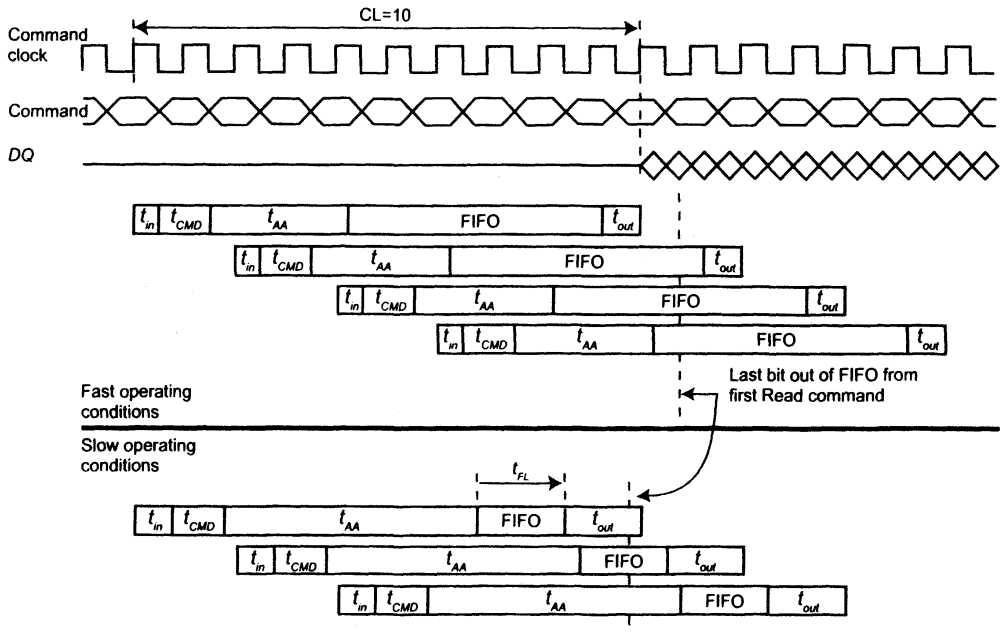


Figure 12.40 Consecutive Read command timing.

Figure 12.40 is interesting in the way that it illustrates the asynchronous nature of DRAM operation. Another point, which will be made again in our discussion about Read latency, is the way in which the clock alignment circuit operates in the DRAM and its effect on the FIFO period. Notice that in the example for the fast operating corner how the short I/O delay forces the output clock closer to the CL expiration time. At the same time that the output clock pushes out the internal data timing, the input delay and access delay is pulled in due to the faster operating conditions. This forces data from the array to be output closer to the Read command, making the overall FIFO period much longer. The longer FIFO period forces a deeper FIFO circuit design. Conversely, in the slow operating case, the clock alignment circuit pulls in the output clock timing relative to the CL expiration time due to the longer *I/O* delay. At the same time, the slower operating conditions pushes the timing of the data access out from the Read command making the FIFO period much shorter and constraining the FIFO design to a lower forward latency (t_{FL}) requirement.

When designing the Read data FIFO, the first step is to determine the required depth of the FIFO. The depth of the FIFO determines the throughput capability of the outputs to ensure that the output circuits are neither overflowing with data nor starved for data. There are two critical boundaries that must be considered. The first is to make sure that when data is read from the array, and the internal Read latency count has not expired, that the Read data can be stored until it is driven through the output circuit path.

The peripheral array logic does not have storage available for data and, furthermore, if array accesses are purposely delayed, then a violation of minimum column cycle time will occur for subsequent Reads during consecutive burst Read operations. The second condition is that the delay through the Read data FIFO is not large enough to significantly impact access delay and, therefore, prevent the device from meeting the minimum Read latency specified by the data sheet. For this case, the delay through the FIFO needs to be minimized.

With these two criteria in mind, we consider an example for a high-performance device. This example is taken from the specifications for a GDDR3 high-performance graphics DRAM. The maximum data rate is 1.6 GT/s, CL=10 and *BL*=4. The example is illustrated through a simple, step-by-step calculation procedure that shows some of the necessary characteristics of the Read data FIFO when evaluating circuit design options for the FIFO implementation:

1. First, because we need to deliver four bits of data for each column cycle, we know that the column cycle time will be 2.5ns. Next, the serializer circuit requires 4 bits of data at its input—one bit to align with each phase of the Read data output clock. Therefore, the cycle time for each bit at the output of the FIFO is:

$$t_{FC} = \frac{1}{\frac{DR}{4}} = 2.5 \text{ ns} \tag{12.5}$$

   where DR=1.6GT/s.

2. Next, we need to determine the maximum input-to-output delay (tFL_{max}), otherwise known as *forward latency*, allowed through the FIFO in order to meet the minimum programmed CL value:
   6ns > t_{AA} > 3.5ns array access delay (max., min.)
   t_{CMD} = 1 ns Command to *colCyc* delay
   t_{SUS} = 0.5 ns Data setup time requirement at serializer input relative to the Read data output clock
   2.8ns > t_{IO} > 1.5 ns asynchronous *I/O* delay (max., min.)

$$^{tFL}max = CL \cdot {}^{t}CK - t_{CMD} - {}^{t}AA - {}^{t}SUS - {}^{t}IO$$

$$tFL_{max} = 2.2ns \tag{12.6}$$

The values for all of the timing parameters (except t_{CK}) in the t_{FL} calculation are for the worst-case operating corner. The value t_{CK} is the minimum period for the command clock frequency. The definition for t_{AA} and t_{IO} show the wide range of delay over which these timing variables can vary.

3. Depending on the chosen circuit architecture, the preceding FIFO design parameters are dependent on the FIFO depth. Given a series of consecutive Read accesses, the FIFO depth is determined by the number of column accesses that can be loaded into the FIFO before the first access in the series is consumed. Read FIFO depth must be calculated using the longest possible latency and shortest t_{AA} and t_{IO} delay. This timing difference will determine the maximum required FIFO depth (*FD*) to maintain constant data throughput. We derive the equation for determining the FIFO depth by looking at the "fast operating" conditions in Figure 12.40. The FIFO depth is based on the peak number of column accesses that can be simultaneously loaded into the FIFO (for DDR *I/O*):

$$FD = \left(\frac{2CL_{max}}{BL} + 1\right) - \frac{tI_{min} + tCMD + tAA_{min} + tO_{min}}{tCCD_{min}}$$

$$FD = 3.6 \tag{12.7}$$

For fractional values of FIFO depth, *FD* must be rounded up to the next integer value. If the CL value is increased relative to the value defined in this example, the FIFO depth would have to be increased accordingly. Likewise, any improvement in delay through the output circuit path can result in increased FIFO depth.

From the preceding calculations, we find that our FIFO design must have a maximum 2.2 ns forward latency, a minimum 2.5 ns/bit cycle time at the output of the FIFO and a depth of 4. However, for increased data rates and increased range of programmable CL values, the FIFO circuit can become much more complicated. This is primarily because conventional circuit designs, particularly at the FIFO output to serializer interface, can have too much overhead and limit performance. For example, if the data rate were to increase to 2.5 GT/s, then, with a four-phase output clock, the per-bit cycle time at the output of the FIFO would be reduced to 1.6 ns.

Circuit implementations for the Read data FIFO can take many forms. For high-performance DRAMs, the FIFO depth can increase greatly depending on the minimum column cycle time. If we consider that mid-per-

formance DRAMs typically have latencies of two or three cycles, with column cycle times of two clock periods, the latency-to-column cycle time ratio is close to 1. However, for high-performance DRAM, such as a GDDR4 device, latencies grow to 10 or 12 cycles and column cycle times are four clock periods leading to latency-to-column cycle time ratios of 2–3. Worse yet, for GDDR3 devices, the column cycle times are 2 and latency is 10 or higher leading to latency-to-column cycle time ratios of 5 or greater. The result is that for high-performance devices there is a requirement for deeper data FIFOs. A traditional pointer-based FIFO can be used for deep FIFOs. The problem with these circuits is that output data muxing requirements can increase the output delay of the FIFO so that data cannot cycle at the output clock cycle time.

Another option is to implement the FIFO as a set of series connected latches controlled by asynchronous protocol controllers [14, 25]. This type of FIFO implementation reduces the FIFO design complexity and can increase performance at the output of the FIFO; but because the FIFO is a series connection of latches, this type of design suffers from higher t_{FL} and, hence, longer latency. This text does not discuss specific FIFO implementations. The designer must choose the implementation based on the calculated requirements.

For high-performance DRAM, synchronization at the output of the FIFO can be complicated by the data rate. There are two primary alternatives: either the circuit design meets the cycle time requirements through specific circuit implementation techniques, or the width of the data path at the FIFO output is increased at the cost of design complexity. Figure 12.41 shows an example of both approaches. The difference between the two approaches is that for the low-cycle time approach, an aggressive circuit implementation using dynamic logic techniques must be considered. For the increased data path width approach, a method for toggling between the data sets must be implemented and the data must be muxed correctly through the FIFO circuits. This muxing would be in addition to, or a modification of, the clock phase based muxing shown in Figure 12.39. Another consideration when increasing the data width is the increased clock loading and complexity of the serialization circuit.

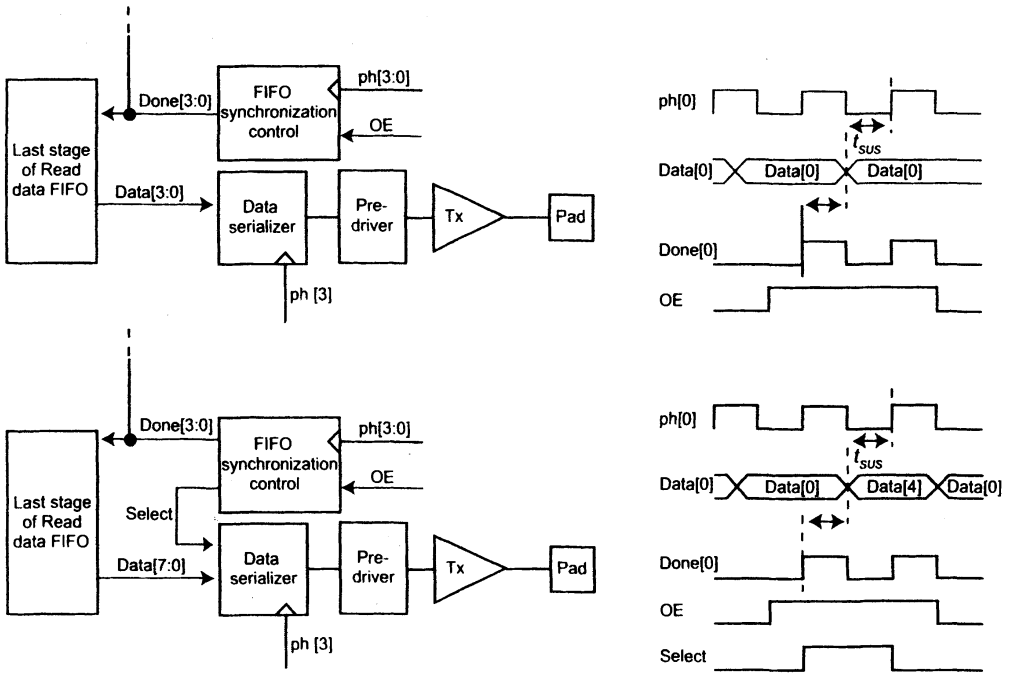


Figure 12.41 FIFO output synchronization solutions for multi-phase output clock.

In the diagrams of Figures 12.39 and 12.41, the signal OE is shown as part of the output synchronization logic for the Read data FIFO. This signal is the data output enable signal (*OE*) and its purpose in the synchronization logic is to indicate that the output circuit path has consumed data from the FIFO. For unidirectional data paths, the *OE* signal is used to indicate the period for which valid output data is driven through the output circuit path. The same can be said about *OE* for bi-directional data channels except the *OE* signal also synchronously enables the data serializer (Chapter 10). Implementation of the synchronous output logic depends on the FIFO circuit implementation. The OE signal is typically shifted through synchronization logic and timed signal transitions (labeled Done in Figure 12.41) are sent back to the Read data FIFO for each bit consumed by the output circuit path. The Read data FIFO responds by switching the data bits on the outputs of the FIFO circuit.

The timing of the *OE* signal is generated by the latency control logic. For high-performance DRAM, CL is deterministic. What this means is that when the value of CL is programmed into the DRAM, the device will deliver Read data exactly CL command clock cycles following a Read command. For memory subsystems, deterministic latency simplifies the data interface so that the memory controller can time the arrival of data from the memory device with a known relationship to the system clock. Tracking Read latency on the DRAM side of the channel sounds relatively simple until we consider that the asynchronous delays through the *I/O* paths can

span multiple clock cycles. Add to this fact that delay variation through the *I/O* paths is directly impacted by process, voltage and temperature, which can cause the delay to vary across clock period boundaries. What we find is that achieving deterministic Read data latency is not a simple matter of counting clock cycles. In the next section, we investigate difficulties related to deterministic Read latency tracking. We also investigate how the inclusion of clock synchronization logic (Chapter 11) on the DRAM simultaneously complicates CL tracking and accommodates synchronization and clock domain crossing between the command clock and the output data clock domains.

12.5.2 Read Latency (CL) Tracking

Thus far, in our discussion of the peripheral logic interface, we have defined Read latency (CL) and how this specification affects the Read data FIFO design. In Chapter 5 and Chapter 11, the output clock synchronization circuit methodologies using either a DLL or PLL circuit were discussed. Part of the discussion of clock alignment touched on the t_{AC} specification. This timing specification dictates the maximum variation that data and any source-synchronous strobes can have relative to the command clock of the DRAM. Ideally, this timing specification is 0 ps. The clock alignment circuit is used to advance the output data clock by the delay through the output circuit path (t_{out}) in order to minimize t_{AC}. At the same time, the reference clock at the input of the phase detector of the clock alignment circuit is delayed by the input distribution delay (t_{in}) of the command and address capture clock. To counteract the asynchronous *I/O* delay, a delay model of the *I/O* circuits is added to the feedback path of the alignment circuit forcing the reference and feedback inputs to a 0-degree phase relationship. Figure 12.42 is a simplified block diagram showing a DLL clock synchronization path for a DDR DRAM.

When the command clock period begins to approach the asynchronous *I/O* delay of the input and output circuit paths, tracking Read latency becomes much more difficult. In early SDRAM, the clock period was much greater than the *I/O* and command decoder delays and the command clock served as both the input clock and output clock. In this situation, the t_{AC} delay was essentially the delay through the output circuit path. The input delay could be ignored since it was masked by the clock period, which was long enough to encompass the input distribution and command decoder delay. For these early devices, data was actually driven from the device at CL-1 and t_{AC} delay allowed enough setup time relative to the clock edge occurring at CL that the system timing budget allowed data to be captured with the command clock at the corresponding CL clock edge. If we fast-for-

ward to DDR DRAM devices and beyond, we find a clock alignment circuit on the DRAM generating an output clock domain separate from the command clock domain. The output clock domain is advanced relative to the command clock by the output circuit path delay, similar to the diagram in Figure 12.42. The clock alignment circuit is used to align the output data to the clock edge corresponding to CL and, at the same time, minimize t_{AC} to predominately short-term components of output clock jitter.

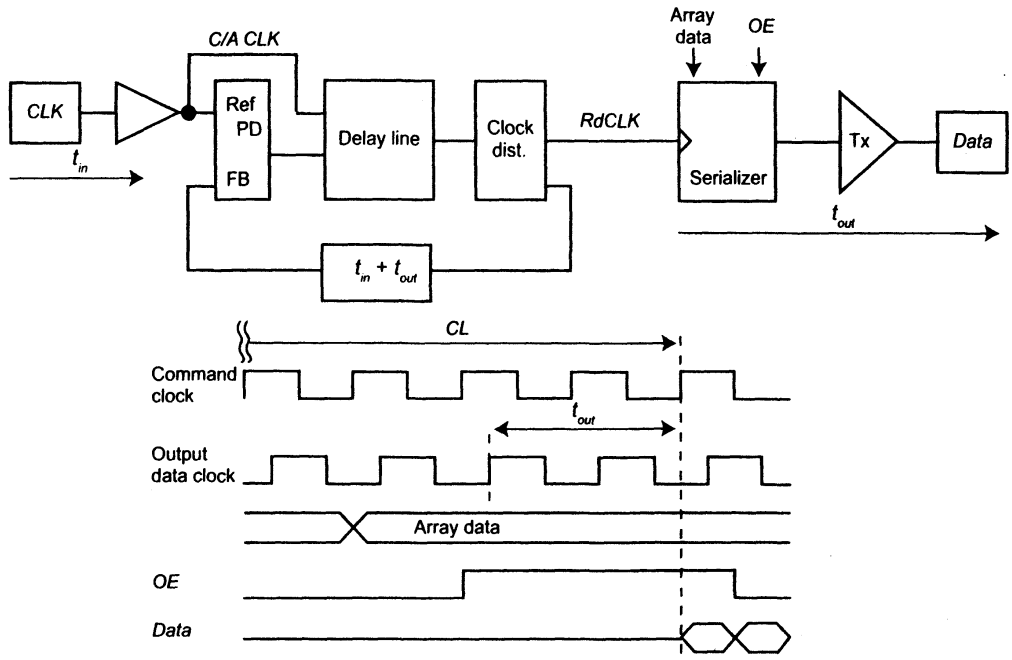


Figure 12.42 Effect of synchronization circuit on output data timing (DLL example).

Considering that for high-performance DRAM, CL is still defined in terms of the command clock, we must consider the implications of transferring timing information, established by the command clock domain, to correctly align with data output timing generated in the output clock domain. Notice in Figure 12.42, we see that CL is defined relative to *C/A CLK* and that the *OE* signal is synchronous with the *RdCLK*.

For high-performance DRAM, the clock period is often less than the input and output circuit path delays. Additionally, as voltage and temperature vary, the output of the clock alignment circuit can vary by a relatively significant amount of time. The preceding statements imply that the phase relationship between the *C/A CLK* and *RdCLK* is indeterminate. Therefore, passing timing signals from the *C/A CLK* to the *RdCLK*, while simultaneously tracking CL relative to the *C/A CLK*, is somewhat complicated. For the remainder of this section, we will examine one method [20] employed on several high-performance DRAM designs that both establishes a determinate relationship between the *C/A CLK* and *RdCLK* domains as well as

tracks the relative delay variations between the two clock domains with long-term changes in voltage and temperature.

The method for CL tracking that we are about to examine can be difficult to explain and, depending on how well it is explained, difficult to understand. To facilitate our understanding of this method, we will break the function of the circuits into two parts. The first part will explain how the relative phase relationship between *C/A CLK* and *RdCLK* is comprehended. By determining the phase relationship between the two clock domains, we can establish a basis for tracking delay variations (resulting in phase alignment changes) that occur between the two clock domains. The second part will examine how variations in the phase relationship between the two clock domains are tracked and, ultimately, how we determine the precise internal *RdCLK* clock cycle from which to fire the *OE* signal so that data is correctly aligned CL cycles from the receipt of a Read command.

To summarize, for high-performance DRAM, precise CL control depends on the inclusion of a clock alignment circuit. Without the clock alignment circuit, simply counting the correct number of command clocks as a basis for firing the data transmitter circuits will result in a large time variation between the anticipated CL and actual CL. This variation is a result of the asynchronous input and output circuit path distribution delay. Therefore, if cycle-accurate CL is part of the protocol specification, then a clock alignment circuit is a necessity.

If we refer to Figure 12.42, we see that there is already a circuit in the clock alignment topology that tracks the phase difference between the *C/A CLK* and the *RdCLK* domains. This circuit is the *I/O* feedback model used in the clock alignment circuit. The absolute phase difference between the *C/A CLK* and the *RdCLK* is given by:

$$\theta_d = \frac{t_{in} + t_{out}}{t_{CK}} 2\pi \text{ rad} \tag{12.8}$$

If $t_{in} + t_{out} > t_{CK}$, then the absolute phase difference is greater than a single clock period. Given this situation, the method used to establish the phase difference between the clock domains must account for this absolute difference and not just the fractional phase difference that might be present. Figure 12.43 is a circuit configuration that can synchronize a signal between the two clock domains.

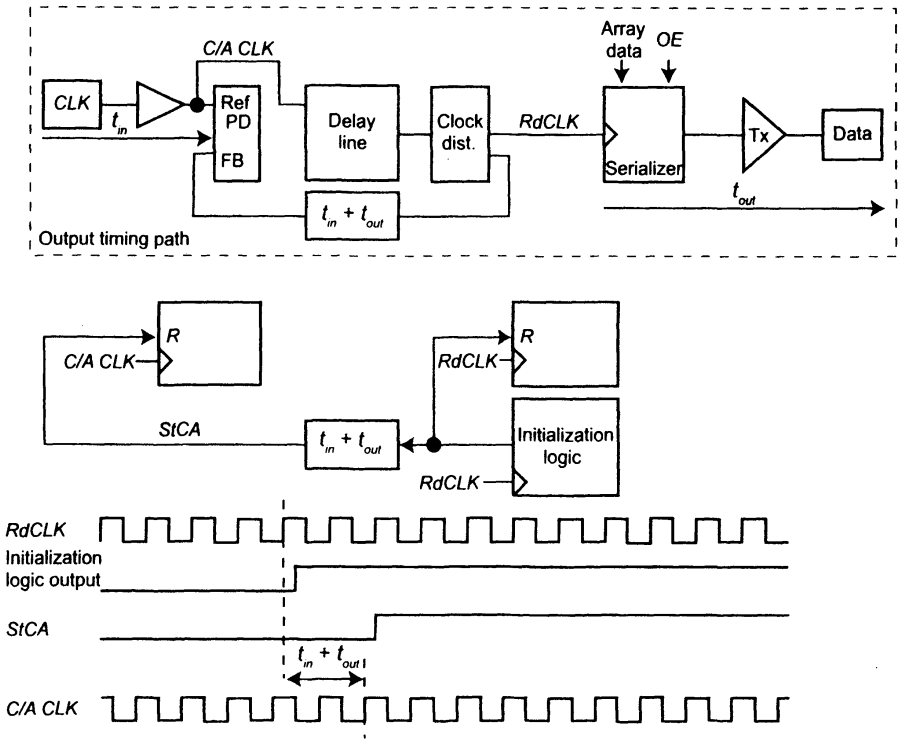


Figure 12.43 Synchronization between output data clock *(RdCLK)* domain and command clock *(C/A CLK)* domain (DLL-based example).

The circuit diagram in Figure 12.43 shows a signal being transmitted from the *RdCLK* domain back to the *C/A CLK* domain through an *I/O* delay model identical to that used for the clock alignment circuit. Because the *RdCLK* signal is advanced relative to the *C/A CLK* signal by the clock alignment circuit, and the phase difference between the feedback clock and *C/A CLK* is nominally 0 rad, then we can assume that the transmitted signal, *StCA*, is synchronized to the *C/A CLK* domain. It is a simple case where *StCA* is delayed by the same delay for which the *RdCLK* is advanced. Given that *StCA* originates from the *RdCLK* clock domain, it is safe to assume that *StCA* is synchronized to the *C/A CLK* domain following the *I/O* delay model. Of course, there will be imperfections in delay matching of the *I/O* model circuits, and there will be phase differences between the reference and feedback clocks at the phase detector of the clock alignment circuit. For these cases, margin within the transmitting clock cycle should be considered and, if the clock period is very short, frequency division of the clock should also be considered.

Now that we have established a method to synchronize a signal between the *C/A CLK* and *RdCLK* domains and also accounted for the absolute phase difference between the two clock domains, we now require a method to continuously track phase differences between the two clock domains given

variations in voltage and temperature. Figure 12.44 is a modification of Figure 12.43 showing how the phase difference can be referenced and phase variations tracked between the *C/A CLK* and *RdCLK* domains. What is shown here is a method whereby the counter referenced to the *RdCLK* domain is started at the same time that the *StCA* signal is launched through the *I/O* delay model. When the *StCA* signal emerges from the *I/O* delay model, it is used to synchronously start the counter referenced to the *C/A CLK* domain. The result, shown in the timing diagram of Figure 12.44, is two free running counters with corresponding values separated by the phase difference between the *C/A CLK* and *RdCLK* clock domains. Now, as long-term changes in voltage and temperature occur, the corresponding counter values will adjust accordingly and always maintain the correct phase difference because each counter is running independently in each of the clock domains!

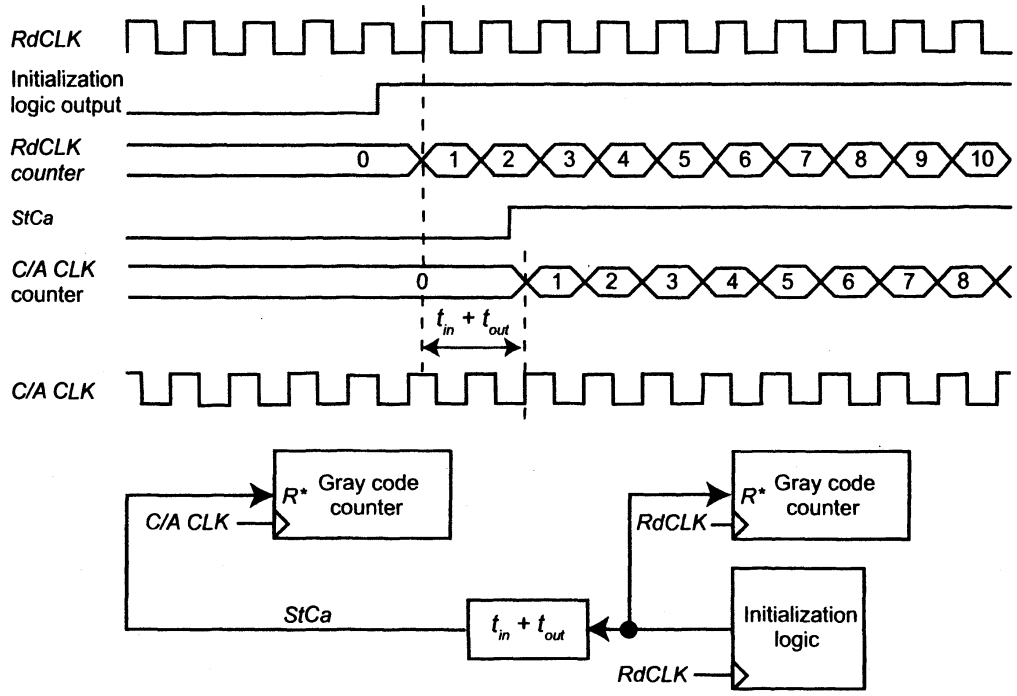


Figure 12.44 Counter-based phase tracking initialization.

Up to this point in the description of our CL tracking scheme, we have established a continuous reference between the *C/A CLK* and *RdCLK* clocks that reflects the phase difference between the clock domains, even considering changes in voltage and temperature. The next step is to account for the programmed CL value and any synchronization points that are in the Read command decode path or OE timing and logic path. Synchronization points are flip-flops or retiming logic that is in the logic path affecting signals associated with Read command processing or OE generation. An example would be a flip-flop used to retime the Read command to the command

clock following the command decoder. Keep in mind, of course, from the previous discussion, that the Read command sent to the array is not retimed to any clock; but the Read command used for CL tracking is synchronized because CL tracking is, inherently, a synchronous operation.

If synchronization points are present in the CL tracking logic, and these retiming circuits logically delay signals or logic used for CL timing, then we must account for these cycles when tracking CL. Continuing with our CL tracking methodology, Figure 12.45 is again a modification of earlier diagrams to show a calculation that is used to determine a load value for the *C/A CLK* clock domain counter. Because the synchronous initialization of the counters accounts for the I/O delay induced phase difference between the *C/A CLK* and *RdCLK* domains, by preloading one of the counters with an offset based on the compensated programmed CL value, the delay between corresponding values of the two counters will reflect the internal CL timing necessary to ensure correct external CL timing. The internal CL timing is given by:

$$tCL_{int} = (CL - SP)t_{CK} - (t_{in} + t_{out}) \tag{12.9}$$

where SP is the number of synchronization points in the CL tracking logic path. There are two parts to Equation (12.9). The first part is the synchronous portion that accounts for the programmed CL and the synchronization points (SP) in the tracking logic. The second part accounts for the asynchronous phase difference between the *RdCLK* and *C/A CLK* clock domains caused by the clock alignment circuit.

Before continuing, the following discussion will summarize the conditions shown in the timing diagram of Figure 12.45. The offset value that is loaded into the *C/A CLK* domain counter prior to initialization is based on the synchronous portion of Equation (12.9). Let us consider the hypothetical situation where SP = 0 and $(t_{in} + t_{out}) = 0$. For this situation, the value of tCL_{int} would be equal to the external value of t_{CL} ($t_{CL} = \text{CL} \bullet t_{CK}$) and corresponding count values for the *C/A CLK* counter and *RdCLK* counter would occur simultaneously. If we now assign a non-zero value to $(t_{in} + t_{out})$ and leave SP = 0, then any count value out of the *C/A CLK* counter would correspond to the same *RdCLK* counter value but be advanced by $(t_{in} + t_{out})$ before t_{CL} expires. At this point, the relationship between the counters is accounting for the input and output circuit path delays. Next, if we were to start the *C/A CLK* counter with an offset value equal to CL-SP, while starting the *RdCLK* counter at the value 0, we would observe that any count value occurring at the output of the *C/A CLK* counter would occur tCL_{int}

later at the output of the *RdCLK* counter. With the counters configured in this fashion, we are now internally tracking CL, which is defined relative to the external command clock domain.

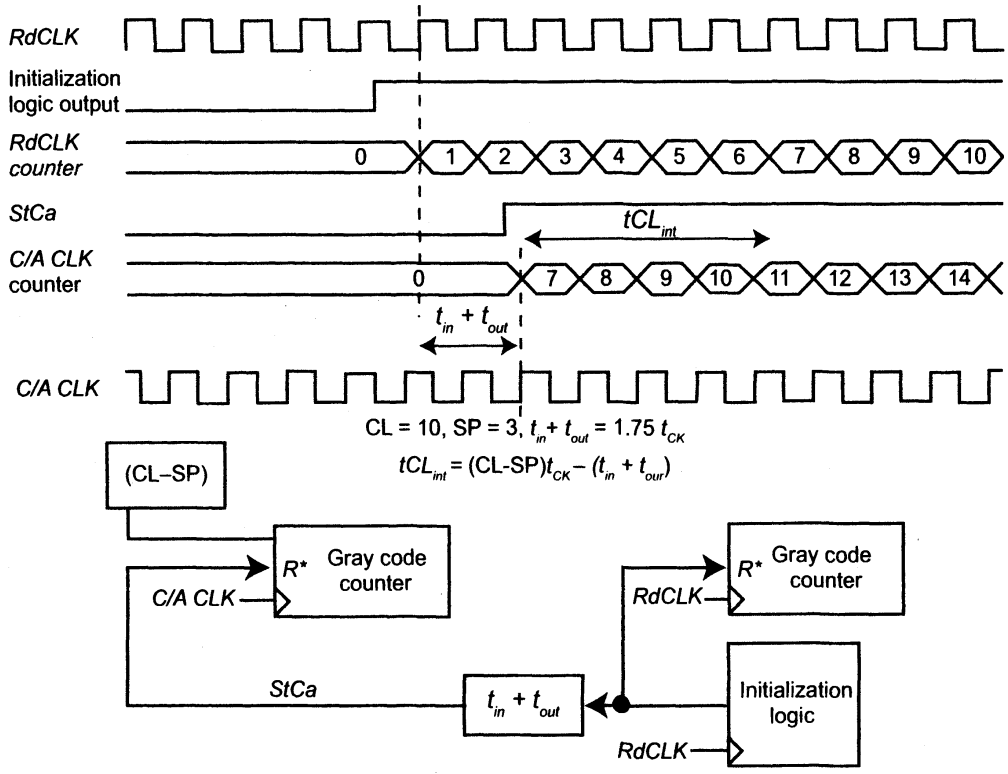


Figure 12.45 Load value for *C/A CLK* clock domain counter.

Now that we have established a method for tracking the internal CL timing, we need a means to store count values following a Read command. The method described so far requires careful consideration of the synchronous relationship between when a Read command is captured at the inputs of the DRAM and when transitions occur on the *C/A CLK* counter. For a matched-mode command and address capture, where the clock and commands are transported together to the capture latches, we must ensure that the command decoder generates a signal, labeled "Read" in Figure 12.46, which has a known correspondence to the clock edge that is aligned to the Read command at the inputs of the device. This signal must be used to capture the transition out of the *C/A CLK* counter that corresponds to the cycle on which the Read command is decoded.

If we refer to Figure 12.40 in Section 12.5.1, we see that in this example five Read commands are received before the expiration of the internal CL time. The implication of this is that if we are to capture the value of the *C/A CLK* counter for consecutive Read commands, then we must store multiple count values before the captured value of the first Read command is consumed. Once again, we use a FIFO to solve this problem. Referring to Fig-

ure 12.46, each time a Read command is decoded, the signal Read is generated and used to load the *C/A CLK* counter value into the FIFO. Once the *RdCLK* counter attains the current output value of the FIFO, a digital comparator indicates a match and signals back to the FIFO that the value has been consumed. At the same time, the comparator indicates to the OE timing circuit that the transmit circuits must be enabled on the next cycle. The OE timing circuit drives the OE signal to the data serializers in the output circuit paths for the length of the Read burst. The sequence of events required to enable the output data path is shown in Figure 12.47. Note that a synchronous transfer of the OE signal from the digital comparator to the serializer would cost one SP. Likewise, the synchronous transfer from the OE timer circuit to the serializer would cost another SP. Gray-code counters are used for the clock domain counters [22]. By using gray-code counters, false compares are eliminated from the digital comparator and FIFO loads are more reliable because only one bit transitions on each clock cycle. This configuration is illustrated in Figure 12.46.

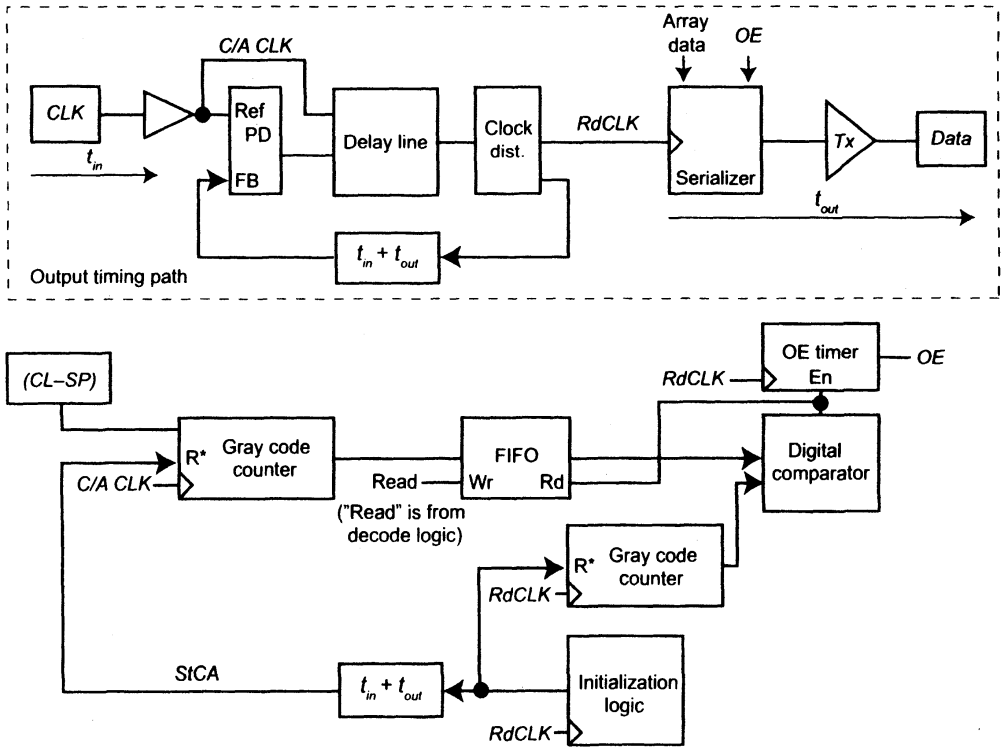


Figure 12.46 Functional block diagram of CL tracking circuit.

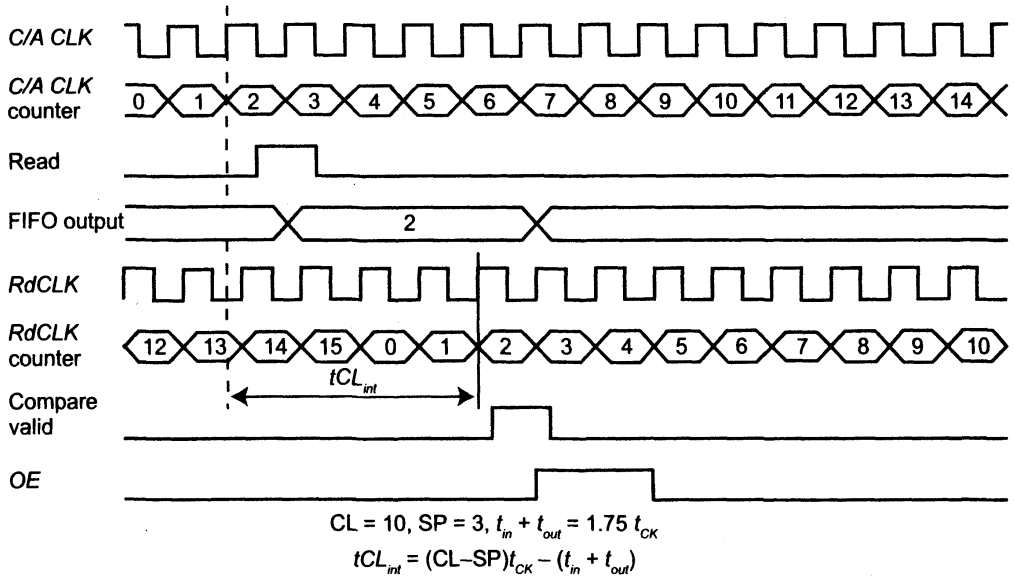


Figure 12.47 Timing diagram of output data path enable.

The preceding discussion of the Read latency tracking circuit is but one example of how we might track Read latency. Another proposal [21] is very similar but uses tokens in connected shift registers rather than counters to track the difference between clock domains. This method has merit except that there is higher clock loading and potentially longer output delay due to a large number of mux terms if a wide range of latency values is required. For high-performance DRAM, the absolute value of CL increases greatly relative to SDRAM and early DDR DRAM and, because the device must often be designed to cover a wide range of data rates, the number of CL values that are valid is greatly increased over less demanding DRAM performance levels. Another advantage of the method described in this text is that the *C/A CLK* and *RdCLK* are not required to be physically routed to the same area of the die. The use of a FIFO allows the clock routing for the separate clock domains to remain in their respective areas of the die.

12.6 COMMENTS ON FUTURE DIRECTION FOR DRAM LOGIC DESIGN

Processor performance continues to advance as more cores and higher bus frequencies demand higher data rates from the DRAM. These demands directly impact the DRAM interface protocol and, hence, the DRAM control logic. This chapter is far from a comprehensive overview of DRAM control logic. There are many details not included in this discussion but hopefully the reader will come away with a new appreciation of some of the

difficulties and, perhaps, a few ideas for solutions to problems associated with DRAM interface logic design.

One area not covered in this chapter, but becoming of greater importance in the area of DRAM interface logic design, is channel initialization algorithms for the timing of data capture. At the time of this writing, several proposals are under consideration for standardized channel initialization algorithms. To date, memory controller designs have used the published DRAM protocols for initializing the data channels. Whether this trend continues and channel initialization remains in the province of the memory controller or becomes a large part of the DRAM logic design remains to be seen.

There are other areas of consideration for high-performance DRAM that will profoundly impact DRAM logic design. In some niche applications, packetized DRAM [16] has been employed to reduce pin count at the expense of increased channel bandwidth requirements. In future DRAM, the packetized protocol may become more prevalent and standardized. As a result, DRAM logic designers will be faced with a host of new and interesting problems. Differential signaling is another possible trend. With differential signaling, channel bandwidth can and must be greatly increased to justify the cost of higher pin count. Unfortunately, the implications of these possible trends for DRAM logic design are not presented here but are mentioned so that the reader might pursue these areas for forward-looking research.

REFERENCES

[1] D. Harris, *Skew Tolerant Circuit Design.* San Francisco, CA: Morgan Kaufman Publishers, 2001.

[2] John P. Uyemura, *CMOS Logic Circuit Design.* Norwell, MA: Kluwer Academic Publishers, 1999.

[3] Y. Taur, "CMOS scaling and issues in sub-0.25um systems" in *Design of High-performance Microprocessor Circuits* ed. by A. Chandrakasan, W. J. Bowhill, and F. Fox. Piscataway, NJ: IEEE Press, 2001.

[4] T. Sakurai and A. R. Newton, "Alpha-power law MOSFET model and its application to CMOS inverter delay and other formulas," *IEEE Journal on Solid-State Circuits*, vol. 25, pp. 584–593, April 1990.

[5] K. Chen and C. Hu, "Performance and scaling in deep submicron CMOS," *IEEE Journal on Solid-State Circuits*, vol. 33, pp. 1586–1589, Oct. 1998.

[6] R. K. Krishnamurthy, A. Alvandpour, S. Mathew, M. Anders, V. De, B.and Borkar, "High-performance, low-power, and leakage-tolerance challenges for sub-70nm microprocessor circuits," in *Proc. 28th European ESSCIRC 2002*, pp. 315–321.

[7] Y. Takao, H. Kudo, J. Mitani, Y. Kotani, S. Yamaguchi, K. Yoshie, K., and M. Kawano, "A 0.11 μm CMOS technology with copper and very-low-k interconnects for high-performance system-on-a chip cores," in *IEDM Technical Digest International*, 2000, pp. 77–80.

[8] S. Wong, G. Lee, and D. Ma, "Modeling of interconnect capacitance, delay and crosstalk in VLSI," *IEEE Transactions on Semiconductor Manufacturing*, vol. 13, pp. 108–111, Feb. 2000.

[9] T. Sakurai, "Closed-form expressions for interconnection delay, coupling and crosstalk in VLSIs," *IEEE Transactions on Electron Devices*, vol. 40, pp. 118–124, Jan. 1993.

[10] R. H. Krambeck, C. M. Lee, and H. S. Law, "High-speed compact circuits with CMOS," *IEEE Journal on Solid-State Circuits*, vol. SC-17, pp. 614–619, June 1982.

[11] P. Gronowski, "Issues in dynamic logic design" in *Design of High-performance Microprocessor Circuits*. ed. by A. Chandrakasan, W. J. Bowhill, and F. Fox. Piscataway, NJ: IEEE Press, 2001.

[12] K. Bernstein, K. M. Carrig, C. M. Durham, P. R. Hanson, D. Hogenmiller, E. J. Nowak, and N. J. Rohrer, *High-speed CMOS Design Styles*. Norwell, MA: Kluwer Academic Publishers, 1999.

[13] D. V. Campenhout, T. Mudge, and K. A. Sakallah, "Timing verification of sequential domino circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, pp. 645–658, May 1999.

[14] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, pp. 720–738, June 1989.

[15] S. B. Furber and P. Day, "Four-phase micropipeline latch control circuits," *IEEE Transactions on Very Large Scale Integration,* vol. 4, pp. 247–253. June 1996.

[16] J. H. Choi, Y.-S. Sohn, C.-K. Kim, W.-K. Park, J.-H. Lee, U. Kang, G.-S. Byun, I.-S. Park, B.-C. Kim, H.-S. Hwang, C.-H. Kim, and S.-I. Cho, "A 5.0 Gbps/pin packet-based DRAM with low latency receiver and process insensitive PLL," in *Symp. on VLSI Circuits Digest of Tech. Papers 2005*, 2005, pp. 50–51.

[17] H. A. Collins and R. E. Nikel, "DDR-SDRAM, high-speed, source-synchronous interfaces create design challenges," *EDN*, Sept. 2, 1999.

[18] 1Gb DDR3 SDRAM datasheet, Micron Technology, Inc.

[19] S. H. Lee and J. B. Johnson. 2006. Method and apparatus for timing domain crossing. US Patent #7,084,680, August, 2006.

[20] J. B. Johnson, B. Keeth, F. Lin, and H. Zheng, "Phase intolerant latency control for a combination 512Mb 2.0Gb/s/pin GDDR3 and 2.5Gb/s/pin GDDR4 SDRAM," in *Digest ISSCC 2007*, 2007, pp. 494–495.

[21] S.-B. Lee, S.-J. Jang, J.-S. Kwak, S.-J. Hwang, Y.-H. Jun, S.-I. Cho, and C.-G. Lee, "A 1.6-Gb/s/pin double data rate SDRAM with wave-pipelined CAS

latency control," *IEEE Journal on Solid-State Circuits*, vol. 40, pp. 223–232, Jan. 2005.

[22] D. D. Gajski, *Principles of Digital Design*. Saddle River, N.J.: Prentice-Hall, Inc., 1997.

[23] T. Zhang and S. Sapatnekar, "Simultaneous shield and buffer insertion for crosstalk noise reduction in global routing," *IEEE Transactions on Very Large Scale Integration*, vol. 15, pp. 624–635, June 2007.

[24] C. Svenssen and J. Yuan, "High-speed CMOS circuit technique," *IEEE Journal on Solid-State Circuits*, vol. 24, pp. 62–70, Feb. 1989.

[25] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, *Logic Synthesis of Asynchronous Controllers and Interfaces*. Berlin: Springer-Verlag, 2002.

[26] H. Partovi, "Clocked storage elements," in *Design of High-Performance Microprocessor Circuits* ed. by A. Chandrakasan, W. J. Bowhill, and F. Fox. Piscataway, NJ: IEEE Press, 2001.

Chapter
13

Power Delivery

An interesting thing happens on the way to high performance. Tolerance for supply noise and voltage gradients drops off a cliff. This increased sensitivity to supply voltage arises from two different factors. The first is that the timing margins for high-frequency designs are very thin, leaving less margin on the table for supply variation. The second is that the inevitable scaling of supply voltages creates a need for better power delivery networks just to maintain the status quo. After all, 100 mV of supply noise on a 1-volt supply is a much greater problem than 100 mV of supply noise on a 2.5-volt supply. In this chapter, we explore the power delivery problem in greater detail and ways to mediate it.

As an obvious outcome of moving toward higher clock frequencies and faster data rates, on-die timing gets squeezed. This is especially true in and around high-speed *I/O*, pipeline, and control circuits. As timing budgets shrink, the margins available for supply variation also shrink. This leads to the inevitable conclusion that to reduce these variations circuit designs must either tolerate more power supply variation or improve the power delivery network. Typically the most timing-sensitive circuits are involved in *I/O* operations, such as capturing input data and driving output data. Data capture circuits often involve clock receivers, DLL or PLL blocks, clock distribution networks, and input buffers and capture latches. The actual path from the clock and data pins to the capture latches can be very long—on the order of multiple clock cycles in electrical length. A long electrical path, of course, leads to a greater sensitivity to power supply noise and variation.

The delay sensitivity of typical CMOS logic to supply voltage ranges between two to ten times greater than that of fully differential circuits, such as current mode logic (CML) [1]. As a result, highly sensitive timing circuits should use as much differential logic as possible. Furthermore, CML

logic is significantly faster due to its current-steering mode of operation, and also from smaller signal swings. Propagation delays for CML logic are shown to be 60% faster than those of standard CMOS logic. However, the static power of CML circuits is higher than that of CMOS logic because they gobble current regardless of whether they are switching or not [2]. Dynamic power remains comparable between the two logic styles.

So, given the fact that we use supply-tolerant logic such as CML throughout our high-speed critical timing circuits, do we still need to worry about power delivery? Unfortunately, yes, power delivery remains critical for high-performance designs. This is true for critical timing circuits, but also for power-hungry circuits in which the on-die supply network must be designed to meet both peak and sustained current demands. A good example of the latter is the helper flip flop (HFF) block in a GDDR3 memory device. Supply current to the HFF block is high because it services a wide data path over very short (2.5ns) column cycle times. We fully investigate this example later in the chapter.

Historically, power delivery networks (PDN) have not received the level of attention that they deserve. Given the performance demands of modern high-performance parts, a lax attitude towards PDN design is no longer an option. As shown in Figure 13.1, supply voltages are on a perpetual downward trend. While reductions in supply voltage help reduce power dissipation, they also place significantly greater demands on PDN designs, as previously discussed. High-performance designs can generally tolerate only 3–5% of AC noise on the supplies. This translates to 75–125 mV for a 2.5-volt supply, while we're looking at only 36–60 mV for a 1.2-volt supply. Ultimately, the PDN needs ongoing improvements at a rate that tracks reductions in the supply voltage. This is necessary merely to maintain the status quo as it will take even greater efforts to get ahead of the curve.

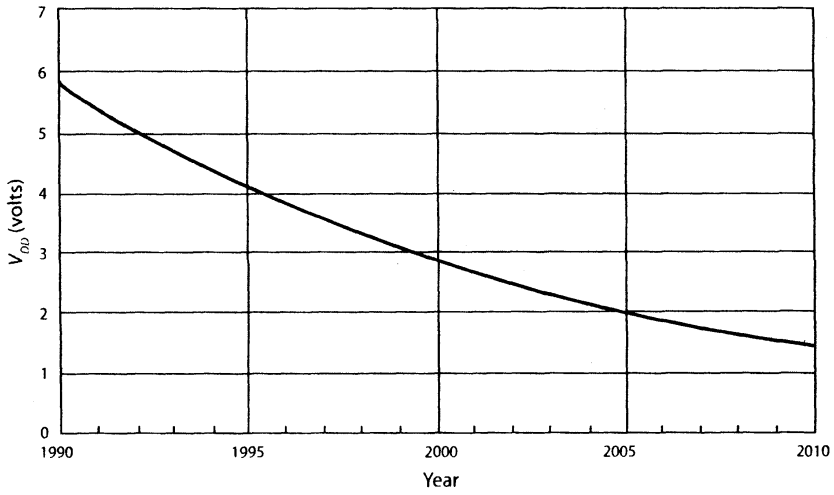


Figure 13.1 DRAM supply voltage trend.

13.1 POWER DELIVERY NETWORK DESIGN

There are a variety of methods available to ensure that the power delivery network (PDN) is capable of meeting design requirements. The methods all involve some type of PDN model and either static or dynamic simulations. One popular method, widely supported by modern IC design tools, is static supply analysis. This requires a PDN model that is either built up by hand or extracted from the actual layout database using specialized software. This type of analysis uses static current loads connected throughout the PDN model to evaluate the capabilities of the design. Each current load model idealizes the load currents of circuits that operate in the vicinity of each connection point. This method is a reasonable approach to solving PDN problems while producing relatively quick simulation turn times.

Another approach to PDN design involves using full dynamic simulations. Again, this requires a PDN model but with actual circuits in lieu of current sources to stimulate the model. Using full dynamic simulations provides a notable advantage over static analysis because it supports PDN analysis with a variety of vectors, and it also exposes the effects of PDN noise on circuit performance and timing. Thus, it shows both cause and effect. Furthermore, if the PDN model incorporates a full package netlist, then this flow both enables and supports device and package co-design. Co-design is a process in which the device and package are designed and tuned in concert with one another to maximize their combined performance. This is a powerful and evolving piece of high-performance design flows. Co-design is an absolute necessity in the multi-gigahertz realm.

Let's take a look at an example of PDN design using the now-familiar GDDR3 device. Our exemplary GDDR3 device has three levels of on-die metallization plus one redistribution layer (RDL) that permits connection to a four-layer flip chip in package (FCIP) substrate. That's a total of eight metal layers for this high-performance DRAM. It may seem a bit excessive, but the results are truly impressive.

Figure 13.2 shows both the power bussing and signal routes of the metal3 layer from the GDDR3 design. There is actually dense metal3 over the white areas in the array. For clarity, this is not shown in the image. Essentially, metal3 is the upper-level metal used extensively for on-die power distribution. It works in concert with the metal2 layer, that runs orthogonal to metal3 to deliver power throughout the entire die. The primary bond pads of this GDDR3 device are separated into two rows, as shown in Figure 13.2. The power pads within these pad rows tie directly into the metal3/metal2 power bus network.

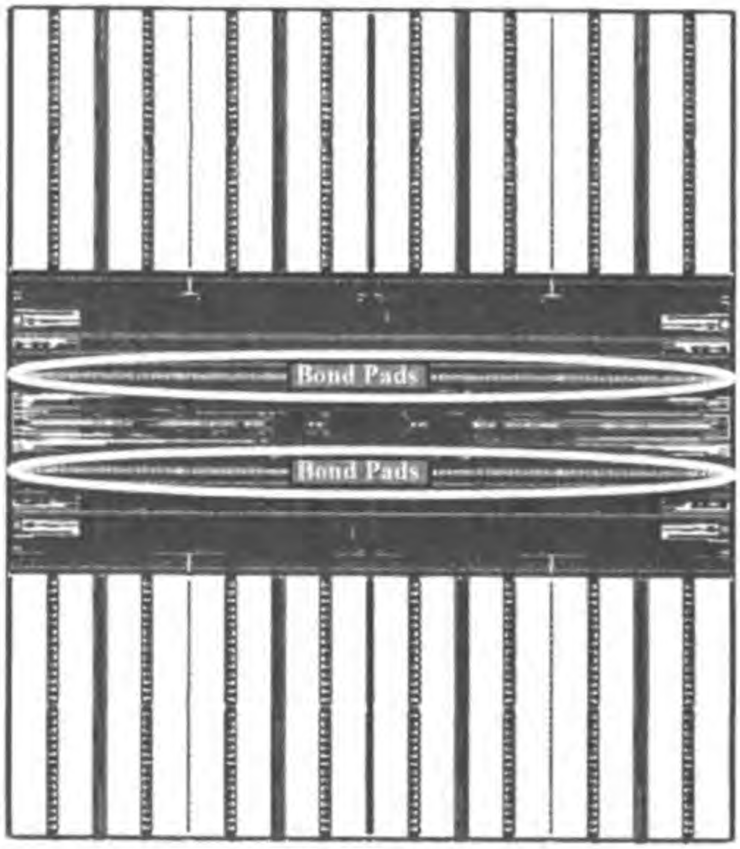


Figure 13.2 GDDR3 metal3 layout.

Obviously, what is shown in Figure 13.2 represents an actual layout from the exemplary GDDR3 device. During the early and intermediate phases of a DRAM design, this level of layout does not actually exist. Ideally, layout of the die would not even start without a preliminary floor plan and PDN plan in place. While it is possible to scribble out a floor plan and PDN plan on the back of a napkin, it is more prudent and reliable to build and tune these representations together using simulation models. The diagram shown in Figure 13.3 is a schematic diagram of an actual PDN model used for tuning the GDDR3 device. This model is built using parameterized wire elements that represent metal bus segments in a proposed PDN. The elements are parameterized to permit easy changes to bus widths and lengths as required by the tuning exercise. The model also includes decoupling capacitors scattered throughout its structure. These capacitors are important to PDN design and analysis because they serve as local sources of charge for all of the circuit blocks.

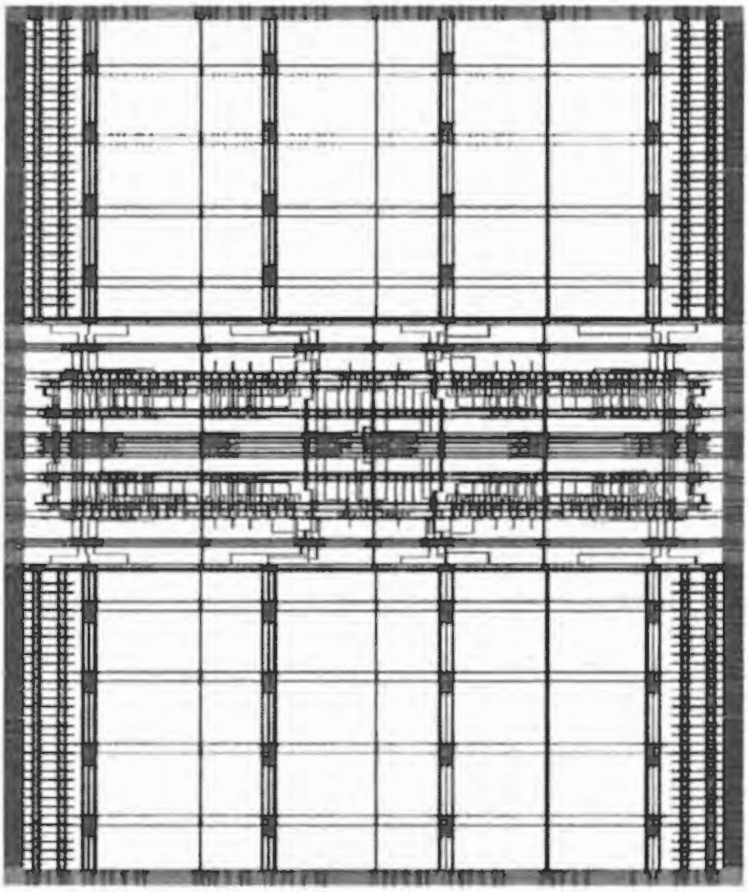


Figure 13.3 GDDR3 on-die PDN model.

Basically, every circuit contained in the full-chip simulation netlist must be mapped onto the PDN model. This eliminates the ideal power and ground connections that we generally deal with in simulation space, replacing them with connections to the non-idealized power grid. Elimination of ideal power connections significantly increases simulation time because even small amounts of current moving around the network can impact large segments of the voltage grid. Localized disturbances of the supply grid impacts the circuit simulation in a variety of ways. Notably, transistor and gate behavior depend on local conditions. As a result, some circuits speed up while others slow down. Signaling levels, both within a circuit block and between blocks, are affected by power supply noise and gradients. Changes in the common mode voltage from one area of the die to another impact both signal and clock distribution. In essence, your simulation will exhibit more real-world behavior than ever before. As a result, you will be able to observe power supply voltages throughout the PDN under a variety of operating conditions and simulation vectors, but, more importantly, you can see

how the resulting supply noise impacts circuit operation—a tremendous one-two punch in observeability.

13.2 DEVICE/PACKAGE CO-DESIGN

With the addition of a package model to the PDN simulation just described, it becomes possible to perform device/package co-design. As stated earlier, the package used for the exemplary GDDR3 device is designed with FCIP technology. Figure 13.4 illustrates what an FCIP package looks like in cross section. Figure 13.4a is a conceptual drawing, while Figure 13.4b–d are cross-sectional photographs of an actual GDDR3 package at various magnifications. Essentially the die is mounted with its circuit side facing the package substrate. An RDL layer is commonly used to move die connection points to locations that are compatible with the underlying substrate technology. The substrate itself is typically constructed of conventional PCB materials, although at a very fine pitch. The number of conductor layers within the substrate is based on a balance between design needs and cost. All four conductor layers are clearly visible in the GDDR3 package shown in Figure 13.4b–d.

The package design used on a first-generation GDDR3 device is depicted through its various metal layers in Figure 13.5. All five layers are shown in Figure 13.5a–e, starting with the RDL layer and proceeding down through the four substrate layers from top to bottom. Figure 13.5f is an exploded view showing all five package layers in their proper stack-up order. This package is the product of a chip and package co-design flow. The flow optimized each element in the overall power delivery network, including the package and on-die PDN, for maximum device performance.

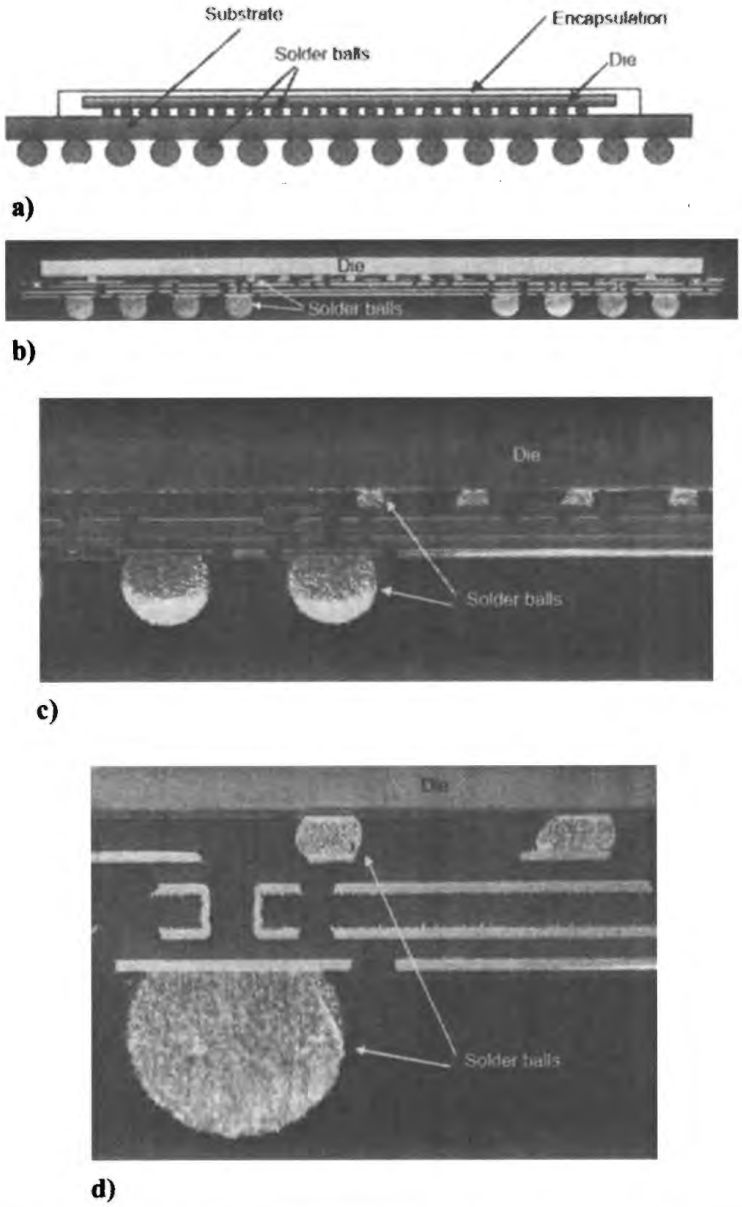
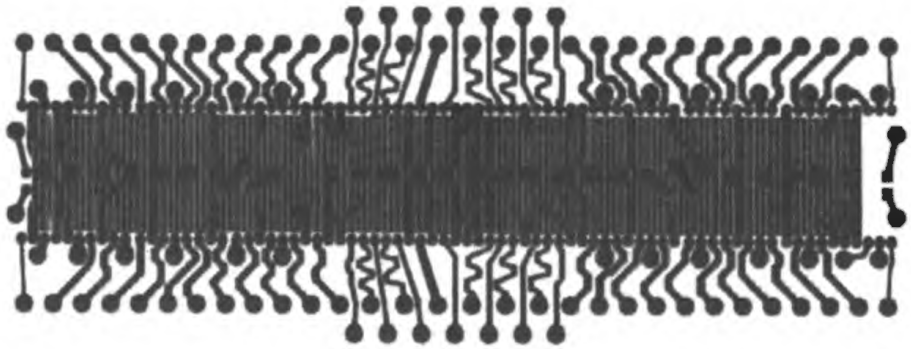
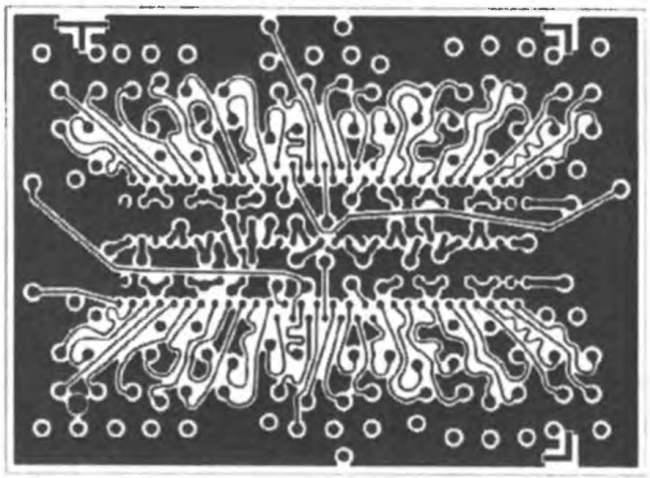


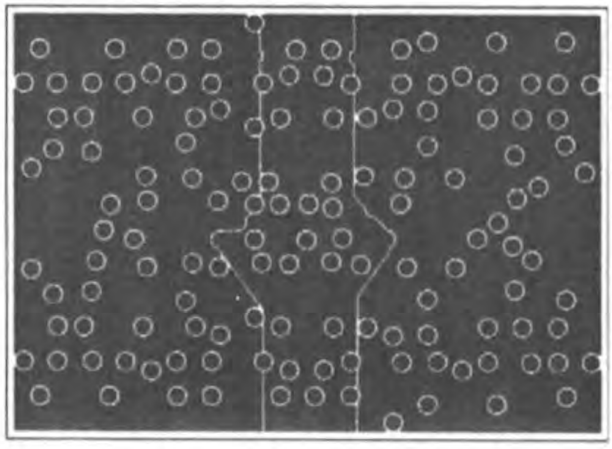
Figure 13.4 a) GDDR3 FCIP conceptual package diagram, b) GDDR3 FCIP package cross section photograph, c) GDDR3 FCIP package cross section detail, and d) GDDR3 FCIP package cross section, magnified detail.



a)

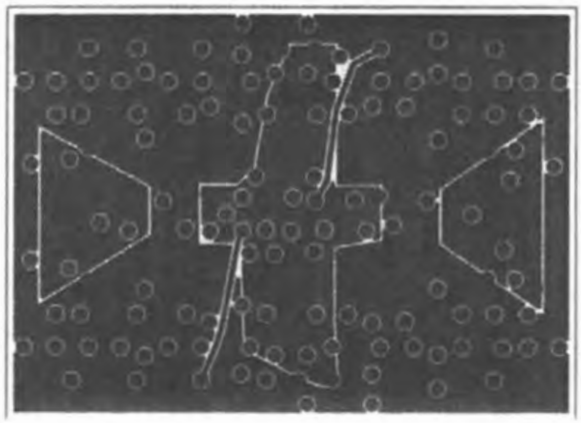


b)

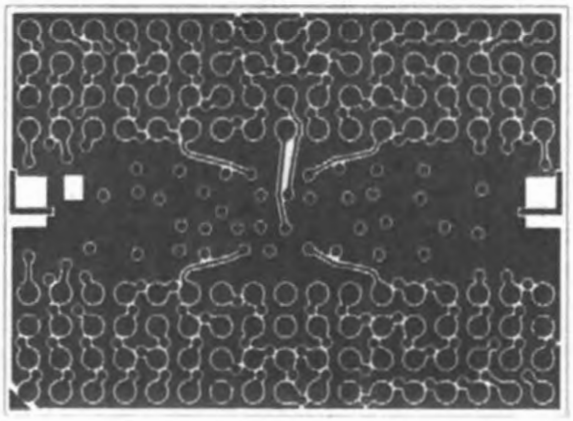


c)

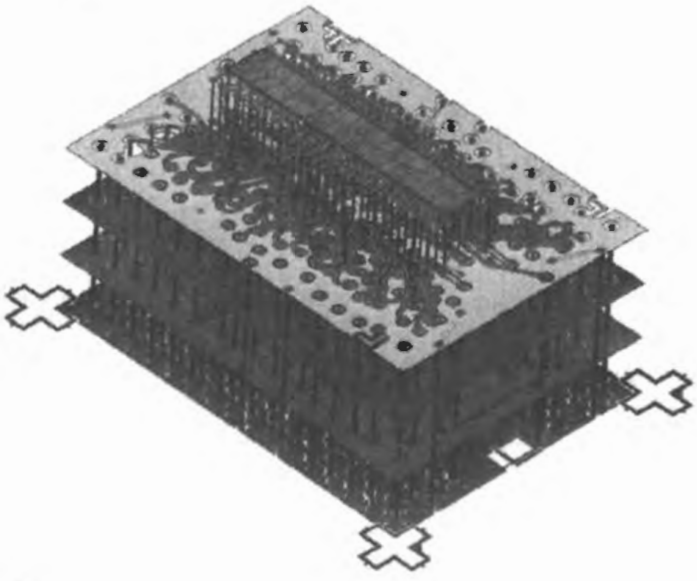
Figure 13.5 a) GDDR3 RDL layer, b) GDDR3 top substrate layer, and c) GDDR3 second substrate layer.



d)



e)



f)

Figure 13.5 d) GDDR3 third substrate layer, e) GDDR3 bottom substrate layer, and f) GDDR3 package layer stack.

13.3 FULL-CHIP SIMULATIONS

Let's take a look at how full-chip simulations using PDN and package models can be used to improve power delivery and device performance using a GDDR3 example. A full-chip simulation for the GDDR3 device was constructed using a device model, a PDN model, a package model, and a bus model, as shown conceptually in Figure 13.6. The bus model essentially connects the package model external ball terminals to power supplies and bus wire models to ensure that the input signals entering the package model meet specification and also to ensure that output signals are properly loaded. All stimulus and supply voltages reach the device only by traversing through the bus, package, and PDN model.

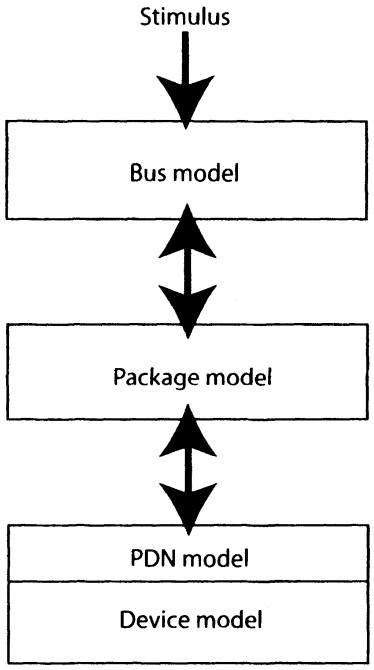


Figure 13.6 Conceptual simulation diagram.

To identify problem areas, a variety of full-chip simulations were run on the design in which both timing and power supply voltages were analyzed. One particular area of concern that emerged from the simulations was the Read clock trees. The clock trees provide critical timing information to the output data path and contribute significantly to several measurable output timing specifications. Initial simulations showed that the clock tree power busses produced approximately 100 mV of noise and sag whenever the clock trees were enabled. This result, labeled as "Before," can be seen in the simulation waveforms shown in Figure 13.7. Using the package/PDN/device simulation, we analyzed a variety of design changes to correct this problem. We ultimately chose to implement a design change that added

three pairs of V_{DD} and V_{SS} bond pads within each of the Read clock trees. These new bond pads were readily supported by the FCIP packaging technology of this design. Simulation of the revised design yielded the improvement shown in Figure 13.7 and labeled, "After." As shown, the power supply noise and sag were reduced to only 50 mV, which met our internal design goal for this voltage supply.

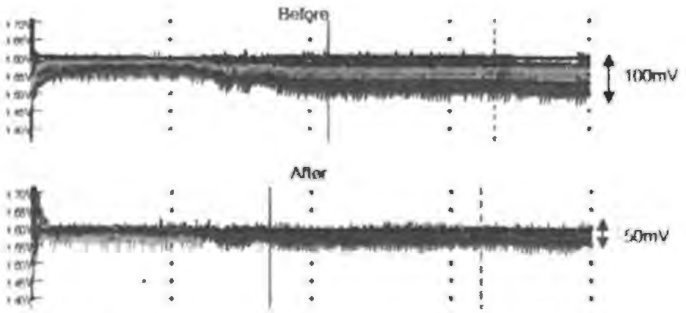


Figure 13.7 V_{DD} simulation waveforms for Read clock trees.

Figure 13.8 once again shows the metal3 layout for the GDDR3 device. In this view, however, the V_{DD}/V_{SS} power pads, which were added to support the Read clock trees, are circled to show their relative locations.

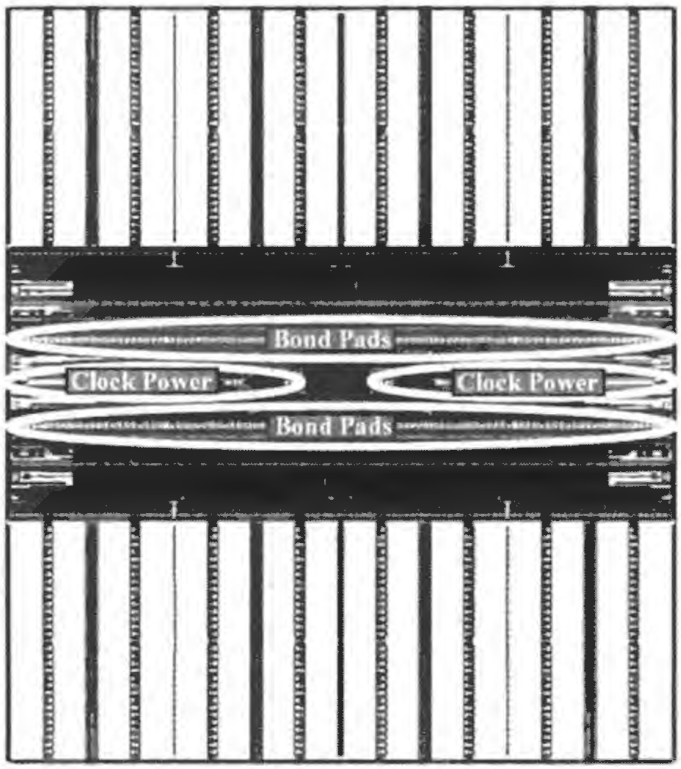


Figure 13.8 GDDR3 metal3 layout with clock tree power pads.

Another example from recent work on the GDDR3 design is depicted in Figure 13.9 and Figure 13.10. In these before-and-after images, it is easy to see the gains achieved through the PDN/package simulation and co-design efforts for the HFF block. The waveforms show the V_{DD} and V_{SS} nets in the HFF block for a back-to-back Write and back-to-back Read vector. This type of pattern places a tremendous amount of stress on column cycle time and HFF block power delivery. In Figure 13.9, the supply busses are clearly inadequate to provide stable voltage levels to the HFF circuits. They are virtually collapsing. In Figure 13.10, the HFF supply busses show a significant improvement after much-needed PDN tuning. The DC voltage levels remain stable, and the amount of AC noise lies within design targets. Without PDN/package simulations, the problem illustrated in Figure 13.9 would not have been discovered and corrected prior to tapeout. A very costly and perhaps catastrophic problem was averted.

Owing to the demands of high-performance specifications and shrinking supply voltages, power delivery is fast becoming the bane of semiconductor designers. This is especially true for leading-edge projects where the timing and supply-induced problems grow significantly more complex with each passing generation. The reality of ever-tighter design margins and shrinking supply voltages must foment solutions that move device simulations closer and closer to actual silicon behavior. Power delivery problems must be dealt with early in the design flow and addressed through an aggressive combination of simulation and co-design. Nothing, absolutely nothing can be overlooked in this regard, since everything matters.

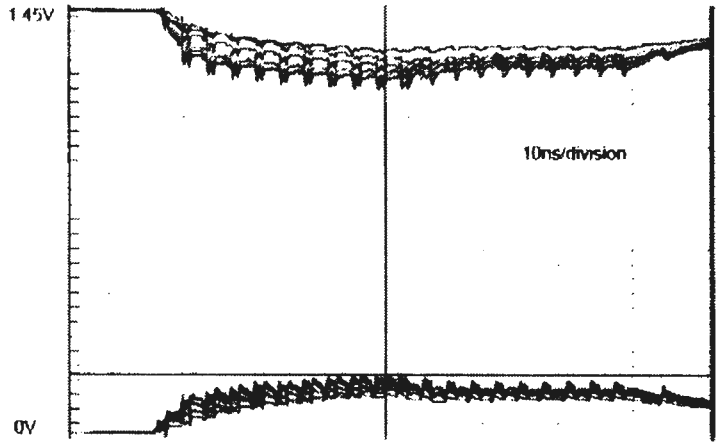


Figure 13.9 V_{DD} and V_{SS} waveforms for HFF block before tuning.

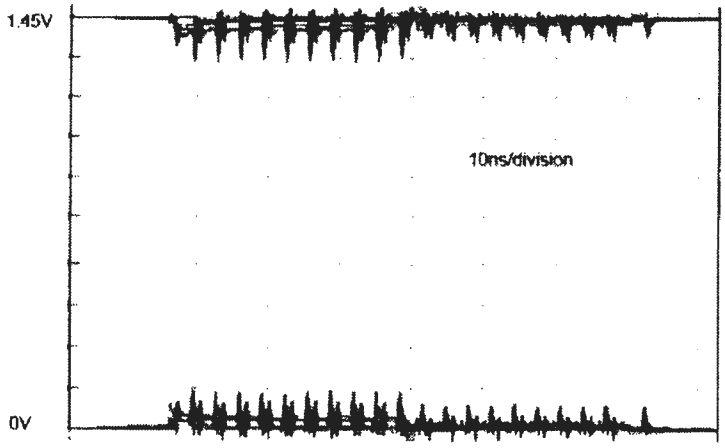


Figure 13.10 V_{DD} and V_{SS} waveforms for HFF block after tuning.

REFERENCES

[1] J. Chapman, J. Currin, and S. Payne, "A low-cost high-performance CMOS timing vernier for ATE," in *Proceedings International Test Conference 1995*, October 1995, pp. 459–468.

[2] I. Li, S. Raghavendran, and D. Comer, "CMOS current mode logic gates for high-speed applications," in *12th NASA Symposium on VLSI Design*, October 4–5, 2005.

Chapter
14

Future Work in High-Performance Memory

The semiconductor industry will continue its inevitable march forward—deriving ever greater benefits through additional scaling and higher levels of integration. Barring the unforeseen, device density, power, and speed will continue to follow their historical trends.

We gain insight into what the future may hold for DRAM by looking at the past and at various trends. For instance, Figure 14.1 is a graph showing a trend line for DRAM data rate per pin by year. This data shows steady gains from 1995 through 2010 with the data rate doubling approximately every three years. While any data beyond present day is somewhat speculative, it is based on recent trending and extrapolated historical data. Figure 14.1 shows that device performance, as it relates to *I/O* bandwidth, will continue to advance beyond the end of the decade. This is both exciting and foreboding for designers. It is exciting because engineers relish a challenging design project, yet foreboding because the trend presented in Figure 14.1 represents a never-ending series of design problems that become increasingly more difficult.

While higher bandwidths are a perceived benefit, unfortunately this benefit comes at a cost. The power dissipated by these high-strung designs also appears to rise with each new generation. Figure 14.2 shows a trend line of power dissipation for a variety of recent DRAM interface technologies. The data shown is based on the IDD4 specification for each technology. IDD4 measures supply current for back-to-back page Reads and/or back-to-back page Writes at minimum column cycle time. This specification examines power at the maximum sustainable device bandwidth. Despite the fact that supply voltages drop with each new technology, power

is nevertheless going up. The fact that the trend line rises is not surprising since it is tightly correlated to *I/O* bandwidth, internal clock frequencies, and the data pre-fetch size—all of which are on the increase.

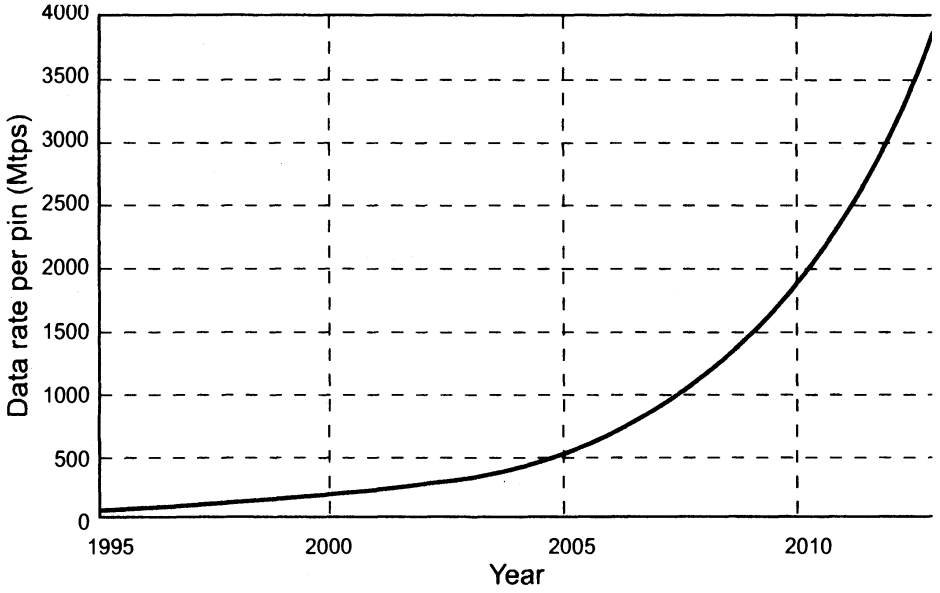


Figure 14.1 Trend line for DRAM data rate per pin.

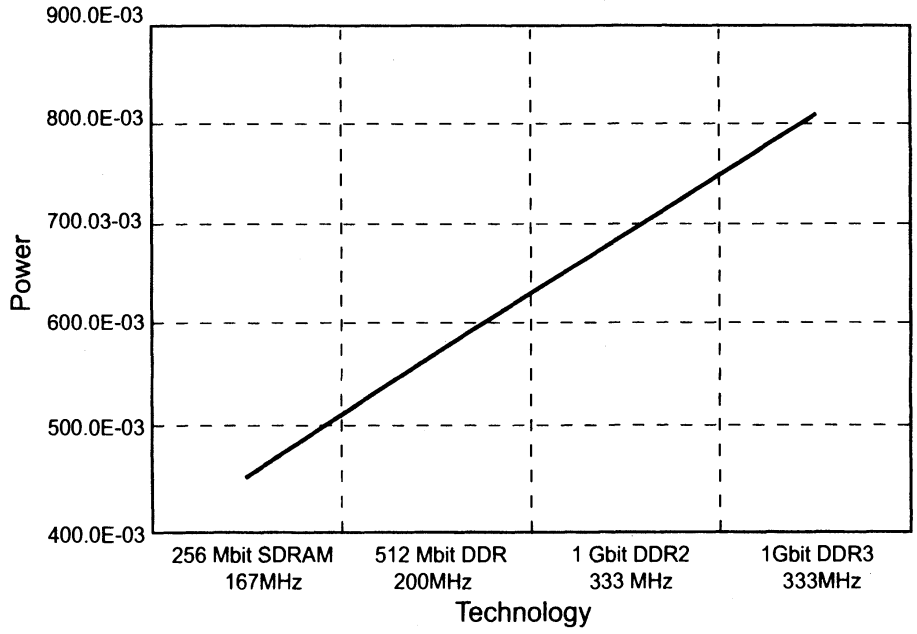


Figure 14.2 IDD4 power dissipation trend line versus technology.

Most every computer system in use today and those being designed for future use are highly power constrained. This is at odds with the device power dissipation depicted in Figure 14.2. As a result, one should expect that future technologies will adopt more sophisticated power management schemes to keep device and system power within budget. These schemes

will strive to reduce both active and standby power to absolute minimum levels through a variety of mechanisms. The most obvious being to shut down circuits not needed for ongoing operations. The degree to which circuits can be shut down depends on our tolerance for the delays we encounter when turning these circuits back on. Nothing comes for free.

Another issue pertaining to power is leakage current. All transistors leak when they are turned off—some more, some less. Historically, DRAM transistors are designed to have low leakage, especially the array Mbit transistors, since leakage impacts data retention and Refresh. However, the quest for higher performance and high-bandwidth *I/O* is creating pressure to build better transistors. Better, in this case, meaning higher drive and lower V_{TH}. Unfortunately, what leads to higher performance also leads to higher leakage current. So, in order to keep the array leakage current low while increasing peripheral transistor performance, DRAM manufacturers will be forced to add process complexity. This will translate to more masks, more process steps, and higher cost, which is just the opposite of what is needed in a highly competitive commodity market. Nevertheless, if the performance goals are to be realized, then added process complexity is the price to be paid.

Without a doubt, the most notable and predictable feature of DRAM technology is memory density and its associated growth over the years. Advances in memory density are fueled by a perpetual scaling of the underlying CMOS process technology. Figure 14.3 shows a ten-year timeline of historical DRAM shipments by density. Although the cadence in recent years appears to have tapered off a bit, memory density has essentially doubled every two to three years. Accordingly, Figure 14.4 shows a forecast of shipments by density through the end of 2010. As expected, the lower densities continue to taper off while the higher densities, starting at 512MB, grow in share until they peak and tail off. There's nothing surprising about this data since it merely follows the historical trend.

Increased memory density does create some problems for high-performance DRAM devices. There are obvious architectural ramifications, as discussed earlier in Chapter 8. Large blocks of memory can lead to long wire routes, cycle time problems, and longer latency. Not exactly what you expect in a high-performance part. Typical first-generation parts of a specific memory density will not achieve the same level of performance as their lower density counterparts. It usually takes one or two process shrinks before performance levels rise to targeted levels.

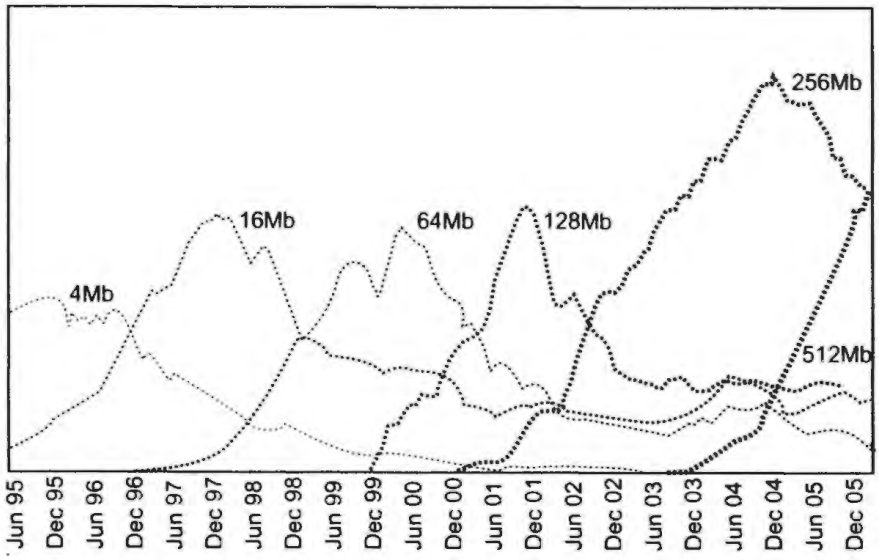


Figure 14.3 Historical DRAM shipments by density.

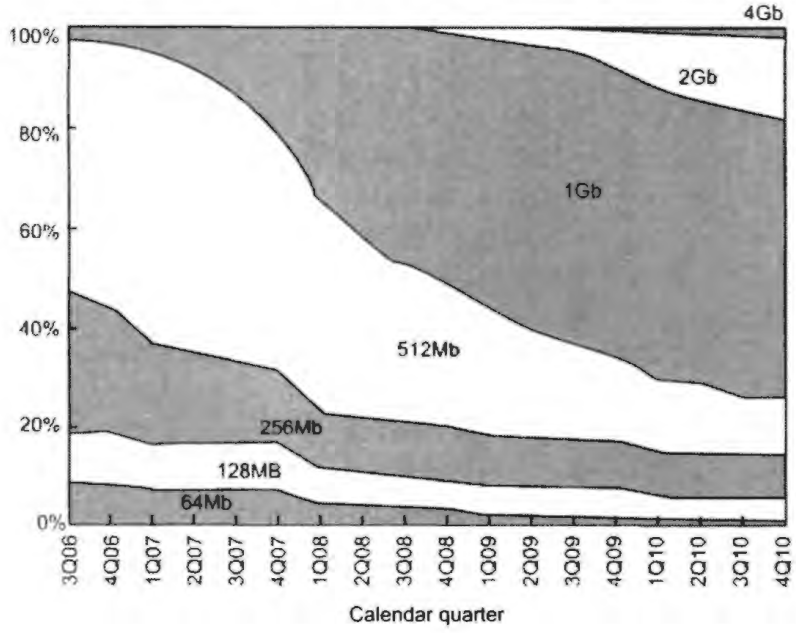


Figure 14.4 DRAM shipments by density forecast.

So, what will DRAM devices look like in the future? Given the historical trends, they should continue to grow in density and speed, burn more power, and operate at lower and lower supply voltages. But will that necessarily be the case? It's reasonable to expect these trends to hold for the foreseeable future, but they are not sustainable indefinitely. Recent transformations in the microprocessor world should serve to illustrate this point. For years, the personal computer world looked like a horse race

involving system benchmark performance and clock frequencies. This continued year after year until processor frequencies began to reach beyond 3GHz. At this point, the design challenges associated with such high-frequency operation coupled with dizzying power dissipation problems forced a change in direction. Basically, the major players started to walk away from single core processors toward multi-core designs. This resulted in seemingly higher overall performance at more reasonable power levels—a winning strategy for everyone involved. While the benchmark wars continue unabated, focus has shifted from maximizing clock frequency to increasing the number of processor cores available. The race is on, but in a new direction.

Given the course change in the microprocessor world, it seems reasonable to expect similar changes in the DRAM world. After all, *I/O* frequencies cannot rise indefinitely, especially when cost and power remain critical. I am unable to predict what those changes will be or how they will impact DRAM design, but they must produce improved system performance and lower overall power dissipation to succeed.

Research is underway on the next generation of high-performance DRAM. In fact, this type of research never ends. However, the convergence of lower supply voltages, tighter power constraints, and ever-higher clock frequencies are indeed creating a difficult set of problems to overcome. In addition, process and device variability is rising to prominence as a foreboding new challenge for the DRAM world. Variability becomes more prevalent with each new process node. This phenomenon is the by-product of transistor scaling. The most common sources of variability are line edge roughness and nonuniform dopant distribution. Design engineering teams must come to grips with variability and learn to design circuits and architectures with demonstrably lower sensitivity. This will not be an easy task. It hinges, in part, upon an ability to model variability and simulate its effects.

The future for DRAM technology looks both exciting and challenging. A combination of new architectures, circuits, and process technologies provide the needed capabilities to forge ahead. Tomorrow's high-performance devices will no doubt make today's best devices seem quaint by comparison. Each new generation presents an opportunity to advance the state of the art. Even while we strive to use our best ideas on today's design project, we nevertheless discover a better idea for use on the next. This is why DRAM design is so exciting. We are forever adding new knowledge to old. We raise device performance and customer expectations with each new project—all while standing on the shoulders of giants.

Appendix

Supplemental Reading

In this second edition of *DRAM Circuit Design*, we introduce the reader to a variety of topics—from introductory to advanced. However, we may not have covered specific topics to the reader's satisfaction. For this reason, we have compiled a list of supplemental readings from major conferences, journals, and books categorized by subject. It is our hope that unanswered questions are addressed by the authors of these readings, who are experts in the field of CMOS design.

General DRAM Design and Operation

[1] S. Fuji, K. Natori, T. Furuyama, S. Saito, H. Toda, T. Tanaka, and O. Ozawa, "A low-power sub 100 ns 256K bit dynamic RAM," *IEEE Journal of Solid-State Circuits*, vol. 18, pp. 441–446, October 1983.

[2] A. Mohsen, R. I. Kung, C. J. Simonsen, J. Schutz, P. D. Madland, E. Z. Hamdy, and M. T. Bohr, "The design and performance of CMOS 256K bit DRAM devices," *IEEE Journal of Solid-State Circuits*, vol. 19, pp. 610–618, October 1984.

[3] M. Aoki, Y. Nakagome, M. Horiguchi, H. Tanaka, S. Ikenaga, J. Etoh, Y. Kawamoto, S. Kimura, E. Takeda, H. Sunami, and K. Itoh, "A 60-ns 16-Mbit CMOS DRAM with a transposed data-line structure," *IEEE Journal of Solid-State Circuits*, vol. 23, pp. 1113–1119, October 1988.

[4] M. Inoue, T. Yamada, H. Kotani, H. Yamauchi, A. Fujiwara, J. Matsushima, H. Akamatsu, M. Fukumoto, M. Kubota, I. Nakao, N. Aoi, G. Fuse, S. Ogawa, S. Odanaka, A. Ueno, and H. Yamamoto, "A 16-Mbit DRAM with a relaxed sense-amplifier-pitch open-bit-line architecture," *IEEE Journal of Solid-State Circuits*, vol. 23, pp. 1104–1112, October 1988.

[5] T. Watanabe, G. Kitsukawa, Y. Kawajiri, K. Itoh, R. Hori, Y. Ouchi, T. Kawahara, and R. Matsumoto, "Comparison of CMOS and BiCMOS 1-Mbit DRAM performance," *IEEE Journal of Solid-State Circuits*, vol. 24, pp. 771–778, June 1989.

[6] K. Itoh, "Trends in megabit DRAM circuit design," *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 778–789, June 1990.

[7] Y. Nakagome, H. Tanaka, K. Takeuchi, E. Kume, Y. Watanabe, T. Kaga, Y. Kawamoto, F. Murai, R. Izawa, D. Hisamoto, T. Kisu, T. Nishida, E. Takeda, and K. Itoh, "An experimental 1.5-V 64-Mb DRAM," *IEEE Journal of Solid-State Circuits*, vol. 26, pp. 465–472, April 1991.

[8] P. Gillingham, R. C. Foss, V. Lines, G. Shimokura, and T. Wojcicki, "High-Speed, High-reliability circuit design for megabit DRAM," *IEEE Journal of Solid-State Circuits*, vol. 26, pp. 1171–1175, August 1991.

[9] K. Kimura, T. Sakata, K. Itoh, T. Kaga, T. Nishida, and Y. Kawamoto, "A block-oriented RAM with half-sized DRAM cell and quasi-folded data-line architecture," *IEEE Journal of Solid-State Circuits*, vol. 26, pp. 1511–1518, November 1991.

[10] Y. Oowaki, K. Tsuchida, Y. Watanabe, D. Takashima, M. Ohta, H. Nakano, S. Watanabe, A. Nitayama, F. Horiguchi, K. Ohuchi, and F. Masuoka, "A 33-ns 64-Mb DRAM," *IEEE Journal of Solid-State Circuits*, vol. 26, pp. 1498–1505, November 1991.

[11] T. Kirihata, S. H. Dhong, K. Kitamura, T. Sunaga, Y. Katayama, R. E. Scheuerlein, A. Satoh, Y. Sakaue, K. Tobimatus, K. Hosokawa, T. Saitoh, T. Yoshikawa, H. Hashimoto, and M. Kazusawa, "A 14-ns 4-Mb CMOS DRAM with 300-mW active power," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 1222–1228, September 1992.

[12] K. Shimohigashi and K. Seki, "Low-voltage ULSI design," *IEEE Journal of Solid-State Circuits*, vol. 28, pp. 408–413, April 1993.

[13] G. Kitsukawa, M. Horiguchi, Y. Kawajiri, T. Kawahara, T. Akiba, Y. Kawase, T. Tachibana, T. Sakai, M. Aoki, S. Shukuri, K. Sagara, R. Nagai, Y. Ohji, N. Hasegawa, N. Yokoyama, T. Kisu, H. Yamashita, T. Kure, and T. Nishida, "256-Mb DRAM circuit technologies for file applications," *IEEE Journal of Solid-State Circuits*, vol. 28, pp. 1105–1113, November 1993.

[14] T. Kawahara, Y. Kawajiri, M. Horiguchi, T. Akiba, G. Kitsukawa, T. Kure, and M. Aoki, "A charge recycle refresh for Gb-Scale DRAMs in file applications," *IEEE Journal of Solid-State Circuits*, vol. 29, pp. 715–722, June 1994.

[15] S. Shiratake, D. Takashima, T. Hasegawa, H. Nakano, Y. Oowaki, S. Watanabe, K. Ohuchi, and F. Masuoka, "A staggered NAND DRAM array

architecture for a Gbit scale integration," in *1994 Symposium on VLSI Circuits*, June 1994, p. 75.

[16] T. Ooishi, K. Hamade, M. Asakura, K. Yasuda, H. Hidaka, H. Miyamoto, and H. Ozaki, "An Automatic Temperature compensation of internal sense ground for sub-quarter micron DRAMs," in *1994 Symposium on VLSI Circuits*, June 1994, p. 77.

[17] A. Fujiwara, H. Kikukawa, K. Matsuyama, M. Agata, S. Iwanari, M. Fukumoto, T. Yamada, S. Okada, and T. Fujita, "A 200MHz 16Mbit synchronous DRAM with block access mode," in *1994 Symposium on VLSI Circuits*, June 1994, p. 79.

[18] Y. Kodama, M. Yanagisawa, K. Shigenobu, T. Suzuki, H. Mochizuki, and T. Ema, "A 150-MHz 4-bank 64M-bit SDRAM with address incrementing pipeline scheme," in *1994 Symposium on VLSI Circuits*, June 1994, p. 81.

[19] D. Choi, Y. Kim, G. Cha, J. Lee, S. Lee, K. Kim, E. Haq, D. Jun, K. Lee, S. Cho, J. Park, and H. Lim, "Battery operated 16M DRAM with post package programmable and variable self refresh," in *1994 Symposium on VLSI Circuits*, June 1994, p. 83.

[20] S. Yoo, J. Han, E. Haq, S. Yoon, S. Jeong, B. Kim, J. Lee, T. Jang, H. Kim, C. Park, D. Seo, C. Choi, S. Cho, and C. Hwang, "A 256M DRAM with simplified register control for low power self refresh and rapid burn-in," in *1994 Symposium on VLSI Circuits*, June 1994, p. 85.

[21] M. Tsukude, M. Hirose, S. Tomishima, T. Tsuruda, T. Yamagata, K. Arimoto, and K. Fujishima, "Automatic voltage-swing reduction (AVR) scheme for ultra low power DRAMs," in *1994 Symposium on VLSI Circuits*, June 1994, p. 87.

[22] D. Stark, H. Watanabe, and T. Furuyama, "An experimental cascade cell dynamic memory," in *1994 Symposium on VLSI Circuits*, June 1994, p. 89.

[23] T. Inaba, D. Takashima, Y. Oowaki, T. Ozaki, S. Watanabe, and K. Ohuchi, "A 250mV bit-line swing scheme for a 1V 4Gb DRAM," in *1995 Symposium on VLSI Circuits*, June 1995, p. 99.

[24] I. Naritake, T. Sugibayashi, S. Utsugi, and T. Murotani, "A crossing charge recycle refresh scheme with a separated driver sense-amplifier for Gb DRAMs," in *1995 Symposium on VLSI Circuits*, June 1995, p. 101.

[25] S. Kuge, T. Tsuruda, S. Tomishima, M. Tsukude, T. Yamagata, and K. Arimoto, "SOI-DRAM circuitt technologies for low power high speed multi-giga scale memories," in *1995 Symposium on VLSI Circuits*, June 1995, p. 103.

[26] Y. Watanabe, H. Wong, T. Kirihata, D. Kato, J. DeBrosse, T. Hara, M. Yoshida, H. Mukai, K. Quader, T. Nagai, P. Poechmueller, K. Pfefferl, M. Wordeman, and S. Fujii, "A 286mm^2 256Mb DRAM with X32 both-ends DO," in *1995 Symposium on VLSI Circuits*, June 1995, p. 105.

[27] T. Kirihata, Y. Watanabe, H. Wong, J. DeBrosse, M. Yoshida, D. Katoh, S. Fujii, M. Wordeman, P. Poechmueller, S. Parke, and Y. Asao, "Fault-tolerant designs for 256 Mb DRAM," in *1995 Symposium on VLSI Circuits*, June 1995, p. 107.

[28] D. Takashima, Y. Oowaki, S. Watanabe, and K. Ohuchi, "A novel power-off mode for a battery-backup DRAM," in *1995 Symposium on VLSI Circuits*, June 1995, p. 109.

[29] T. Ooishi, Y. Komiya, K. Hamada, M. Asakura, K. Yasuda, K. Furutani, T. Kato, H. Hidaka, and H. Ozaki, "A mixed-mode voltage-down converter with impedance adjustment circuitry for low-voltage wide-frequency DRAMs," in *1995 Symposium on VLSI Circuits*, June 1995, p. 111.

[30] S.-J. Lee, K.-W. Park, C.-H. Chung, J.-S. Son, K.-H. Park, S.-H. Shin, S. T. Kim, J.-D. Han, H.-J. Yoo, W.-S. Min, and K.-H. Oh, "A low noise 32bit-wide 256M synchronous DRAM with column-decoded I/O line," in *1995 Symposium on VLSI Circuits*, June 1995, p. 113.

[31] T. Sugibayashi, I. Naritake, S. Utsugi, K. Shibahara, R. Oikawa, H. Mori, S. Iwao, T. Murotani, K. Koyama, S. Fukuzawa, T. Itani, K. Kasama, T. Okuda, S. Ohya, and M. Ogawa, "A 1-Gb DRAM for file applications," *IEEE Journal of Solid-State Circuits*, vol. 30, pp. 1277–1280, November 1995.

[32] T. Yamagata, S. Tomishima, M. Tsukude, T. Tsuruda, Y. Hashizume, and K. Arimoto, "Low voltage circuit design techniques for battery-operated and/or giga-scale DRAMs," *IEEE Journal of Solid-State Circuits*, vol. 30, pp. 1183–1188, November 1995.

[33] H. Nakano, D. Takashima, K. Tsuchida, S. Shiratake, T. Inaba, M. Ohta, Y. Oowaki, S. Watanabe, K. Ohuchi, and J. Matsunaga, "A dual layer bitline DRAM array with V_{CC}/V_{SS} hybrid precharge for multi-gigabit DRAMs," in *1996 Symposium on VLSI Circuits*, June 1996, p. 190.

[34] J. Han, J. Lee, S. Yoon, S. Jeong, C. Park, I. Cho, S. Lee, and D. Seo, "Skew minimization techniques for 256M-bit synchronous DRAM and beyond," in *1996 Symposium on VLSI Circuits*, June 1996, p. 192.

[35] H. Wong, T. Kirihata, J. DeBrosse, Y. Watanabe, T. Hara, M. Yoshida, M. Wordeman, S. Fujii, Y. Asao, and B. Krsnik, "Flexible test mode design for DRAM characterization," in *1996 Symposium on VLSI Circuits*, June 1996, p. 194.

[36] D. Takashima, Y. Oowaki, S. Watanabe, K. Ohuchi, and J. Matsunaga, "Noise suppression scheme for giga-scale DRAM with hundreds of I/Os," in *1996 Symposium on VLSI Circuits*, June 1996, p. 196.

[37] S. Tomishima, F. Morishita, M. Tsukude, T. Yamagata, and K. Arimoto, "A long data retention SOI-DRAM with the body refresh function," in *1996 Symposium on VLSI Circuits*, June 1996, p. 198.

[38] M. Nakamura, T. Takahashi, T. Akiba, G. Kitsukawa, M. Morino, T. Sekiguchi, I. Asano, K. Komatsuzaki, Y. Tadaki, Songsu Cho, K. Kajigaya, T. Tachibana, and K. Sato, "A 29-ns 64-Mb DRAM with hierarchical array architecture," *IEEE Journal of Solid-State Circuits*, vol. 31, pp. 1302–1307, September 1996.

[39] K. Itoh, Y. Nakagome, S. Kimura, and T. Watanabe, "Limitations and challenges of multigigabit DRAM chip design," *IEEE Journal of Solid-State Circuits*, vol. 32, pp. 624–634, May 1997.

[40] Y. Idei, K. Shimohigashi, M. Aoki, H. Noda, H. Iwai, K. Sato, and T. Tachibana, "Dual-period self-refresh scheme for low-power DRAMs with on-chip PROM mode register," *IEEE Journal of Solid-State Circuits*, vol. 33, pp. 253–259, February 1998.

[41] K. Kim, C.-G. Hwang, and J. G. Lee, "DRAM technology perspective for gigabit era," *IEEE Transactions Electron Devices,* vol. 45, pp. 598–608, March 1998.

[42] H. Tanaka, M. Aoki, T. Sakata, S. Kimura, N. Sakashita, H. Hidaka, T. Tachibana, and K. Kimura, "A precise on-chip voltage generator for a giga-scale DRAM with a negative word-line scheme," in *1998 Symposium on VLSI Circuits*, June 1998, p. 94.

[43] T. Fujino and K. Arimoto, "Multi-Gbit-scale partially frozen (PF) NAND DRAM with SDRAM compatible interface," in *1998 Symposium on VLSI Circuits*, June 1998, p. 96.

[44] A. Yamazaki, T. Yamagata, M. Hatakenaka, A. Miyanishi, I. Hayashi, S. Tomishima, A. Mangyo, Y. Yukinari, T. Tatsumi, M. Matsumura, K. Arimoto, and M. Yamada, "A 5.3Gb/s 32Mb Embedded SDRAM core with slightly boosting scheme," in *1998 Symposium on VLSI Circuits*, June 1998, p. 100.

[45] C. Kim, K. H. Kyung, W. P. Jeong, J. S. Kim, B. S. Moon, S. M. Yim, J. W. Chai, J. H. Choi, C. K. Lee, K. H. Han, C. J. Park, H. Choi, and S. I. Cho, "A 2.5V, 2.0GByte/s packet-based SDRAM with a 1.0Gbps/pin interface," in *1998 Symposium on VLSI Circuits*, June 1998, p. 104.

[46] M. Saito, J. Ogawa, H. Tamura, S. Wakayama, H. Araki, Tsz-Shing Cheung, K. Gotoh, T. Aikawa, T. Suzuki, M. Taguchi, and T. Imamura, "500-Mb/s nonprecharged data bus for high-speed DRAMs," *IEEE Journal of Solid-State Circuits*, vol. 33, pp. 1720–1730, November 1998.

DRAM Cells

[47] C. G. Sodini and T. I. Kamins, "Enhanced capacitor for one-transistor memory cell," *IEEE Transactions Electron Devices*, vol. ED-23, pp. 1185–1187, October 1976.

[48] J. E. Leiss, P. K. Chatterjee, and T. C. Holloway, "DRAM design using the taper-isolated dynamic RAM cell," *IEEE Journal of Solid-State Circuits*, vol. 17, pp. 337–344, April 1982.

[49] K. Yamaguchi, R. Nishimura, T. Hagiwara, and H. Sunami, "Two-dimensional numerical model of memory devices with a corrugated capacitor cell structure," *IEEE Journal of Solid-State Circuits*, vol. 20, pp. 202–209, February 1985.

[50] N. C. Lu, P. E. Cottrell, W. J. Craig, S. Dash, D. L. Critchlow, R. L. Mohler, B. J. Machesney, T. H. Ning, W. P. Noble, R. M. Parent, R. E. Scheuerlein, E. J. Sprogis, and L. M. Terman, "A substrate-plate trench-capacitor (SPT) memory cell for dynamic RAMs," *IEEE Journal of Solid-State Circuits*, vol. 21, pp. 627–634, October 1986.

[51] Y. Nakagome, M. Aoki, S. Ikenaga, M. Horiguchi, S. Kimura, Y. Kawamoto, and K. Itoh, "The impact of data-line interference noise on DRAM scaling," *IEEE Journal of Solid-State Circuits*, vol. 23, pp. 1120–1127, October 1988.

[52] K. W. Kwon. I. S. Park, D. H. Han, E. S. Kim, S. T. Ahn, and M. Y. Lee, "Ta_2O_5 capacitors for 1 Gbit DRAM and beyond," in *1994 IEDM Technical Digest,* pp. 835–838.

[53] D. Takashima, S. Watanabe, H. Nakano, Y. Oowaki, and K. Ohuchi, "Open/folded bit-line arrangement for ultra-high-density DRAMs," *IEEE Journal of Solid-State Circuits*, vol. 29, pp. 539–542, April 1994.

[54] Wonchan Kim, Joongsik Kih, Gyudong Kim, Sanghun Jung, and Gijung Ahn, "An experimental high-density DRAM cell with a built-in gain stage," *IEEE Journal of Solid-State Circuits*, vol. 29, pp. 978–981, August 1994.

[55] Y. Kohyama, T. Ozaki, S. Yoshida, Y. Ishibashi, H. Nitta, S. Inoue, K. Nakamura, T. Aoyama, K. Imai, and N. Hayasaka, "A fully printable, self-aligned and planarized stacked capacitor DRAM access cell technology for 1 Gbit DRAM and beyond," in *IEEE Symposium on VLSI Technology Digest of Technical Papers,* June 1997, pp. 17–18.

[56] B. El-Kareh, G. B. Bronner, and S. E. Schuster, "The evolution of DRAM cell technology, *Solid State Technology*, vol. 40, pp. 89–101, May 1997.

[57] S. Takehiro, S. Yamauchi, M. Yoshimaru, and H. Onoda, "The simplest stacked BST capacitor for future DRAMs using a novel low temperature growth enhanced crystallization," in *IEEE Symposium on VLSI Technology Digest of Technical Papers*, June 1997, pp. 153–154.

[58] T. Okuda and T. Murotani, "A four-level storage 4-Gb DRAM," *IEEE Journal of Solid State Circuits*, vol. 32, pp. 1743–1747, November 1997.

[59] A. Nitayama, Y. Kohyama, and K. Hieda, "Future directions for DRAM memory cell technology," in *1998 IEDM Technical Digest,* pp. 355–358.

[60] K. Ono, T. Horikawa, T. Shibano, N. Mikami, T. Kuroiwa, T. Kawahara, S. Matsuno, F. Uchikawa, S. Satoh, and H. Abe, "(Ba, Sr)TiO_3 capacitor

technology for Gbit-scale DRAMs," in *1998 IEDM Technical Digest,* pp. 803–806.

[61] H. J. Levy, E. S. Daniel, and T. C. McGill, "A transistorless-current-mode static RAM architecture," *IEEE Journal of Solid-State Circuits*, vol. 33, pp. 669–672, April 1998.

DRAM Sensing

[62] N. C.-C. Lu and H. H. Chao, "Half-V_{DD}/bit-line sensing scheme in CMOS DRAMs," *IEEE Journal of Solid-State Circuits*, vol. 19, pp. 451–454, August 1984.

[63] P. A. Layman and S. G. Chamberlain, "A compact thermal noise model for the investigation of soft error rates in MOS VLSI digital circuits," *IEEE Journal of Solid-State Circuits*, vol. 24, pp. 79–89, February 1989.

[64] R. Kraus, "Analysis and reduction of sense-amplifier offset," *IEEE Journal of Solid-State Circuits*, vol. 24, pp. 1028–1033, August 1989.

[65] R. Kraus and K. Hoffmann, "Optimized sensing scheme of DRAMs," *IEEE Journal of Solid-State Circuits*, vol. 24, pp. 895–899, August 1989.

[66] H. Hidaka, Y. Matsuda, and K. Fujishima, "A divided/shared bit-line sensing scheme for ULSI DRAM cores," *IEEE Journal of Solid-State Circuits*, vol. 26, pp. 473–478, April 1991.

[67] T. Nagai, K. Numata, M. Ogihara, M. Shimizu, K. Imai, T. Hara, M. Yoshida, Y. Saito, Y. Asao, S. Sawada, and S. Fuji, "A 17-ns 4-Mb CMOS DRAM," *IEEE Journal of Solid-State Circuits*, vol 26, pp. 1538–1543, November 1991.

[68] T. N. Blalock and R. C. Jaeger, "A high-speed sensing scheme for 1T dynamic RAMs utilizing the clamped bit-line sense amplifier," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 618–625, April 1992.

[69] M. Asakura, T. Ooishi, M. Tsukude, S. Tomishima, T. Eimori, H. Hidaka, Y. Ohno, K. Arimoto, K. Fujishima, T. Nishimura, and T. Yoshihara, "An experimental 256-Mb DRAM with boosted sense-ground scheme," *IEEE Journal of Solid-State Circuits*, vol. 29, pp. 1303–1309, November 1994.

[70] T. Eirihata, S. H. Dhong, L. M. Terman, T. Sunaga, and Y. Taira, "A variable precharge voltage sensing," *IEEE Journal of Solid-State Circuits*, vol. 30, pp. 25–28, January 1995.

[71] T. Hamamoto, Y. Morooka, M. Asakura, and H. Ozaki, "Cell-plate-line/bit-line complementary sensing (CBCS) architecture for ultra low-power DRAMs," *IEEE Journal of Solid-State Circuits*, vol. 31, pp. 592–601, April 1996.

[72] T. Sunaga, "A full bit prefetch DRAM sensing circuit," *IEEE Journal of Solid-State Circuits*, vol. 31, pp. 767–772, June 1996.

DRAM On-Chip Voltage Generation

[73] M. Horiguchi, M. Aoki, J. Etoh, H. Tanaka, S. Ikenaga, K. Itoh, K. Kajigaya, H. Kotani, K. Ohshima, and T. Matsumoto, "A tunable CMOS-DRAM voltage limiter with stabilized feedback amplifier," *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 1129–1135, October 1990.

[74] D. Takashima, S. Watanabe, T. Fuse, K. Sunouchi, and T. Hara, "Low-power on-chip supply voltage conversion scheme for ultrahigh-density DRAMs," *IEEE Journal of Solid-State Circuits*, vol. 28, pp. 504–509, April 1993.

[75] T. Kuroda, K. Suzuki, S. Mita, T. Fujita, F. Yamane, F. Sano, A. Chiba, Y. Watanabe, K. Matsuda, T. Maeda, T. Sakurai, and T. Furuyama, "Variable supply-voltage scheme for low-power high-speed CMOS digital design," *IEEE Journal of Solid-State Circuits*, vol. 33, pp. 454–462, March 1998.

DRAM SOI

[76] S. Kuge, F. Morishita, T. Tsuruda, S. Tomishima, M. Tsukude, T. Yamagata, and K. Arimoto, "SOI-DRAM circuit technologies for low power high speed multigiga scale memories," *IEEE Journal of Solid-State Circuits*, vol. 31, pp. 586–591, April 1996.

[77] K. Shimomura, H. Shimano, N. Sakashita, F. Okuda, T. Oashi, Y. Yamaguchi, T. Eimori, M. Inuishi, K. Arimoto, S. Maegawa, Y. Inoue, S. Komori, and K. Kyuma, "A 1-V 46-ns 16-Mb SOI-DRAM with body control technique," *IEEE Journal of Solid-State Circuits*, vol. 32, pp. 1712–1720, November 1997.

Embedded DRAM

[78] T. Sunaga, H. Miyatake, K. Kitamura, K. Kasuya, T. Saitoh, M. Tanaka, N. Tanigaki, Y. Mori, and N. Yamasaki, "DRAM macros for ASIC chips," *IEEE Journal of Solid-State Circuits*, vol. 30, pp. 1006–1014, September 1995.

Redundancy Techniques

[79] H. L. Kalter, C. H. Stapper, J. E. Barth, Jr., J. DiLorenzo, C. E. Drake, J. A. Fifield, G. A. Kelley, Jr., S. C. Lewis, W. B. van der Hoeven, and J. A. Yankosky, "A 50-ns 16-Mb DRAM with a 10-ns data rate and on-chip ECC," *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 1118–1128, October 1990.

[80] M. Horiguchi, J. Etoh, M. Aoki, K. Itoh, and T. Matsumoto, "A flexible redundancy technique for high-density DRAMs," *IEEE Journal of Solid-State Circuits*, vol. 26, pp. 12–17, January 1991.

[81] S. Kikuda, H. Miyamoto, S. Mori, M. Niiro, and M. Yamada, "Optimized redundance selection based on failure-related yield model for 64-Mb DRAM and beyond," *IEEE Journal of Solid-State Circuits*, vol. 26, pp. 1550–1555, November 1991.

[82] T. Kirihata, Y. Watanabe, Hing Wong, J. K. DeBrosse, M. Yoshida, D. Kato, S. Fujii, M. R. Wordeman, P. Poechmueller, S. A. Parke, and Y. Asao, "Fault-tolerant designs for 256 Mb DRAM," *IEEE Journal of Solid-State Circuits*, vol. 31, pp. 558–566, April 1996.

DRAM Testing

[83] T. Ohsawa, T. Furuyama, Y. Watanabe, H. Tanaka, N. Kushiyama, K. Tsuchida, Y. Nagahama, S. Yamano, T. Tanaka, S. Shinozaki, and K. Natori, "A 60-ns 4-Mbit CMOS DRAM with built-in selftest function," *IEEE Journal of Solid-State Circuits*, vol. 22, pp. 663–668, October 1987.

[84] P. Mazumder, "Parallel testing of parametric faults in a three-dimensional dynamic random-access memory," *IEEE Journal of Solid-State Circuits*, vol. 23, pp. 933–941, August 1988.

[85] K. Arimoto, Y. Matsuda, K. Furutani, M. Tsukude, T. Ooishi, K. Mashiko, and K. Fujishima, "A speed-enhanced DRAM array architecture with embedded ECC," *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 11–17, February 1990.

[86] T. Takeshima, M. Takada, H. Koike, H. Watanabe, S. Koshimaru, K. Mitake, W. Kikuchi, T. Tanigawa, T. Murotani, K. Noda, K. Tasaka, K. Yamanaka, and K. Koyama, "A 55-ns 16-Mb DRAM with built-in self-test function using microprogram ROM," *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 903–911, August 1990.

[87] T. Kirihata, Hing Wong, J. K. DeBrosse, Y. Watanabe, T. Hara, M. Yoshida, M. R. Wordeman, S. Fujii, Y. Asao, and B. Krsnik, "Flexible test mode approach for 256-Mb DRAM," *IEEE Journal of Solid-State Circuits*, vol. 32, pp. 1525–1534, October 1997.

[88] S. Tanoi, Y. Tokunaga, T. Tanabe, K. Takahashi, A. Okada, M. Itoh, Y. Nagatomo, Y. Ohtsuki, and M. Uesugi, "On-wafer BIST of a 200-Gb/s failed-bit search for 1-Gb DRAM," *IEEE Journal of Solid-State Circuits*, vol. 32, pp. 1735–1742, November 1997.

Synchronous DRAM

[89] T. Sunaga, K. Hosokawa, Y. Nakamura, M. Ichinose, A. Moriwaki, S. Kakimi, and N. Kato, "A full bit prefetch architecture for synchronous DRAMs," *IEEE Journal of Solid-State Circuits*, vol. 30, pp. 998–1005, September 1995.

[90] T. Kirihata, M. Gall, K. Hosokawa, J.-M. Dortu, Hing Wong, P. Pfefferi, B. L. Ji, O. Weinfurtner, J. K. DeBrosse, H. Terletzki, M. Selz, W. Ellis, M. R. Wordeman, and O. Kiehl, "A 220-mm/sup2/, four-and eight-bank, 256-Mb SDRAM with single-sided stitched WL architecture," *IEEE Journal of Solid-State Circuits*, vol. 33, pp. 1711–1719, November 1998.

Low-Voltage DRAM

[91] K. Lee, C. Kim, D. Yoo, J. Sim, S. Lee, B. Moon, K. Kim, N. Kim, S. Yoo, J. Yoo, and S. Cho, "Low voltage high speed circuit designs for Giga-bit DRAMs," in *1996 Symposium on VLSI Circuits*, June 1996, p. 104.

[92] M. Saito, J. Ogawa, K. Gotoh, S. Kawashima, and H. Tamura, "Technique for controlling effective V_{TH} in multi-Gbit DRAM sense amplifier," in *1996 Symposium on VLSI Circuits*, June 1996, p. 106.

[93] K. Gotoh, J. Ogawa, M. Saito, H. Tamura, and M. Taguchi, "A 0.9 V sense-amplifier driver for high-speed Gb-scale DRAMs," in *1996 Symposium on VLSI Circuits*, June 1996, p. 108.

[94] T. Hamamoto, Y. Morooka, T. Amano, and H. Ozaki, "An efficient charge recycle and transfer pump circuit for low operating voltage DRAMs," in *1996 Symposium on VLSI Circuits*, June 1996, p. 110.

[95] T. Yamada, T. Suzuki, M. Agata, A. Fujiwara, and T. Fujita, "Capacitance coupled bus with negative delay circuit for high speed and low power (10GB/s < 500mW) synchronous DRAMs," in *1996 Symposium on VLSI Circuits*, June 1996, p. 112.

High-Speed DRAM

[96] S. Wakayama, K. Gotoh, M. Saito, H. Araki, T. S. Cheung, J. Ogawa, and H. Tamura, "10-ns row cycle DRAM using temporal data storage buffer architecture," in *1998 Symposium on VLSI Circuits*, June 1998, p. 12.

[97] Y. Kato, N. Nakaya, T. Maeda, M. Higashiho, T. Yokoyama, Y. Sugo, F. Baba, Y. Takemae, T. Miyabo, and S. Saito, "Non-precharged bit-line sensing scheme for high-speed low-power DRAMs," in *1998 Symposium on VLSI Circuits*, June 1998, p. 16.

[98] S. Utsugi, M. Hanyu, Y. Muramatsu, and T. Sugibayashi, "Non-complimentary rewriting and serial-data coding scheme for shared-sense-amplifier open-bit-line DRAMs," in *1998 Symposium on VLSI Circuits*, June 1998, p. 18.

[99] Y. Sato, T. Suzuki, T. Aikawa, S. Fujioka, W. Fujieda, H. Kobayashi, H. Ikeda, T. Nagasawa, A. Funyu, Y. Fujii, K. I. Kawasaki, M. Yamazaki, and M. Taguchi, "Fast cycle RAM (FCRAM); a 20-ns random row access,

pipe-lined operating DRAM," in *1998 Symposium on VLSI Circuits*, June 1998, p. 22.

High-Speed Memory Interface Control

[100] S.-J. Jang, S.-H. Han, C.-S. Kim, Y.-H. Jun, and H.-J. Yoo, "A compact ring delay line for high speed synchronous DRAM," in *1998 Symposium on VLSI Circuits*, June 1998, p. 60.

[101] H. Noda, M. Aoki, H. Tanaka, O. Nagashima, and H. Aoki, "An on-chip timing adjuster with sub-100-ps resolution for a high-speed DRAM interface," in *1998 Symposium on VLSI Circuits*, June 1998, p. 62.

[102] T. Sato, Y. Nishio, T. Sugano, and Y. Nakagome, "5GByte/s data transfer scheme with bit-to-bit skew control for synchronous DRAM," in *1998 Symposium on VLSI Circuits*, June 1998, p. 64.

[103] T. Yoshimura, Y. Nakase, N. Watanabe, Y. Morooka, Y. Matsuda, M. Kumanoya, and H. Hamano, "A delay-locked loop and 90-degree phase shifter for 800Mbps double data rate memories," in *1998 Symposium on VLSI Circuits*, June 1998, p. 66.

High-Performance DRAM

[104] T. Kono, T. Hamamoto, K. Mitsui, and Y. Konishi, "A precharged-capacitor-assisted sensing (PCAS) scheme with novel level controlled for low power DRAMs," in *1999 Symposium on VLSI Circuits*, June 1999, p. 123.

[105] H. Hoenigschmid, A. Frey, J. DeBrosse, T. Kirihata, G. Mueller, G. Daniel, G. Frankowsky, K. Guay, D. Hanson, L. Hsu, B. Ji, D. Netis, S. Panaroni, C. Radens, A. Reith, D. Storaska, H. Terletzki, O. Weinfurtner, J. Alsmeier, W. Weber, and M. Wordeman, "A $7F^2$ cell and bitline architecture featuring tilted array devices and penalty-free vertical BL twists for 4Gb DRAMs" in *1999 Symposium on VLSI Circuits*, June 1999, p. 125.

[106] S. Shiratake, K. Tsuchida, H. Toda, H. Kuyama, M. Wada, F. Kouno, T. Inaba, H. Akita, and K. Isobe, "A pseudo multi-bank DRAM with categorized access sequence," in *1999 Symposium on VLSI Circuits*, June 1999, p. 127.

[107] Y. Kanno, H. Mizuno, and T. Watanabe, "A DRAM system for consistently reducing CPU wait cycles," in *1999 Symposium on VLSI Circuits*, June 1999, p. 131.

[108] S. Perissakis, Y. Joo, J. Ahn, A. DeHon, and J. Wawrzynek, "Embedded DRAM for a reconfigurable array," in *1999 Symposium on VLSI Circuits*, June 1999, p. 145.

[109] T. Namekawa, S. Miyano, R. Fukuda, R. Haga, O. Wada, H. Banba, S. Takeda, K. Suda, K. Mimoto, S. Yamaguchi, T. Ohkubo, H. Takato, and K. Numata, "Dynamically shift-switched dataline redundancy suitable for DRAM macro with wide data bus," in *1999 Symposium on VLSI Circuits*, June 1999, p. 149.

[110] C. Portmann, A. Chu, N. Hays, S. Sidiropoulos, D. Stark, P. Chau, K. Donnelly, and B. Garlepp, "A multiple vendor 2.5-V DLL for 1.6-GB/s RDRAMs," in *1999 Symposium on VLSI Circuits*, June 1999. p. 153.

High-Performance Logic

[111] D. Harris, *Skew Tolerant Circuit Design*. San Francisco, CA: Morgan Kaufman Publishers, 2001.

[112] W. J. Dally and J. W. Poulton, *Digital Systems Engineering*. Cambridge, UK: Cambridge University Press, 1998.

[113] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, pp. 720–738, June, 1989.

[114] C. Svenssen and J. Yuan, "High-speed CMOS circuit technique," *IEEE JSSC*, vol. 24, pp 62–70, Feb. 1989.

[115] A. Chandrakasan, W. J. Bowhill, and F. Fox, *High-performance Microprocessor Circuits*. Piscataway, NJ: IEEE Press, 2001.

[116] I. Sutherland and S. Fairbanks, "GasP: a minimal FIFO control," *Async 2001*, pp. 46–53, March 2001.

[117] C. J. Myers, *Asynchronous Circuit Design*. Hoboken, New Jersey: John Wiley & Sons, Inc, 2001.

I/O and Timing

[118] Y. Moon, J. Choi, K. Lee, D. K. Jeong, and M. K. Kim, "An all-analog multiphase delay-locked loop using a replica delay line for wide-range operation and low-jitter performance," *IEEE Journal on Solid-State Circuits*, vol. 35, pp. 377–384, Mar. 2000.

[119] H. H. Chang, J. W. Lin, C. Y. Yang, and S. I. Liu, "A wide-range delay-locked loop with a fixed latency of one clock cycle," *IEEE Journal on Solid-State Circuits*, vol. 37, pp. 1021–1027, Aug. 2002.

[120] Yasuhiko Sasaki,Naoki Kato, and Hiroaki Nakaya, "Constant-Ratio-Coupled Multi-Grain Digital Synchronizer With Flexible Input-Output Delay Selection for Versatility in Low-Power Applications," *IEEE Journal on Solid-State Circuits*, vol. 41, May 2006.

[121] K. Minami, M. Mizuno, H. Yamaguchi, T. Nakano, Y. Matsushima, Y. Sumi, T. Sato, H. Yamashida, and M. Yamashina, "A 1 GHz portable digital delay-

locked loop with infinite phase capture ranges," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2000, vol. 469, pp. 350–351.

[122] J.-B. Lee, K.-H. Kim, C. Yoo, S. Lee, O.-G. Na, C.-Y. Lee, H.-Y. Song, J.-S. Lee, Z.-H. Lee, K.-W. Yeom, H.-J. Chung, I.-W. Seo, M.-S. Chae, Y.-H. Choi, and S.-I. Cho, "Digitally-controlled DLL and I/O circuits for 500 Mb/s/pin x16 DDR SDRAM," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2001, vol. 431, pp. 68–69.

[123] T. Matano, Y. Takai, T. Takahashi, Y. Sakito, I. Fujii, Y. Takaishi, H. Fujisawa, S. Kubouchi, S. Narui, K. Arai, M. Morino, M. Nakamura, S. Miyatake, T. Sekiguchi, and K. Koyama, "A 1-Gb/s/pin 512-Mb DDRII SDRAM using a digital DLL and a slew-rate-controlled output buffer," *IEEE Journal on Solid-State Circuits*, vol. 38, pp. 762–768, May 2003.

[124] J.-T. Kwak, C.-K. Kwon, K.-W. Kim, S.-H. Lee, and J.-S. Kih, "Low cost high performance register-controlled digital DLL for 1Gbps x32 DDR SDRAM," in *Symposium VLSI Circuits Dig. Tech. Papers*, 2003, pp. 283–284.

[125] H. H. Chang and S. I. Liu, "A wide-range and fast-locking all-digital cycle-controlled delay-locked loop," *IEEE Journal on Solid-State Circuits*, vol. 40, pp. 661–670, Mar. 2005.

[126] J. S. Wang, Y. W. Wang, C. H. Chen, and Y. C. Liu, "An ultra-low power fast-lock-in small-jitter all-digital DLL," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2005, pp. 422–433.

[127]1 S. R. Han and S. I. Liu, "A 500-MHz-1.25-GHz fast-locking pulse-width control loop with presettable duty cycle," *IEEE Journal on Solid-State Circuits,* vol. 39, pp. 463–468, March 2004.

[128] M. Mota and J. Christiansen, "A four-channel self-calibrating high-resolution time to digital converter," in *Proc. IEEE Int. Conf. Electronics*, Circuits, and Systems, 1998, pp. 409–412.

[129] Y. J. Wang, S. K. Kao, and S. I. Liu, "All-digital delay-locked loop/pulse-width-control loop with adjustable duty cycles," *IEEE Journal on Solid-State Circuits*, vol. 41, June 2006.

[130] T. Hsu, B. Shieh, and C. Lee, "An all-digital phase-locked loop (ADPLL)-based clock recovery circuit," *IEEE Journal on Solid-State Circuits*, vol. 34, pp. 1063–1073, Aug. 1999.

[131] J. Dunning, G. Garcia, J. Lundberg, and E. Nuckolls, "An all-digital phase-locked loop with 50-cycle lock time suitable for high-performance microprocessors," *IEEE Journal on Solid-State Circuits*, vol. 30, pp. 412–422, Apr. 1995.

[132] T. Saeli, K. Minami, H. Yoshida, and H. Suzuki, "A direct-skew-detect synchronous mirror delay for application-specific integrated circuits, *IEEE Journal on Solid-State Circuits*, vol. 34, pp. 372–379, Mar. 1999.

[133] K. Sung, B. D. Yang, and L.-S. Kim, "Low power clock generator based on an area-reduced interleaved synchronous mirror delay scheme," in *Proc. IEEE Int. Symposium Circuits and Systems*, 2002, pp. 671–674.

[134] Y.-H. Koh and O.-K. Kwon, "A fast-locking digital delay line with duty conservation," in *Proc. IEEE Asia-Pacific Conf. Circuits and Systems*, 1998, pp. 287–290.

[135] T. Seaeki, H. Nakamura, and J. Shimizu, "A 10 ps jitter 2 clock cycle lock time CMOS digital clock generator based on an interleaved synchronous mirror delay scheme," in *Symposium VLSI Circuits Dig. Tech. Papers*, 1997, pp. 109–110.

[136] D. Shim, D.-Y. Lee, S. Jung, C.-H. Kim, and W. Kim, "An analog synchronous mirror delay for high-speed DRAM application," *IEEE Journal on Solid-State Circuits*, vol. 34, pp. 484–493, April 1999.

[137] Y. M. Wang and J. S. Wang, "A low-power half-delay-line fast skew-compensation circuit," *IEEE Journal on Solid-State Circuits*, vol. 39, June 2004.

[138] Y. Jung, S. Lee, D. Shim, W. Kim, C. Kim and S. Cho, "A dual-loop delay-locked loop using multiple voltage-controlled delay lines," *IEEE Journal on Solid-State Circuits*, vol. 36, pp. 784–791, May 2001.

[139] J. Kim, S. Lee, T. Jung, C. Kim, S. Cho and B. Kim, "A low-jitter mixed-mode DLL for high-speed DRAM applications," *IEEE Journal on Solid-State Circuits*, vol. 35, pp. 1430–1436, Oct. 2000.

[140] G. Dehng, J. Lin, and S. Liu, "A fast-lock mixed-mode DLL using a 2-b SAR algorithm," *IEEE Journal on Solid-State Circuits*, vol. 35, pp. 1464–1471, Oct. 2001.

[141] B. Razavi, *Design of Analog CMOS Integrated Circuits*. New York: McGraw-Hill, 2001.

[142] T. H. Lee, *The Design of CMOS Radio-Frequency Integrated Circuits*. Cambridge, U.K.: Cambridge Univ. Press, 1998.

[143] T. Gabara, W. Fischer, J. Harrington, and W. Troutman, "Forming damped LRC parasitic circuits in simultaneously switched CMOS output buffers," *IEEE Journal on Solid-State Circuits*, vol. 32, pp. 407–418, March 1997.

[144] M. Dolle, "Analysis of simultaneous switching noise," in *Int. Symposium Circuits Systems*, May 1995, pp. 904–907.

[145] K. Kim et al., "A 1.4 Gb/s DLL using 2nd order charge-pump scheme with low phase/duty error for high-speed DRAM application," in IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers, 2004, pp. 212–523.

[146] S. Lee et al., "A 1.6 Gb/s/pin double data rate SDRAM with wave-pipelined CAS latency control," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, 2004, pp. 210–213.

[147] H. Fujisawa, T. Takahashi, H. Yoko, I. Fujii, Y. Takai, and M. Nakamura, "1-Gb/s/pin multi-gigabit DRAM design with low impedance hierarchical I/O architecture," in *VLSI Circuits Symposium Dig. Tech. Papers*, 2002, pp. 118–119.

[148] C. Yoo, K. Kyung, K. Lim, H.-C. Lee, J.-W. Chai, N.-W. Heo, D.-J. Lee, and C.-H Kim, "A 1.8-V 700-Mb/s/pin 512-Mb DDR-II DRAM with on-die termination and off-chip driver calibration," *IEEE Journal on Solid-State Circuits*, vol. 39, pp. 941–951, June 2004.

[149] C. Y. Chung and A. Waizman, "Optimum signal integrity through appropriate analysis of signal return path and power delivery," in *Elec. Components and Tech. Conference*, 2001, pp. 1402–1407.

[150] C. S. Choy, C. F. Chan, M. H. Ku, and J. Povazanec, "Design procedure of low-noise high-speed adaptive output drivers," in *IEEE International Symposium on Circuits and Systems*, 1997, pp. 1796–1799.

[151] S. R. Vemuru, "Effects of simultaneous switching noise on the tapered buffer design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 5, Sept. 1997.

[152] S. J. Jou, S. H. Kuo, J. T. Chiu, and T. H. Lin, "Low switching noise and load-adaptive output buffer design techniques," *IEEE Journal of Solid State Circuits*, vol. 36, Aug. 2001.

[153] L. Sumanen, M. Waltari, V. Hakkarainen, and K. Halonen, "CMOS dynamic comparator for pipeline AID converters," in *IEEE International Symposium Circuits and Systems*, ISCAS 2002, vol. 5, pp. 157–160.

[154] D. Somasekhar and K. Roy; "Differential current switch logic: a low power DCVS logic family," *IEEE Journal on Solid-State Circuits*, vol. 31, pp. 981-991, July 1996.

[155] H. Tamura et al., "5-Gb/s bidirectional balanced-line link compliant with plesiochronous clocking," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, Feb. 2001, pp. 50–51.

[156] C. H. Lim and W. R. Dassch, "Output buffer with self-adjusting slew rate and on-chip compensation," in *Proc. of IEEE Symposium on IC/Package Design Integration*, 1998, pp. 51–55.

[157] T. Shirotori and K. Nogami, "PLL-based, impedance-controlled output buffer," in *Symposium on VLSI Circuits Digest of Technical Papers*, 1991, pp. 49–50.

[158] S.-J. Jou, W.-C. Cheng, and Y.-T. Lin, "Simultaneous switching noise analysis and low bouncing buffer design," in *IEEE Conf. on Custom Integrated Circuits*, May 1998, pp. 545–548.

[159] T. Gabara, W. Fischer, J. Harrington, and W. Troutman, "Forming damped LRC parasitic circuits in simultaneously switched CMOS output buffers," in *IEEE Conf. on Custom Integrated Circuits*, May 1996, pp. 277–280.

[160] R. Senthinthan and J. L. Prince, "Application specific CMOS output driver circuit design techniques to reduce simultaneous switching noise," *IEEE Journal on Solid-State Circuits*, vol. 28, pp. 1383–1388, Dec. 1993.

[161] H. I. Hanafi, R. H. Dennard, C.-L. Chen, R. J., and D. S. Zicherman, "Design and characterization of a CMOS off-chip driver/receiver with reduced power-supply disturbance," *IEEE Journal on Solid-State Circuits*, vol. 27, pp. 783–791, 1992.

[162] E. Chioffi, F. Maloberti, G. Marchesi, and G. Torelli, "High-speed, low-switching noise CMOS memory data output buffer," *IEEE Journal on Solid-State Circuits*, vol. 29, pp. 1359–1365, 1994.

[163] A. B. Dowlatabadi, "A robust, load-insensitive pad driver," *IEEE Journal on Solid-State Circuits*, 2000, vol. 35, pp. 660–665.

[164] J. Kennedy, R. Mooney, R. Ellis, J. Jaussi, S. Borkar, J. H. Choi, J. K. Kim, C. K Kim, W.-S. Kim, C.-H. Kim, S. I. Cho, S. Loeffler, J. Hoffmann, W. Hokenmaier, R. Houghton, and T. Vogelsang, "A 3.6-Gb/s point-to-point heterogeneous-voltage-capable DRAM interface for capacity-scalable memory subsystems," *IEEE Journal on Solid-State Circuits*, vol. 40, pp. 233–244, Jan. 2005.

Glossary

1T1C A DRAM memory cell consisting of a single MOSFET access transistor and a single storage capacitor.

Bitline Also called a digitline or columnline. A common conductor made from metal or polysilicon that connects multiple memory cells together through their access transistors. The bitline is ultimately used to connect memory cells to the sense amplifier block to permit Refresh, Read, and Write operations.

Bootstrapped Driver A driver circuit that employs capacitive coupling to boot, or raise up, a capacitive node to a voltage above V_{CC}.

Buried Capacitor Cell A DRAM memory cell in which the capacitor is constructed below the digitline.

Charge Pump *See* Voltage Pump.

Clock Distribution Network, CDN Also called clock tree. A clock distribution network delivers clock from one place to multiple locations. In order to maintain signal quality and minimize timing skew, buffers are generally inserted at various points, and matched routing is also applied.

CMOS, Complementary Metal-Oxide Semiconductor A silicon technology for fabricating integrated circuits. *Complementary* refers to the technology's use of both NMOS and PMOS transistors in its construction. The PMOS transistor is used primarily to pull signals toward the positive power supply V_{DD}. The NMOS transistor is used primarily to pull signals toward ground. The *metal-oxide semiconductor* describes the sandwich of metal oxide (actually polysilicon in modern devices) and silicon that makes up the NMOS and PMOS transistors.

COB, Capacitor over Bitline A DRAM memory cell in which the capacitor is constructed above the digitline (bitline).

Columnline *See* Bitline.

Column Redundancy The practice of adding spare digitlines to a memory array so that defective digitlines can be replaced with nondefective digitlines.

Current-Mode Logic, CML A small-swing, high-speed, fully differential logic style.

DCSA, Direct Current Sense Amplifier An amplifier connected to the memory array *I/O* lines that amplifies signals coming from the array.

Delay-Chain Logic A heuristic logic style comprised of a series of delay elements that create a sequence of events. The delay elements tend to be implemented using one-shot (monostable multi-vibrator) circuits and high-load static logic chains. This type of circuitry issuited for post-silicon adjustments and is commonly found in the peripheral array logic region.

Digitline *See* Bitline.

DCL, Read Latency or *CAS* Latency

DLL, Delay-Locked Loop A circuit that generates and inserts an optimum delay to temporarily align two signals. In DRAM, a DLL synchronizes the input and output clock signals of the DRAM to the *I/O* data signals.

DRAM, Dynamic Random Access Memory A memory technology that stores information in the form of electric charge on capacitors. This technology is considered dynamic because the stored charge degrades over time due to leakage mechanisms. The leakage necessitates periodic Refresh of the memory cells to replace the lost charge.

Dummy Structure Additional circuitry or structures added to a design, most often to the memory array, that help maintain uniformity in live circuit structures. Nonuniformity occurs at the edges of repetitive structures due to photolithography and etch process limitations.

EDO DRAM, Extended Data Out DRAM A variation of fast page mode (FPM) DRAM. In EDO DRAM, the data outputs remain valid for a specified time after *CAS** goes HIGH during a Read operation. This feature permits higher system performance than would be obtained otherwise.

Efficiency A design metric, which is defined as the ratio of memory array die area and total die area (chip size). It is expressed as a percentage.

Equilibration Circuit A circuit that equalizes the voltages of a digitline pair by shorting the two digitlines together. Most often, the equilibration

circuit includes a bias network, which helps to set and hold the equilibration level to a known voltage (generally $V_{DD}/2$) prior to Sensing.

Feature Size Generally refers to the minimum realizable process dimension. In the context of DRAM design, however, feature size equates to a dimension that is half of the digitline or wordline layout pitch.

Folded DRAM Array Architecture A DRAM architecture that uses non-crosspoint-style memory arrays in which a memory cell is placed only at alternating wordline and digitline intersections. Digitline pairs, for connection to the sense amplifiers, consist of two adjacent digitlines from a single memory array. For layout efficiency, each sense amplifier connects to two adjacent memory arrays through isolation transistor pairs.

FPM, Fast Page Mode A second-generation memory technology permitting consecutive Reads from an open page of memory, in which the column address could be changed while *CAS** was still LOW.

Gate-Delay Logic In this text, gate-delay logic refers to a section of domino logic that is first made functionally complete through a conversion of static signals to dual-rail, monotonic signals. Following conversion, the logic signals are used as precharge/evaluate terms to downstream domino gates. This style of logic is used in the command decoder and address register logic to avoid clock period dependencies for data access latency.

Helper Flip-Flop, HFF A positive feedback (regenerative) circuit for amplifying the signals on the *I/O* lines.

***I/O* Devices** MOSFET transistors that connect the array digitlines to the *I/O* lines (through the sense amplifiers). Read and Write operations from/to the memory arrays always occur through *I/O* devices.

Intrinsic Delay Also called intrinsic forward path delay. The intrinsic delay represents a propagation delay of the critical timing path from inputs (e.g., *CLK*) to outputs (e.g., *DQ*s), excluding the insertion delay from timing adjustment circuits (e.g., DLL).

Isolation Devices MOSFET transistors that isolate array digitlines from the sense amplifiers.

Matched Routing A method of maintaining an original timing relationship after signal distribution and placement.

Mbit, Memory Bit A memory cell capable of storing one bit of data. In modern DRAMs, the mbit consists of a single MOSFET access transistor and a single storage capacitor. The gate of the MOSFET connects to the wordline or rowline, while the source and drain of the MOSFET connect to the storage capacitor and the digitline, respectively.

Memory Array An array of memory or mbit cells.

Multiplexed Addressing The practice of using the same chip address pins for both the row and column addresses. The addresses are clocked into the device at different times.

On-Die Termination,ODT The impedance put on DRAM high-speed *I/O* to reduce reflections and improve signal integrity. The termination value depends on the system configuration and channel impedance.

Open DRAM Array Architecture A DRAM architecture that uses cross-point-style memory arrays in which a memory cell is placed at every wordline and digitline intersection. Digitline pairs, for connection to the sense amplifiers, consist of a single digitline from two adjacent memory arrays.

PDN, Power Delivery Network. The system of wiring and decoupling elements connects individual transistors and devices to the system power supply. In DRAM, the basic PDN includes on-die power bussing, decoupling capacitors, and the device package power bussing. A more complete PDN description would also include the DIMM, DIMM connector, mother board, and related decoupling capacitors.

Peripheral Array Logic This section of logic is part of the helper flip-flop (HFF) control and column timing chain logic. Often this logic is implemented using the "delay-chain" logic style described in this text.

Peripheral Logic Interface All of the logic necessary for implementing the system level protocol. Generally, this logic resides between the *I/O* circuits and the peripheral array logic.

Pitch The distance between like points in a periodic array. For example, digitline pitch in a DRAM array is the distance between the centers or edges of two adjacent digitlines.

RAM, Random Access Memory Computer memory that allows access to any memory location without restrictions.

Read Pipe The partial multiplexor circuit following the HFF logic. This logic acts as a FIFO to match the latency of array data accesses with the programmed latency and throughput requirements of the output circuit path. There are often data reordering functions as part of the logic in this circuit or set of circuits.

Refresh The process of restoring the electric charge in DRAM memory cell capacitors to full levels through Sensing. Note that Refresh occurs every time a wordline is activated and the sense amplifiers are fired.

Rowline *See* Wordline.

SDRAM, Synchronous DRAM A DRAM technology in which addressing, command, control, and data operations are accomplished in synchronism with a master clock signal.

Sense Amplifier A type of regenerative amplifier that senses the contents of memory cells and restores them to full levels.

Signal Integrity The quality of signal transportation during circuit operation.

Signal Return Path Shift, SRPS Return path shift may be caused when a signal's reference plane changes during transmission due to being pulled up or down by drivers at the near or far end.

Simultaneous Switching Noise, SSN *I/O* power supply noises generated by simultaneously switching output drivers.

Source Synchronous Clock edges transmitted with the data used to launch it are available at the receive end for capture.

Trench Capacitor A DRAM storage capacitor fabricated in a deep hole (trench) in the semiconductor substrate.

Voltage-Controlled Delay Line, *VCDL* An adjustable delay chain in which the delay of each stage is controlled by adjusting a bias voltage. VCDL is generally found in analog delay-locked or phase-locked loops.

Voltage Pump Also called a charge pump. A circuit for generating voltages that lie outside of the power supply range.

WL, Write Latency

Wordline Also called a rowline. A wordline is a polysilicon conductor for forming memory cell access transistor gates and connecting multiple memory cells into a physical row. Driving a wordline HIGH activates, or turns ON, the access transistors in a memory array row.

Write Pipe The partial demultiplexor circuit following the write data captures latches. This logic matches the data rate of the input circuit path with the column cycle time of the array data path in the peripheral logic interface. The demultiplexor is also commonly referred to as a serial-to-parallel converter.

Index

Numerics

1T1C memory cell, 10, 22–23, 33

3-transistor DRAM cell, 6–7

$6F^2$, 36, 65–66, 70, 75, 84–86, 89

$8F^2$, 35–36, 66, 75, 77–79

A

Access time (t_{AC}), 4, 18, 68, 126, 135, 169, 245, 301–303, 313, 324

Access time

- from *CAS** (t_{CAC}), 14, 169, 301, 303
- from column access (t_{AA}), 169, 347, 349–350, 353, 355–356
- from *RAS** (t_{RAC}), 107, 169

ACT. See Active pull-up

Active pull-up (ACT), 26–30, 48–53, 157

Active to Read or Write delay (t_{RCD}), 169, 302

Address

- decode trees, 54, 58–60, 62, 88, 99, 102, 131
- multiplexing, 3, 7–8, 126, 180–181
- path elements, 14, 126–130, 131–134
- predecode, 60–61, 100, 105, 107–108, 126, 128–134
- transition detection, 131–133

Address register logic, 316, 324–328

Address to Write delay time (t_{AW}), 5

Alignment

- strobe, 195, 207, 333, 342–345
- clock and data, 194–195, 207–211, 213, 216, 252, 337–338, 345, 352–354, 359–362
- phase, 281, 352, 361

Arbiter, 142–143, 272–273

Array, 2–4, 9

- architecture, 31, 35, 65–98
- architecture, comparison, 93–97
- efficiency, 70–73, 76, 79–80, 93–94

B

Bandgap reference, 153–154, 217

Bandwidth, 154, 175–176, 182–183, 243, 246, 248, 292, 296–297, 304, 318, 323–324, 346, 368, 385–387

Binning, 170, 314

Bias circuits, 44–46, 51, 113–114, 118–120, 153, 197, 278–279, 334

Bilevel
digitline, 83–97
digitline array architecture, 88–93
Bitline, 9
capacitance number, limitations, 10
PRECHARGE voltage, 24, 45
Bitline over capacitor (BOC), 33
BOC. See Bitline over capacitor
BOOST signal, 155–161
Bootstrap wordline driver, 54–58, 60–61, 73
Breakdown voltages, 147
Bundled signal protocol, 328
Buried capacitor, 33, 39–42
Buried digitline, 41–42
Burn-in, 149–152, 161, 242, 260
Burst length, 21, 171, 176, 219, 297, 303, 322, 326, 329

C

Capacitor
buried, 33, 39–42
trench, 42–44
Capacitor over bitline (COB), 41
Capacity, 159
Capture latch, 193, 203, 209, 288, 312–313, 320, 324–325, 338–339, 365, 371
CAS. See Column address strobe
CAS latency (CL), 20, 169, 219, 245, 302–303, 312–313
C_{digit}, digitline capacitance, 25
Charge pump, 147, 158
Charge-pump phase detector (CPPD), 253, 283
C_{IN}, input capacitance, 2
CL. See *CAS* latency
Clock cycle time (t_{CK}), 169, 187, 200, 203, 210, 255, 262–263, 268–269, 277, 281–282, 301–303, 327, 355, 361, 364
Clock (data) recovery circuit (CRC, CDR), 208–209, 252, 292
Clock distribution network (CDN), 195, 202, 208–209, 213, 231, 251, 257, 289–291, 322, 371
Cmbit, Mbit capacitance, 26
CML. See Current mode logic
COB. See Capacitor over bitline
Co-design, 373, 376, 382
Column address path, 14, 126, 131, 175, 187
decoder, 4, 13–14, 325
redundancy, 60, 99–104, 107, 131, 178, 188
Column address strobe (*CAS*), 3, 8, 13–17, 29, 126, 169, 301, 311, 320
Column cycle time (t_{CCD}), 170, 175–176, 180, 296–297, 303, 322, 326, 331, 348, 355, 357, 372, 385
Column hold time (t_{CAH}), 8
Column decoder, 30, 99–102, 133, 180, 189–190, 195, 306, 318
Column select (*CSEL*), 28, 47, 51, 87, 100–102, 118, 133, 177–182, 189
Column setup time (t_{ASC}), 8
Columnline, 6–7, 10, 13–14, 22
Conversion
dual-rail, 320, 324
CML-to-CMOS, (CML2CMOS), 283, 285
parallel-to-serial (P2S), 188, 219, 244, 346

serial-to-parallel (S2P), 330–331, 339
signal, 196–197, 217
static-to-dynamic, 312, 320
time-to-digital. See Measure-controlled delay
voltage, 57
voltage-to-time (VTC), 227
CPPD. See Charge-pump phase detector
CSEL. See Column select
Current mode logic (CML), 193, 197–198, 212, 247–248, 277–278, 282, 371–372

D

Data-bus inversion (DBI), 194, 236–237
Data eye and diagram, 196, 208–209, 222–223, 237, 239–241, 332
Data input buffer, 111–112, 127
Data path elements, 111
Data Read
muxes, 122, 125
path, 118
DC sense amplifier (DCSA), 111, 118–121, 125, 176, 319
DCD and DCC. See Duty-cycle distortion and correction
DCSA. See DC sense amplifier
DDR. See Double data rate
DDR2, 31, 171, 175, 180, 185, 221–222, 235, 243, 245, 296
DDR3, 31, 177, 185, 194
Decoupling capacitor, 231, 238–241, 243, 258, 374
Delay-chain logic, 306–308, 329
Delay–locked loop (DLL), 206, 227, 251, 337, 359–362, 371
all-digital, 253, 254
analog, 276, 286–290
mixed-mode, 254, 276, 285–286, 289–291
register-controlled, 135–145
Delay variations, 196–197, 202, 258, 262, 267, 282, 286, 298, 304, 308, 349, 359, 361
Demultiplexor, Write data, 330, 338–346
Design examples, 79, 83–97, 150, 350
De-skewing, 195, 207, 251, 253, 287–288
Deterministic latency, 188, 333, 346, 358
Digital filter, 226–227, 269, 272
Digitline, 9–10, 22, 33
capacitance (C_{digit}), 23, 25, 39, 46, 66–68, 91
construction, 42
open digitline array layout, 24, 39–40, 67–80, 84–86, 89, 92–96
pair, 24, 26, 36, 44–47, 77, 85, 89
twist, 36–37, 76
voltages, 25
Discontinuity, 261, 267, 274
DLL. See Delay-locked loop
Domino logic, 309–314
Donut gate, 56
Double data rate (DDR), 14, 16–17, 31, 124, 135–137, 171, 180,185, 193, 221, 249, 252, 269, 296, 303, 322, 332–335, 351, 359, 367
DQ, 13, 17–18, 31, 52, 124–125,

DQ, continued, 135–137, 175, 184–185, 209, 230–233, 239, 245, 255, 351
mask (*DQM*), 18–19
strobe (*DQS*), 16, 135–137, 194, 245
DQM. See *DQ* mask
DQS. See *DQ* strobe
DQS-DQ skew, (t_{DQSQ}), 245
Dual-rail logic, 312
Duty-cycle distortion and correction, 113, 124, 194, 208, 211, 217, 251–252, 260, 278, 289–291, 344
Dual-loop, 274, 286, 289
DVC2, 147, 165–166
Dynamic random access memory (DRAM), 1, 13
1T1C memory cell, 10, 22–23, 33
3–transistor cell, 6–7
address multiplexing, 3, 7–8, 126, 301
array, 2–4, 9, 33
evolution, 1, 170–171, 249, 301
first generation, 1
internal V_{CC}, 91, 147–151
layout, 24, 31–33, 35, 39, 41, 46
Mbit, 22–23
modes of operation, 7, 13–14, 20, 301
opening a row, 10, 11–13, 29–30
organization, 12
power consumption, 45–46
power dissipation, 39, 48, 67–68, 77, 90–91, 385–386, 389
reading data out, 4–6, 135
Refresh, 5–6, 10, 12–13, 48–49, 67, 77, 91, 128–129, 387
second generation, 7, 9, 13, 17
synchronization, 135–145, 171, 187, 193, 245, 251–289, 346, 357
synchronous, 15, 99–100, 107, 169–171, 174, 193, 259, 298, 301, 308, 319
test modes, 125–126, 273
testing, 52, 314–315
types and operation, 1–21

E

EDO. See Extended data out
Edge recovery, 254, 276, 285, 289
EQ. See Equilibrate
Equilibrate *(EQ)*, 44–45
Equilibration, 44–46, 177
Electrostatic discharge (ESD), 124, 200, 219, 242–244
Extended data out (EDO), 7, 14, 20, 99–100, 107, 126, 170, 301, 303

F

Fast page mode (FPM), 7, 14, 20, 99–100, 107, 170
FBGA. See Fine ball grid array
FCIP. See Four-layer flip chip
FIFO
asynchronous, 327–332, 357
orthogonal, 352–358
Fine ball grid array (FBGA), 231–232, 235
Folded array, 31, 35–36
architecture, 77–85
Folded digitline architectures, 77, 85
Four-layer flip chip (FCIP), 373, 376, 381

FPM. See Fast page mode
Frequency slewing, 262
Fully differential amplifier, 113, 197

G

Gate-delayed logic, 320
GDDR. See Graphics double data rate
Graphics double data rate, 171
GDDR2, 171
GDDR3, 171, 174, 176–177, 184–189, 203, 228, 238, 303–305, 355, 372–382
GDDR4, 171, 175, 177, 184, 222, 231–232, 245, 252, 297, 305, 357
Global
circuitry, 111–144
column select lines, 72, 178, 189
row decoders, 54, 57, 59, 70, 79, 88, 90, 93, 101

H

Helper flip-flop (HFF), 31, 118–119, 121–122, 176–183, 306, 319, 325, 372
HFF. See Helper flip-flop
High-input trip point (V_{IH}), 2, 112, 127, 194

I

I/O, 16–18, 28, 44, 116–124, 135, 193, 219, 248
I/O transistors, 28, 31, 46–53, 100–101, 111, 116, 118, 176–178, 306
Impedance
calibration, 219, 224–230, 243
characteristic, 220–221, 233
high-impedance (tri-state), 116, 125, 246
transmission line, 112, 220–223
Input buffer, 1, 111–115, 127–128, 132, 136, 197, 255, 258, 371
Input capacitance (C_{IN}), 2
Input phase selection, 262–265, 281, 285
Input timing adjustment, 193, 206–210
Interpolation, 259, 266–267, 277, 286–288, 292
Inter-symbol interference (ISI), 194, 208, 221, 249, 252, 333
Intrinsic delay, 255–257, 262–263, 266, 272, 286, 290–291
Isolation, 42, 44, 46, 48–51, 76, 87, 89–90, 155, 254, 258

J

JEDEC, 184, 217
Jitter, 140–141, 194–196, 208–211, 233–234, 237, 241, 245, 251–253, 266, 271–272, 282, 288, 333, 360

L

Latency, 18, 175, 187–189
Leakage, 22, 34, 40, 45, 48–49, 56, 58, 105, 147, 246, 299, 309–310, 387
Linearity, 225, 229, 236, 277–278
Lock time, 140, 252–253, 264, 268–270, 284–285, 289–291
Loop filter, 272, 277–278, 283–284
Low-input trip point (V_{IL}), 2, 112, 127, 194

M

Matched routing, 193, 195, 200–203, 207–208, 234, 245, 315, 320, 323, 335, 365
Mbit, 22
$6F^2$, 38, 65–66, 70, 75, 84–86, 89

Mbit, continued
8F², 35–36, 66, 75, 77–79
capacitance (Cmbit), 25
layout, 33–34
Mbit pair layout, 34
Measure-controlled delay (MCD), 253, 270–271, 285
Memory element, 2, 4, 10–11
Memory array, 2, 4, 9–12, 23, 171, 180–183, 190, 301
layout, 24
size, 4, 12
Monotonic signal, 310, 320, 323, 325
Multiplexed addressing, 3, 8–9, 126
Multiplexor, Read data, 246–359

N

N sense amp latch (*NLAT**), 26–30, 48–53
Negative bias temperature instability (NBTI), 260
Nibble mode, 7, 13–15
*NLAT**. See N sense-amp latch

O

On-die termination (ODT), 194, 221–223, 228, 236, 243, 246
ONO dielectric. See Oxide-nitride-oxide dielectric
Operational amplifiers (op-amp), 229, 278
CMOS, 150
power, 152, 154–156
standby, 154, 156
Open architectures, 65–75
Open array, 31, 38–39, 79
Opening a row, 10–13, 29–30
Output driver, 124–125, 175, 183, 187, 219, 224, 228–230, 234–235, 243, 245–249
Oscillator circuits, 159
Output buffer circuit, 30, 122, 124–125, 242, 283
Oxide-nitride-oxide (ONO) dielectric, 40–43

P

Page mode, 7, 13–15, 99–100, 126, 132, 169–170, 301–302, 304, 315
Page size, 11–12
PD. See Phase detector
PDN. See Power delivery network
Peripheral array logic, 295–296, 308–309, 318–319, 325–355
Peripheral circuitry, 99–100
Peripheral logic interface, 295–367
Phase detector (PD), 136–143, 227, 254, 263, 267–273, 275, 278, 284, 337, 359, 362
Phase drivers, 126, 131
Phase generation, 264, 277, 285, 289–290
Phase-locked loop (PLL), 206, 208–210, 252–253, 292, 337–338, 359, 371
Pipeline, 20, 100, 183, 188, 297, 327, 371
Pitch, 34–35, 78, 116
cells, 46
digitline, 34, 39, 68, 70, 78–79
wordline, 70, 79, 84, 88–93
PLL. See Phase-locked loop
PNP transistors, 153
Polycide, 11, 23, 40

Power
consumption, 45–46, 58, 140, 200, 213, 259, 261, 291, 334
dissipation, 39, 48, 67–68, 77, 90–91, 140, 259–260, 264, 372, 385–386, 389
Power delivery network, (PDN), 372–376, 380, 382
POWERDOWN, 148
POWERUP, 107, 138–139, 148–150
PRECHARGE, 19, 24, 45, 105, 204, 309
Predecode logic, 126–134
Predecoding, 60, 128
Pre-emphasis, 248–249
Prefetch, 195, 297, 300, 326, 330, 332, 346, 351
Pumps and generators, 124, 147, 158–163

R

R/W, 3–6, 13–15, 329
*RAS**. See Row address strobe
*RAS** chain, 60, 295
Rate of activation, 49
RDL. See Redistribution layer
RDLL. See Register-controlled delay locked loop
Read cycle time (t_{RC}), 4, 8, 12
Read latency, 175, 187–188, 330, 350, 353–355, 359–367
Read-Modify-Write, 13–14, 52–53, 119
Read or Write cycle time (t_{PC}), 169, 301
Reads, nonconsecutive, 352
Read pipes, 346, 350–359
Read strobe (*DQS*) access time from *CLK/CLK** (t_{DQSCK}), 245
Receiver, also see Input buffer
clock, 195, 199, 210, 216, 255, 287, 292, 371
command/address (C/A), 175, 187, 202, 227
data, 197, 209, 333–334
differential, 114, 197–199, 323
equalization, 249
ESD. See Electrostatic discharge
input, 113–114, 196–200, 205, 209, 312, 334
ODT. See On-die termination
offset, 200, 205, 226, 300
V_{REF}, 113, 152, 193, 196–197, 199
wide-swing, 199
Redistribution layer, 373, 376, 378
Reduced latency DRAM (RLDRAM), 188–190, 318
Redundancy, 60, 99–109, 126, 131–132, 178, 187
Refresh, 5–6, 10, 12–13, 19–20, 26, 30, 46, 48–49, 67, 77, 91, 107, 128–129, 387
Register-controlled delay-locked loop (RDLL), 135–142
Retiming, 193, 313–314, 336–338, 340–341, 363–364
RLDRAM. See Reduced latency DRAM
Row address strobe (*RAS**), 3, 8–19, 29–30, 44, 60, 99, 104–105, 107–108, 118, 120, 126–128, 132, 155–157, 169, 295, 301, 311–312, 317, 319–321, 324
Row decode, 36, 54, 57, 59, 65, 69, 71–78, 83, 86, 88–97

Row hold time (t_{RAH}), 8
Row redundancy, 102–109, 131
Row setup time (t_{ASR}), 8
Rowline, 6–7, 9–10, 22–23, 30

S

Self-bias, 113–114, 197, 278–281
Sense amplifier, 26–29, 36–38, 44–49, 69–76, 90–96, 177–182, 205
Serializer, 183, 350–351, 355–356, 358, 366
Signal return path shift (SRPS), 238–242
Silicide, 11, 23, 54
Simultaneous-switching noise, 194, 219, 230–237, 248
Skew-tolerant logic, 298, 311
Skin effect, 221, 248
Slew rate, 155, 196, 219–220, 222, 234, 237, 245, 254, 260, 267
SMD. See Synchronous mirror delay
Source synchronous (SS), 193–195, 200, 208–210, 219, 315, 332, 336, 359
Speed grade, 170, 259, 291, 305–306
SRPS. See Signal return path shift
SSN, SSO. See Simultaneous-switching noise
SSTL. See Stub series terminated logic
Stability, 47, 51, 118, 154, 165, 253, 268, 270–271, 284–285, 290
Static column mode, 7, 13–14
Stub series terminated logic (SSTL), 112–113, 222–223
Subarray, 104, 107, 189–190
Subthreshold leakage, 48–49, 58
Successive-approximation register (SAR), 226–227, 270
Supply noise, 194, 208, 230, 238, 258, 265, 278, 282–283, 287–291, 337, 371, 375, 381
Symmetrical delay line, 136–141, 261
Sync point. See Synchronization points
Synchronization in DRAMs, 135–145, 171, 187, 193, 245, 251–289, 346, 357
Synchronous
  DDR, double data rate, 14, 16–17, 31, 124, 135–137, 171, 180, 185, 193, 221, 249, 252, 269, 296, 303, 322, 332–335, 351, 359, 367
  DRAM (SDRAM), 15–20, 135, 170–171, 176, 180, 185, 187, 193, 269, 301–303, 332, 359, 367
Synchronous mirror delay (SMD), 135, 253, 270
Synchronization points (SP), 187, 363–364

T

t_{AA}. See Access time from column access
t_{AC}. See Access time
t_{ASC}. See Column setup time
t_{ASR}. See Row setup time
t_{AW}. See Address to Write delay time
t_{CAC}. See Access time from *CAS**
t_{CAH}. See Column hold time
t_{CCD}. See Column cycle time
t_{CK}. See Clock cycle time
t_{DQSCK}. See Read strobe (*DQS*) access time from *CLK/CLK**
t_{DQSQ}. See *DQS-DQ* skew

t_{DQSS}. See Write command to first *DQS* latching transition
Test modes, 125–126, 273
t_{PC}. See Read or Write cycle time
t_{RAC}. See Access time from *RAS**
Tracking
latency, 340, 348, 352, 357, 360, 366–374
timing, 188, 266, 271–274, 285, 308
voltage and temperature (VT), 253, 263, 274, 286, 291
t_{RAH}. See Row hold time
Training, 195, 207, 209, 252, 288, 292, 337
Transistor-transistor logic (TTL), 6–7, 112
t_{RC}. See Read cycle time
t_{RCD}. See Active to Read or Write delay
t_{RP}, 302
Trench capacitor, 42–44
TTL. See Transistor-transistor logic
t_{WC}. See Write cycle time
Twisting, 36–38, 68, 76–77, 85–90
t_{WP}. See Write pulse width

U

Unit delay, 136–138, 256, 259–262, 266–267, 271–272, 275, 277–281
Unit interval (*UI*), 209–210, 341

V

V_{BB} pump, 158–160
V_{CCP}, 10, 13, 24, 28, 30, 55–58, 101, 147, 157–163, 200
V_{IH}. See High–input trip point
V_{IL}. See Low–input trip point voltage
Voltage-controlled delay line (VCDL), 253, 276–287

W

Wordline, 9–13, 22–26, 30–31, 33–35
Write command to first *DQS* latching transition, (t_{DQSS}), 206–207, 252, 287, 330, 333–334, 341, 347
Write cycle time (t_{WC}), 5
Write driver, 28, 47, 112, 115–118, 306, 329
Write latency, 195, 206, 287, 317, 326, 330–338, 341, 343, 345
Write leveling, 207
Writes, nonconsecutive, 342–345
Write operation waveforms, 28–29
Write pipes, 330, 346–347
Write pulse width (t_{WP}), 5

Y

Yield, 1, 34, 51, 54, 66, 86, 93, 102, 104–105, 125, 200, 243, 298, 350

Z

ZQ calibration. See Impedance calibration