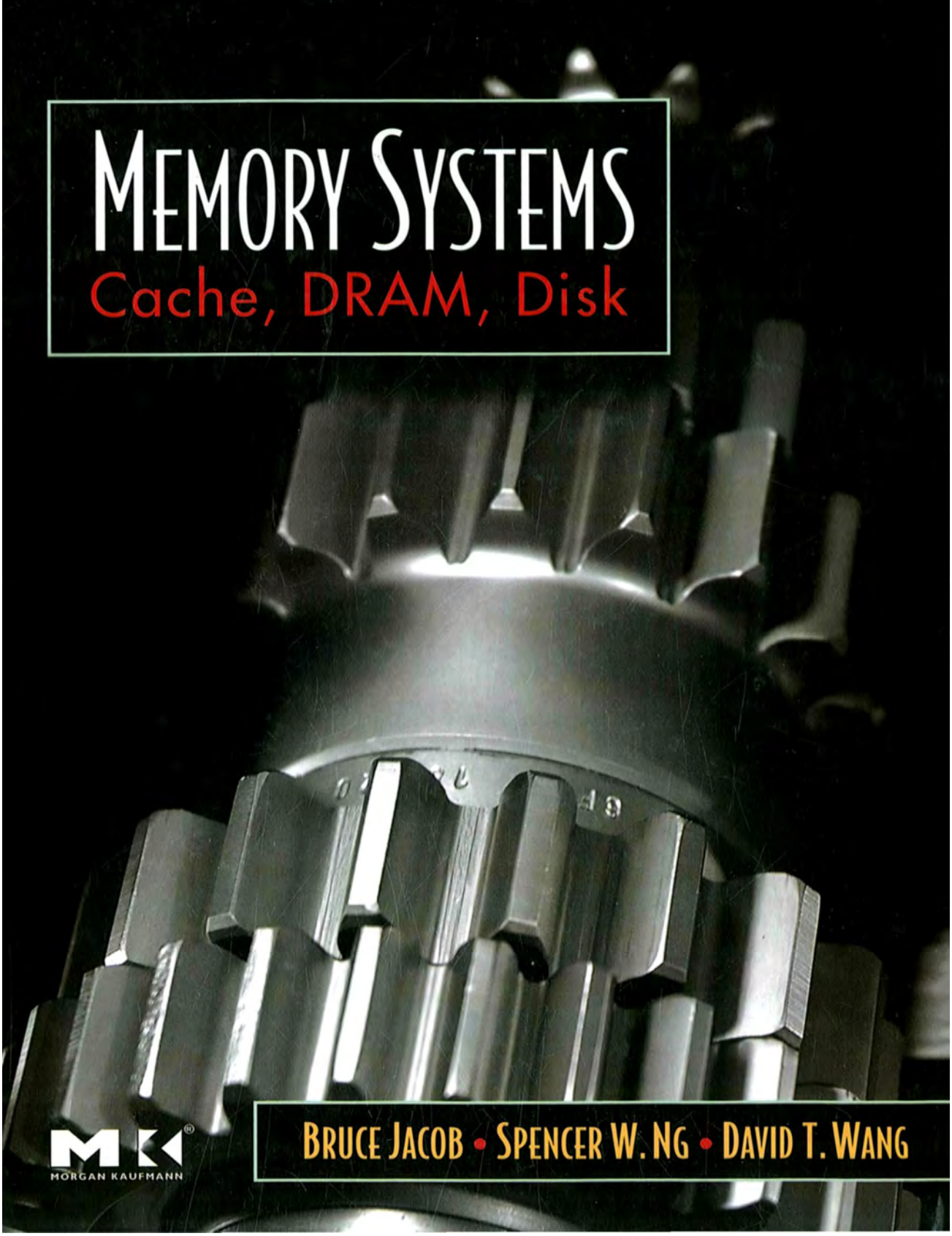


In Praise of *Memory Systems: Cache, DRAM, Disk*

Memory Systems: Cache, DRAM, Disk *is the first book that takes on the whole hierarchy in a way that is consistent, covers the complete memory hierarchy, and treats each aspect in significant detail. This book will serve as a definitive reference manual for the expert designer, yet it is so complete that it can be read by a relative novice to the computer design space. While memory technologies improve in terms of density and performance, and new memory device technologies provide additional properties as design options, the principles and methodology presented in this amazingly complete treatise will remain useful for decades. I only wish that a book like this had been available when I started out more than three decades ago. It truly is a landmark publication. Kudos to the authors.*

—Al Davis, University of Utah

Memory Systems: Cache, DRAM, Disk *fills a huge void in the literature about modern computer architecture. The book starts by providing a high level overview and building a solid knowledge basis and then provides the details for a deep understanding of essentially all aspects of modern computer memory systems including architectural considerations that are put in perspective with cost, performance and power considerations. In addition, the historical background and politics leading to one or the other implementation are revealed. Overall, Jacob, Ng, and Wang have created one of the truly great technology books that turns reading about bits and bytes into an exciting journey towards understanding technology.*

—Michael Schuette, Ph.D., VP of Technology Development at OCZ Technology

This book is a critical resource for anyone wanting to know how DRAM, cache, and hard drives really work. It describes the implementation issues, timing constraints, and trade-offs involved in past, present, and future designs. The text is exceedingly well-written, beginning with high-level analysis and proceeding to incredible detail only for those who need it. It includes many graphs that give the reader both explanation and intuition. This will be an invaluable resource for graduate students wanting to study these areas, implementers, designers, and professors.

—Diana Franklin, California Polytechnic University, San Luis Obispo

Memory Systems: Cache, DRAM, Disk *fills an important gap in exploring modern disk technology with accuracy, lucidity, and authority. The details provided would only be known to a researcher who has also contributed in the development phase. I recommend this comprehensive book to engineers, graduate students, and researchers in the storage area, since details provided in computer architecture textbooks are woefully inadequate.*

—Alexander Thomasian, IEEE Fellow, New Jersey Institute of Technology and Thomasian and Associates

Memory Systems: Cache, DRAM, Disk *offers a valuable state of the art information in memory systems that can only be gained through years of working in advanced industry and research. It is about time that we have such a good reference in an important field for researchers, educators and engineers.*

—Nagi Mekhiel, Department of Electrical and Computer Engineering, Ryerson University, Toronto

This is the only book covering the important DRAM and disk technologies in detail. Clear, comprehensive, and authoritative, I have been waiting for such a book for long time.

—Yiming Hu, University of Cincinnati

Memory is often perceived as the performance bottleneck in computing architectures. Memory Systems: Cache, DRAM, Disk, *sheds light on the mystical area of memory system design with a no-nonsense approach to what matters and how it affects performance. From historical discussions to modern case study examples this book is certain to become as ubiquitous and used as the other Morgan Kaufmann classic textbooks in computer engineering including Hennessy and Patterson's* Computer Architecture: A Quantitative Approach.

—R. Jacob Baker, Micron Technology, Inc. and Boise State University.

Memory Systems: Cache, DRAM, Disk *is a remarkable book that fills a very large void. The book is remarkable in both its scope and depth. It ranges from high performance cache memories to disk systems. It spans circuit design to system architecture in a clear, cohesive manner. It is the memory architecture that defines modern computer systems, after all. Yet, memory systems are often considered as an appendage and are covered in a piecemeal fashion. This book recognizes that memory systems are the heart and soul of modern computer systems and takes a 'holistic' approach to describing and analyzing memory systems.*

The classic book on memory systems was written by Dick Matick of IBM over thirty years ago. So not only does this book fill a void, it is a long-standing void. It carries on the tradition of Dick Matick's book extremely well, and it will doubtless be the definitive reference for students and designers of memory systems for many years to come. Furthermore, it would be easy to build a top-notch memory systems course around this book. The authors clearly and succinctly describe the important issues in an easy-to-read manner. And the figures and graphs are really great—one of the best parts of the book.

When I work at home, I make coffee in a little stove-top espresso maker I got in Spain. It makes good coffee very efficiently, but if you put it on the stove and forget it's there, bad things happen—smoke, melted gasket—'burned coffee meltdown.' This only happens when I'm totally engrossed in a paper or article. Today, for the first time, it happened twice in a row—while I was reading the final version of this book.

—Jim Smith, University of Wisconsin—Madison

Memory Systems

Cache, DRAM, Disk

Memory Systems

Cache, DRAM, Disk

Bruce Jacob
University of Maryland at College Park

Spencer W. Ng
Hitachi Global Storage Technologies

David T. Wang
MetaRAM

With Contributions By

Samuel Rodriguez
Advanced Micro Devices

AMSTERDAM • BOSTON • HEIDELBERG LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO
Morgan Kaufmann is an imprint of Elsevier

MORGAN KAUFMANN PUBLISHERS

Publisher	Denise E.M. Penrose
Acquisitions Editor	Chuck Glaser
Publishing Services Manager	George Morrison
Senior Production Editor	Paul Gottehrer
Developmental Editor	Nate McFadden
Assistant Editor	Kimberlee Honjo
Cover Design	Joanne Blank
Text Design	Dennis Schaefer
Composition	diacriTech
Interior printer	Maple-Vail Book Manufacturing Group
Cover printer	Phoenix Color

Morgan Kaufmann Publishers is an imprint of Elsevier.
30 Corporate Drive, Suite 400, Burlington, MA 01803, USA

This book is printed on acid-free paper.

Library of Congress Cataloging-in-Publication Data
Application submitted

ISBN: 978-0-12-379751-3

For information on all Morgan Kaufmann publications,
visit our Web site at *www.mkp.com* or *www.books.elsevier.com*

Printed in the United States of America
08 09 10 11 12 5 4 3 2 1

Dedication

Jacob *To my parents, Bruce and Ann Jacob, my wife, Dorinda, and my children, Garrett, Carolyn, and Nate*

Ng *Dedicated to the memory of my parents Ching-Sum and Yuk-Ching Ng*

Wang *Dedicated to my parents Tu-Sheng Wang and Hsin-Hsin Wang*

You can tell whether a person plays or not by the way he carries the instrument, whether it means something to him or not.

Then the way they talk and act. If they act too hip, you know they can't play [jack].

—Miles Davis

[...] in connection with musical continuity, Cowell remarked at the New School before a concert of works by Christian Wolff, Earle Brown, Morton Feldman, and myself, that here were four composers who were getting rid of glue. That is: Where people had felt the necessity to stick sounds together to make a continuity, we four felt the opposite necessity to get rid of the glue so that sounds would be themselves.

Christian Wolff was the first to do this. He wrote some pieces vertically on the page but recommended their being played horizontally left to right, as is conventional. Later he discovered other geometrical means for freeing his music of intentional continuity. Morton Feldman divided pitches into three areas, high, middle, and low, and established a time unit. Writing on graph paper, he simply inscribed numbers of tones to be played at any time within specified periods of time.

There are people who say, "If music's that easy to write, I could do it." Of course they could, but they don't. I find Feldman's own statement more affirmative. We were driving back from some place in New England where a concert had been given. He is a large man and falls asleep easily. Out of a sound sleep, he awoke to say, "Now that things are so simple, there's so much to do." And then he went back to sleep.

—John Cage, *Silence*

Contents

Preface "It's the Memory, Stupid!" xxxi

Overview On Memory Systems and Their Design 1

Ov.1 Memory Systems 2

Ov.1.1 Locality of Reference Breeds the Memory Hierarchy 2

Ov.1.2 Important Figures of Merit 7

Ov.1.3 The Goal of a Memory Hierarchy 10

Ov.2 Four Anecdotes on Modular Design 14

Ov.2.1 Anecdote I: Systemic Behaviors Exist 15

Ov.2.2 Anecdote II: The DLL in DDR SDRAM 17

Ov.2.3 Anecdote III: A Catch-22 in the Search for Bandwidth 18

Ov.2.4 Anecdote IV: Proposals to Exploit Variability in Cell Leakage 19

Ov.2.5 Perspective 19

Ov.3 Cross-Cutting Issues 20

Ov.3.1 Cost/Performance Analysis 20

Ov.3.2 Power and Energy 26

Ov.3.3 Reliability 32

Ov.3.4 Virtual Memory 34

Ov.4 An Example Holistic Analysis 41

Ov.4.1 Fully-Buffered DIMM vs. the Disk Cache 41

Ov.4.2 Fully Buffered DIMM: Basics 43

Ov.4.3 Disk Caches: Basics 46

Ov.4.4 Experimental Results 47

Ov.4.5 Conclusions 52

Ov.5 What to Expect 54

Part I Cache 55

Chapter 1 An Overview of Cache Principles 57

1.1 Caches, 'Caches,' and "Caches" 59

1.2 Locality Principles 62

1.2.1 Temporal Locality 63

1.2.2 Spatial Locality 63

1.2.3 Algorithmic Locality 64

1.2.4 Geographical Locality? Demographical Locality? 65

1.3 What to Cache, Where to Put It, and How to Maintain It 66

1.3.1 Logical Organization Basics: Blocks, Tags, Sets 67

1.3.2 Content Management: To Cache or Not to Cache 68

1.3.3 Consistency Management: Its Responsibilities 69

1.3.4 Inclusion and Exclusion 70

1.4 Insights and Optimizations 73

1.4.1 Perspective 73

1.4.2 Important Issues, Future Directions 77

Chapter 2 Logical Organization 79

2.1 Logical Organization: A Taxonomy 79

2.2 Transparently Addressed Caches 82

2.2.1 Implicit Management: Transparent Caches 86

2.2.2 Explicit Management: Software-Managed Caches 86

2.3 Non-Transparently Addressed Caches 90

2.3.1 Explicit Management: Scratch-Pad Memories 91

2.3.2 Implicit Management: Self-Managed Scratch-Pads 92

2.4 Virtual Addressing and Protection 92

2.4.1 Virtual Caches 93

2.4.2 ASIDs and Protection Bits 96

2.4.3 Inherent Problems 96

2.5 Distributed and Partitioned Caches 97

2.5.1 UMA and NUMA 98

2.5.2 COMA 99

2.5.3 NUCA and NuRAPID 99

2.5.4 Web Caches 100

2.5.5 Buffer Caches 102

2.6 Case Studies 103
2.6.1 A Horizontal-Exclusive Organization: Victim Caches, Assist Caches 103
2.6.2 A Software Implementation: BSD's Buffer Cache 104
2.6.3 Another Dynamic Cache Block: Trace Caches 106

Chapter 3 Management of Cache Contents 117
3.1 Case Studies: On-Line Heuristics 120
3.1.1 On-Line Partitioning Heuristics 120
3.1.2 On-Line Prefetching Heuristics 129
3.1.3 On-Line Locality Optimizations 141
3.2 Case Studies: Off-Line Heuristics 151
3.2.1 Off-Line Partitioning Heuristics 151
3.2.2 Off-Line Prefetching Heuristics 155
3.2.3 Off-Line Locality Optimizations 161
3.3 Case Studies: Combined Approaches 169
3.3.1 Combined Approaches to Partitioning 170
3.3.2 Combined Approaches to Prefetching 174
3.3.3 Combined Approaches to Optimizing Locality 180
3.4 Discussions 202
3.4.1 Proposed Scheme vs. Baseline 202
3.4.2 Prefetching vs. Locality Optimizations 203
3.4.3 Application-Directed Management vs. Transparent Management 203
3.4.4 Real Time vs. Average Case 204
3.4.5 Naming vs. Cache Conflicts 205
3.4.6 Dynamic vs. Static Management 208
3.5 Building a Content-Management Solution 212
3.5.1 Degree of Dynamism 212
3.5.2 Degree of Prediction 213
3.5.3 Method of Classification 213
3.5.4 Method for Ensuring Availability 214

Chapter 4 Management of Cache Consistency 217
4.1 Consistency with Backing Store 218
4.1.1 Write-Through 218
4.1.2 Delayed Write, Driven By the Cache 219
4.1.3 Delayed Write, Driven by Backing Store 220

4.2 Consistency with Self 220
4.2.1 Virtual Cache Management 220
4.2.2 ASID Management 225
4.3 Consistency with Other Clients 226
4.3.1 Motivation, Explanation, Intuition 226
4.3.2 Coherence vs. Consistency 231
4.3.3 Memory-Consistency Models 233
4.3.4 Hardware Cache-Coherence Mechanisms 240
4.3.5 Software Cache-Coherence Mechanisms 254

Chapter 5 Implementation Issues 257
5.1 Overview 257
5.2 SRAM Implementation 258
5.2.1 Basic 1-Bit Memory Cell 259
5.2.2 Address Decoding 262
5.2.3 Physical Decoder Implementation 268
5.2.4 Peripheral Bitline Circuits 278
5.2.5 Sense Amplifiers 281
5.2.6 Write Amplifier 283
5.2.7 SRAM Partitioning 285
5.2.8 SRAM Control and Timing 286
5.2.9 SRAM Interface 292
5.3 Advanced SRAM Topics 293
5.3.1 Low-Leakage Operation 293
5.4 Cache Implementation 297
5.4.1 Simple Caches 297
5.4.2 Processor Interfacing 298
5.4.3 Multiporting 298

Chapter 6 Cache Case Studies 301
6.1 Logical Organization 301
6.1.1 Motorola MPC7450 301
6.1.2 AMD Opteron 301
6.1.3 Intel Itanium-2 303

6.2 Pipeline Interface 304
6.2.1 Motorola MPC7450 304
6.2.2 AMD Opteron 304
6.2.3 Intel Itanium-2 304
6.3 Case Studies of Detailed Itanium-2 Circuits 305
6.3.1 L1 Cache RAM Cell Array 305
6.3.2 L2 Array Bitline Structure 305
6.3.3 L3 Subarray Implementation 308
6.3.4 Itanium-2 TLB and CAM Implementation 308

Part II DRAM 313

Chapter 7 Overview of DRAMs 315

7.1 DRAM Basics: Internals, Operation 316
7.2 Evolution of the DRAM Architecture 322
7.2.1 Structural Modifications Targeting Throughput 322
7.2.2 Interface Modifications Targeting Throughput 328
7.2.3 Structural Modifications Targeting Latency 330
7.2.4 Rough Comparison of Recent DRAMs 331
7.3 Modern-Day DRAM Standards 332
7.3.1 Salient Features of JEDEC's SDRAM Technology 332
7.3.2 Other Technologies, Rambus in Particular 335
7.3.3 Comparison of Technologies in Rambus and JEDEC DRAM 341
7.3.4 Alternative Technologies 343
7.4 Fully Buffered DIMM: A Compromise of Sorts 348
7.5 Issues in DRAM Systems, Briefly 350
7.5.1 Architecture and Scaling 350
7.5.2 Topology and Timing 350
7.5.3 Pin and Protocol Efficiency 351
7.5.4 Power and Heat Dissipation 351
7.5.5 Future Directions 351

Chapter 8 DRAM Device Organization: Basic Circuits and Architecture 353
8.1 DRAM Device Organization 353
8.2 DRAM Storage Cells 355
8.2.1 Cell Capacitance, Leakage, and Refresh 356
8.2.2 Conflicting Requirements Drive Cell Structure 356
8.2.3 Trench Capacitor Structure 357
8.2.4 Stacked Capacitor Structure 357
8.3 RAM Array Structures 358
8.3.1 Open Bitline Array Structure 359
8.3.2 Folded Bitline Array Structure 360
8.4 Differential Sense Amplifier 360
8.4.1 Functionality of Sense Amplifiers in DRAM Devices 361
8.4.2 Circuit Diagram of a Basic Sense Amplifier 362
8.4.3 Basic Sense Amplifier Operation 362
8.4.4 Voltage Waveform of Basic Sense Amplifier Operation 363
8.4.5 Writing into DRAM Array 365
8.5 Decoders and Redundancy 366
8.5.1 Row Decoder Replacement Example 368
8.6 DRAM Device Control Logic 368
8.6.1 Synchronous vs. Non-Synchronous 369
8.6.2 Mode Register-Based Programmability 370
8.7 DRAM Device Configuration 370
8.7.1 Device Configuration Trade-offs 371
8.8 Data I/O 372
8.8.1 Burst Lengths and Burst Ordering 372
8.8.2 N-Bit Prefetch 372
8.9 DRAM Device Packaging 373
8.10 DRAM Process Technology and Process Scaling Considerations 374
8.10.1 Cost Considerations 375
8.10.2 DRAM- vs. Logic-Optimized Process Technology 375

Chapter 9 DRAM System Signaling and Timing 377
9.1 Signaling System 377

9.2 Transmission Lines on PCBs 379
9.2.1 Brief Tutorial on the Telegrapher's Equations 380
9.2.2 RC and LC Transmission Line Models 382
9.2.3 LC Transmission Line Model for PCB Traces 383
9.2.4 Signal Velocity on the LC Transmission Line 383
9.2.5 Skin Effect of Conductors 384
9.2.6 Dielectric Loss 384
9.2.7 Electromagnetic Interference and Crosstalk 386
9.2.8 Near-End and Far-End Crosstalk 387
9.2.9 Transmission Line Discontinuities 388
9.2.10 Multi-Drop Bus 390
9.2.11 Socket Interfaces 391
9.2.12 Skew 392
9.2.13 Jitter 392
9.2.14 Inter-Symbol Interference (ISI) 393
9.3 Termination 393
9.3.1 Series Stub (Serial) Termination 394
9.3.2 On-Die (Parallel) Termination 394
9.4 Signaling 395
9.4.1 Eye Diagrams 396
9.4.2 Low-Voltage TTL (Transistor-Transistor Logic) 396
9.4.3 Voltage References 398
9.4.4 Series Stub Termination Logic 398
9.4.5 RSL and DRSL 399
9.5 Timing Synchronization 399
9.5.1 Clock Forwarding 400
9.5.2 Phase-Locked Loop (PLL) 401
9.5.3 Delay-Locked Loop (DLL) 402
9.6 Selected DRAM Signaling and Timing Issues 402
9.6.1 Data Read and Write Timing in DDRx SDRAM Memory Systems 404
9.6.2 The Use of DLL in DDRx SDRAM Devices 406
9.6.3 The Use of PLL in XDR DRAM Devices 406
9.7 Summary 408

Chapter 10 DRAM Memory System Organization 409

10.1 Conventional Memory System 409

10.2 Basic Nomenclature 409

10.2.1 Channel 410

10.2.2 Rank 413

10.2.3 Bank 414

10.2.4 Row 415

10.2.5 Column 415

10.2.6 Memory System Organization: An Example 416

10.3 Memory Modules 417

10.3.1 Single In-line Memory Module (SIMM) 418

10.3.2 Dual In-line Memory Module (DIMM) 418

10.3.3 Registered Memory Module (RDIMM) 418

10.3.4 Small Outline DIMM (SO-DIMM) 419

10.3.5 Memory Module Organization 420

10.3.6 Serial Presence Detect (SPD) 421

10.4 Memory System Topology 422

10.4.1 Direct RDRAM System Topology 422

10.5 Summary 423

Chapter 11 Basic DRAM Memory-Access Protocol 425

11.1 Basic DRAM Commands 425

11.1.1 Generic DRAM Command Format 427

11.1.2 Summary of Timing Parameters 427

11.1.3 Row Access Command 428

11.1.4 Column-Read Command 429

11.1.5 Column-Write Command 430

11.1.6 Precharge Command 431

11.1.7 Refresh Command 431

11.1.8 A Read Cycle 433

11.1.9 A Write Cycle 434

11.1.10 Compound Commands 434

11.2 DRAM Command Interactions 436

11.2.1 Consecutive Reads and Writes to Same Rank 437

11.2.2 Read to Precharge Timing 438

11.2.3 Consecutive Reads to Different Rows of Same Bank 438

11.2.4 Consecutive Reads to Different Banks: Bank Conflict 440
11.2.5 Consecutive Read Requests to Different Ranks 441
11.2.6 Consecutive Write Requests: Open Banks 442
11.2.7 Consecutive Write Requests: Bank Conflicts 444
11.2.8 Write Request Following Read Request: Open Banks 444
11.2.9 Write Request Following Read Request to Different Banks, Bank Conflict, Best Case, No Reordering 445
11.2.10 Read Following Write to Same Rank, Open Banks 446
11.2.11 Write to Precharge Timing 447
11.2.12 Read Following Write to Different Ranks, Open Banks 447
11.2.13 Read Following Write to Same Bank, Bank Conflict 448
11.2.14 Read Following Write to Different Banks of Same Rank, Bank Conflict, Best Case, No Reordering 449
11.2.15 Column-Read-and-Precharge Command Timing 449
11.2.16 Column-Write-and-Precharge Timing 450
11.3 Additional Constraints **450**
11.3.1 Device Power Limit 450
11.3.2 t_{RRD}: Row-to-Row (Activation) Delay 452
11.3.3 t_{FAW}: Four-Bank Activation Window 453
11.3.4 2T Command Timing in Unbuffered Memory Systems 454
11.4 Command Timing Summary **454**
11.5 Summary **454**

Chapter 12 Evolutionary Developments of DRAM Device Architecture **457**
12.1 DRAM Device Families **457**
12.1.1 Cost (Capacity), Latency, Bandwidth, and Power 457
12.2 Historical-Commodity DRAM Devices **458**
12.2.1 The Intel 1103 459
12.2.2 Asynchronous DRAM Devices 461
12.2.3 Page Mode and Fast Page Mode DRAM (FPM DRAM) 461
12.2.4 Extended Data-Out (EDO) and Burst Extended Data-Out (BEDO) Devices 463
12.3 Modern-Commodity DRAM Devices **464**
12.3.1 Synchronous DRAM (SDRAM) 465

12.3.2 Double Data Rate SDRAM (DDR SDRAM) ... 471
12.3.3 DDR2 SDRAM ... 474
12.3.4 Protocol and Architectural Differences ... 475
12.3.5 DDR3 SDRAM ... 476
12.3.6 Scaling Trends of Modern-Commodity DRAM Devices ... 477
12.4 High Bandwidth Path ... 480
12.4.1 Direct RDRAM ... 480
12.4.2 Technical and Pseudo-Technical Issues of Direct RDRAM ... 487
12.4.3 XDR Memory System ... 491
12.5 Low Latency ... 494
12.5.1 Reduced Latency DRAM (RLDRAM) ... 494
12.5.2 Fast Cycle DRAM (FCRAM) ... 495
12.6 Interesting Alternatives ... 495
12.6.1 Virtual Channel Memory (VCDRAM) ... 495
12.6.2 Enhanced SDRAM (ESDRAM) ... 496

Chapter 13 DRAM Memory Controller ... 497
13.1 DRAM Controller Architecture ... 497
13.2 Row-Buffer-Management Policy ... 499
13.2.1 Open-Page Row-Buffer-Management Policy ... 499
13.2.2 Close-Page Row-Buffer-Management Policy ... 499
13.2.3 Hybrid (Dynamic) Row-Buffer-Management Policies ... 500
13.2.4 Performance Impact of Row-Buffer-Management Policies ... 500
13.2.5 Power Impact of Row-Buffer-Management Policies ... 501
13.3 Address Mapping (Translation) ... 502
13.3.1 Available Parallelism in Memory System Organization ... 503
13.3.2 Parameter of Address Mapping Schemes ... 504
13.3.3 Baseline Address Mapping Schemes ... 505
13.3.4 Parallelism vs. Expansion Capability ... 506
13.3.5 Address Mapping in the Intel 82955X MCH ... 506
13.3.6 Bank Address Aliasing (Stride Collision) ... 510
13.4 Performance Optimization ... 511
13.4.1 Write Caching ... 512
13.4.2 Request Queue Organizations ... 513
13.4.3 Refresh Management ... 514

13.4.4 *Agent-Centric Request Queuing Organization* 516
13.4.5 *Feedback-Directed Scheduling* 518
13.5 Summary **518**

Chapter 14 The Fully Buffered DIMM Memory System **519**
14.1 Introduction **519**
14.2 Architecture **521**
14.3 Signaling and Timing **524**
14.3.1 *Clock Data Recovery* 524
14.3.2 *Unit Interval* 525
14.3.3 *Resample and Resync* 525
14.4 Access Protocol **526**
14.4.1 *Frame Definitions* 527
14.4.2 *Command Definitions* 528
14.4.3 *Frame and Command Scheduling* 528
14.5 The Advanced Memory Buffer **530**
14.5.1 *SMBus Interface* 531
14.5.2 *Built-In Self-Test (BIST)* 532
14.5.3 *Thermal Sensor* 532
14.6 Reliability, Availability, and Serviceability **532**
14.6.1 *Checksum Protection in the Transport Layer* 532
14.6.2 *Bit Lane Steering* 533
14.6.3 *Fail-over Modes* 534
14.6.4 *Hot Add and Replace* 534
14.7 FBDIMM Performance Characteristics **535**
14.7.1 *Fixed vs. Variable Latency Scheduling* 538
14.8 Perspective **540**

Chapter 15 Memory System Design Analysis **541**
15.1 Overview **541**
15.2 Workload Characteristics **543**
15.2.1 *164.gzip: C Compression* 544
15.2.2 *176.gcc: C Programming Language Compiler* 545
15.2.3 *197.parser: C Word Processing* 545
15.2.4 *255.vortex: C Object-Oriented Database* 546

15.2.5 172.mgrid: Fortran 77 Multi-Grid Solver: 3D Potential Field 547
15.2.6 SETI@HOME 547
15.2.7 Quake 3 548
15.2.8 178.galgel, 179.art, 183.equake, 188.ammp, JMark 2.0, and 3DWinbench 548
15.2.9 Summary of Workload Characteristics 550
15.3 The RAD Analytical Framework 551
15.3.1 DRAM-Access Protocol 551
15.3.2 Computing DRAM Protocol Overheads 551
15.3.3 Computing Row Cycle Time Constraints 553
15.3.4 Computing Row-to-Row Activation Constraints 555
15.3.5 Request Access Distance Efficiency Computation 557
15.3.6 An Applied Example for a Close-Page System 558
15.3.7 An Applied Example for an Open-Page System 558
15.3.8 System Configuration for RAD-Based Analysis 559
15.3.9 Open-Page Systems: 164.gzip 561
15.3.10 Open-Page Systems: 255.vortex 562
15.3.11 Open-Page Systems: Average of All Workloads 562
15.3.12 Close-Page Systems: 164.gzip 565
15.3.13 Close-Page Systems: SETI@HOME Processor Bus Trace 566
15.3.14 Close-Page Systems: Average of All Workloads 567
15.3.15 t_{FAW} Limitations in Open-Page System: All Workloads 568
15.3.16 Bandwidth Improvements: 8-Banks vs. 16-Banks 568
15.4 Simulation-Based Analysis 570
15.4.1 System Configurations 570
15.4.2 Memory Controller Structure 571
15.4.3 DRAM Command Scheduling Algorithms 572
15.4.4 Workload Characteristics 575
15.4.5 Timing Parameters 575
15.4.6 Protocol Table 575
15.4.7 Queue Depth, Scheduling Algorithms, and Burst Length 577
15.4.8 Effect of Burst Length on Sustainable Bandwidth 578
15.4.9 Burst Chop in DDR3 SDRAM Devices 579
15.4.10 Revisiting the 8-Bank and 16-Bank Issue with DRAMSim 584
15.4.11 8 Bank vs. 16 Banks — Relaxed t_{FAW} and t_{WTR} 587
15.4.12 Effect of Transaction Ordering on Latency Distribution 587

15.5 A Latency-Oriented Study 590
15.5.1 Experimental Framework 590
15.5.2 Simulation Input 592
15.5.3 Limit Study: Latency Bandwidth Characteristics 592
15.5.4 Latency 593
15.6 Concluding Remarks 596

Part III Disk 599

Chapter 16 Overview of Disks 601

16.1 History of Disk Drives 601
16.1.1 Evolution of Drives 603
16.1.2 Areal Density Growth Trend 605
16.2 Principles of Hard Disk Drives 606
16.2.1 Principles of Rotating Storage Devices 607
16.2.2 Magnetic Rotating Storage Device—Hard Disk Drive 608
16.3 Classifications of Disk Drives 609
16.3.1 Form Factor 609
16.3.2 Application 609
16.3.3 Interface 609
16.4 Disk Performance Overview 610
16.4.1 Disk Performance Metrics 610
16.4.2 Workload Factors Affecting Performance 612
16.4.3 Video Application Performance 612
16.5 Future Directions in Disks 612

Chapter 17 The Physical Layer 615

17.1 Magnetic Recording 615
17.1.1 Ferromagnetism 615
17.1.2 Magnetic Fields 616
17.1.3 Hysteresis Loop 618
17.1.4 Writing 618
17.1.5 Reading 620

17.2 Mechanical and Magnetic Components 620
17.2.1 Disks 621
17.2.2 Spindle Motor 623
17.2.3 Heads 625
17.2.4 Slider and Head-Gimbal Assembly 631
17.2.5 Head-Stack Assembly and Actuator 633
17.2.6 Multiple Platters 635
17.2.7 Start/Stop 636
17.2.8 Magnetic Disk Recording Integration 637
17.2.9 Head-Disk Assembly 639
17.3 Electronics 640
17.3.1 Controller 640
17.3.2 Memory 642
17.3.3 Recording Channel 642
17.3.4 Motor Controls 646

Chapter 18 The Data Layer 649
18.1 Disk Blocks and Sectors 649
18.1.1 Fixed-Size Blocks 649
18.1.2 Variable Size Blocks 650
18.1.3 Sectors 650
18.2 Tracks and Cylinders 652
18.3 Address Mapping 654
18.3.1 Internal Addressing 654
18.3.2 External Addressing 654
18.3.3 Logical Address to Physical Location Mapping 655
18.4 Zoned-Bit Recording 658
18.4.1 Handling ZBR 661
18.5 Servo 662
18.5.1 Dedicated Servo 662
18.5.2 Embedded Servo 663
18.5.3 Servo ID and Seek 666
18.5.4 Servo Burst and Track Following 667
18.5.5 Anatomy of a Servo 669
18.5.6 ZBR and Embedded Servo 669

18.6 Sector ID and No-ID Formatting 670
18.7 Capacity 672
18.8 Data Rate 673
18.9 Defect Management 673
18.9.1 Relocation Schemes 674
18.9.2 Types of Defects 675
18.9.3 Error Recovery Procedure 676

Chapter 19 Performance Issues and Design Trade-Offs 677
19.1 Anatomy of an I/O 677
19.1.1 Adding It All Up 679
19.2 Some Basic Principles 681
19.2.1 Effect of User Track Capacity 681
19.2.2 Effect of Cylinder Capacity 682
19.2.3 Effect of Track Density 684
19.2.4 Effect of Number of Heads 686
19.3 BPI vs. TPI 688
19.4 Effect of Drive Capacity 689
19.4.1 Space Usage Efficiency 690
19.4.2 Performance Implication 690
19.5 Concentric Tracks vs. Spiral Track 692
19.5.1 Optical Disks 693
19.6 Average Seek 694
19.6.1 Disks Without ZBR 694
19.6.2 Disks With ZBR 695

Chapter 20 Drive Interface 699
20.1 Overview of Interfaces 699
20.1.1 Components of an Interface 701
20.1.2 Desirable Characteristics of Interface 701
20.2 ATA 702
20.3 Serial ATA 703
20.4 SCSI 705
20.5 Serial SCSI 706
20.6 Fibre Channel 707
20.7 Cost, Performance, and Reliability 709

Chapter 21 Operational Performance Improvement 711
21.1 Latency Reduction Techniques 711
21.1.1 Dual Actuator 711
21.1.2 Multiple Copies 712
21.1.3 Zero Latency Access 713
21.2 Command Queueing and Scheduling 715
21.2.1 Seek Time-Based Scheduling 716
21.2.2 Total Access Time Based Scheduling 717
21.2.3 Sequential Access Scheduling 723
21.3 Reorganizing Data on the Disk 723
21.3.1 Defragmentation 724
21.3.2 Frequently Accessed Files 724
21.3.3 Co-Locating Access Clusters 725
21.3.4 ALIS 726
21.4 Handling Writes 728
21.4.1 Log-Structured Write 728
21.4.2 Disk Buffering of Writes 729
21.5 Data Compression 729

Chapter 22 The Cache Layer 731
22.1 Disk Cache 731
22.1.1 Why Disk Cache Works 731
22.1.2 Cache Automation 732
22.1.3 Read Cache, Write Cache 732
22.2 Cache Organizations 735
22.2.1 Desirable Features of Cache Organization 735
22.2.2 Fixed Segmentation 736
22.2.3 Circular Buffer 737
22.2.4 Virtual Memory Organization 738
22.3 Caching Algorithms 741
22.3.1 Perspective of Prefetch 741
22.3.2 Lookahead Prefetch 742
22.3.3 Look-behind Prefetch 742
22.3.4 Zero Latency Prefetch 743

22.3.5 ERP During Prefetch 743
22.3.6 Handling of Sequential Access 743
22.3.7 Replacement Policies 745

Chapter 23 Performance Testing 747
23.1 Test and Measurement 747
23.1.1 Test Initiator 747
23.1.2 Monitoring and Measuring 749
23.1.3 The Test Drive 750
23.2 Basic Tests 750
23.2.1 Media Data Rate 751
23.2.2 Disk Buffer Data Rate 751
23.2.3 Sequential Performance 752
23.2.4 Random Performance 752
23.2.5 Command Reordering Performance 755
23.3 Benchmark Tests 755
23.3.1 Guidelines for Benchmarking 756
23.4 Drive Parameters Tests 757
23.4.1 Geometry and More 757
23.4.2 Seek Time 759

Chapter 24 Storage Subsystems 763
24.1 Data Striping 763
24.2 Data Mirroring 765
24.2.1 Basic Mirroring 766
24.2.2 Chained Decluster Mirroring 766
24.2.3 Interleaved Decluster Mirroring 767
24.2.4 Mirroring Performance Comparison 767
24.2.5 Mirroring Reliability Comparison 769
24.3 RAID 770
24.3.1 RAID Levels 770
24.3.2 RAID Performance 774
24.3.3 RAID Reliability 775
24.3.4 Sparing 776

24.3.5 RAID Controller 778
24.3.6 Advanced RAIDs 779
24.4 SAN 780
24.5 NAS 782
24.6 ISCSI 783

Chapter 25 Advanced Topics 785
25.1 Perpendicular Recording 785
25.1.1 Write Process, Write Head, and Media 786
25.1.2 Read Process and Read Head 788
25.2 Patterned Media 788
25.2.1 Fully Patterned Media 789
25.2.2 Discrete Track Media 790
25.3 Thermally Assisted Recording 790
25.4 Dual Stage Actuator 792
25.4.1 Microactuators 793
25.5 Adaptive Formatting 794
25.6 Hybrid Disk Drive 796
25.6.1 Benefits 796
25.6.2 Architecture 796
25.6.3 Proposed Interface 797
25.7 Object-Based Storage 798
25.7.1 Object Storage Main Concept 798
25.7.2 Object Storage Benefits 800

Chapter 26 Case Study 803
26.1 The Mechanical Components 803
26.1.1 Seek Profile 804
26.2 Electronics 806
26.3 Data Layout 806
26.3.1 Data Rate 808
26.4 Interface 809
26.5 Cache 810
26.6 Performance Testing 810

26.6.1 Sequential Access *810*
26.6.2 Random Access *811*

Part IV Cross-Cutting Issues 813

Chapter 27 The Case for Holistic Design 815

27.1 Anecdotes, Revisited **816**
27.1.1 Anecdote I: Systemic Behaviors Exist *816*
27.1.2 Anecdote II: The DLL in DDR SDRAM *818*
27.1.3 Anecdote III: A Catch-22 in the Search for Bandwidth *822*
27.1.4 Anecdote IV: Proposals to Exploit Variability in Cell Leakage *824*
27.2 Perspective **827**

Chapter 28 Analysis of Cost and Performance 829

28.1 Combining Cost and Performance **829**
28.2 Pareto Optimality **830**
28.2.1 The Pareto-Optimal Set: An Equivalence Class *830*
28.2.2 Stanley's Observation *832*
28.3 Taking Sampled Averages Correctly **833**
28.3.1 Sampling Over Time *834*
28.3.2 Sampling Over Distance *835*
28.3.3 Sampling Over Fuel Consumption *836*
28.3.4 The Moral of the Story *837*
28.4 Metrics for Computer Performance **838**
28.4.1 Performance and the Use of Means *838*
28.4.2 Problems with Normalization *839*
28.4.3 The Meaning of Performance *842*
28.5 Analytical Modeling and the Miss-Rate Function **843**
28.5.1 Analytical Modeling *843*
28.5.2 The Miss-Rate Function *844*

Chapter 29 Power and Leakage 847

29.1 Sources of Leakage in CMOS Devices **847**
29.2 A Closer Look at Subthreshold Leakage **851**

29.3 vCACTI and Energy/Power Breakdown of Pipelined Nanometer Caches 855
29.3.1 Leakage in SRAM Cells 855
29.3.2 Pipelined Caches 857
29.3.3 Modeling 858
29.3.4 Dynamic and Static Power 859
29.3.5 Detailed Power Breakdown 860

Chapter 30 Memory Errors and Error Correction 865
30.1 Types and Causes of Failures 865
30.1.1 Alpha Particles 866
30.1.2 Primary Cosmic Rays and Terrestrial Neutrons 866
30.1.3 Soft Error Mechanism 867
30.1.4 Single-Bit and Multi-Bit Failures 867
30.2 Soft Error Rates and Trends 868
30.3 Error Detection and Correction 869
30.3.1 Parity 869
30.3.2 Single-Bit Error Correction (SEC ECC) 870
30.3.3 Single-Bit Error Correction, Double-Bit Error Detection (SECDED ECC) 873
30.3.4 Multi-Bit Error Detection and Correction: Bossen's b-Adjacent Algorithm 874
30.3.5 Bit Steering and Chipkill 875
30.3.6 Chipkill with ×8 DRAM Devices 877
30.3.7 Memory Scrubbing 879
30.3.8 Bullet Proofing the Memory System 880
30.4 Reliability of Non-DRAM Systems 880
30.4.1 SRAM 880
30.4.2 Flash 880
30.4.3 MRAM 880
30.5 Space Shuttle Memory System 881

Chapter 31 Virtual Memory 883
31.1 A Virtual Memory Primer 884
31.1.1 Address Spaces and the Main Memory Cache 885
31.1.2 Address Mapping and the Page Table 886
31.1.3 Hierarchical Page Tables 887

31.1.4 Inverted Page Tables *890*
31.1.5 Comparison: Inverted vs. Hierarchical *892*
31.1.6 Translation Lookaside Buffers, Revisited *893*
31.1.7 Perspective: Segmented Addressing Solves the Synonym Problem *895*
31.1.8 Perspective: A Taxonomy of Address Space Organizations *901*
31.2 Implementing Virtual Memory **906**
31.2.1 The Basic In-Order Pipe *908*
31.2.2 Precise Interrupts in Pipelined Computers *910*

References **921**

Index **955**

Preface

"It's the Memory, Stupid!"

If you develop an ear for sounds that are musical it is like developing an ego. You begin to refuse sounds that are not musical and that way cut yourself off from a good deal of experience.

—John Cage

In 1996, Richard Sites, one of the fathers of computer architecture and lead designers of the DEC Alpha, had the following to say about the future of computer architecture research:

> *Across the industry, today's chips are largely able to execute code faster than we can feed them with instructions and data. There are no longer performance bottlenecks in the floating-point multiplier or in having only a single integer unit. The real design action is in memory subsystems—caches, buses, bandwidth, and latency.*
>
> *An anecdote: in a recent database benchmark study using TPC-C, both 200-MHz Pentium Pro and 400MHz 21164 Alpha systems were measured at 4.2–4.5 CPU cycles per instruction retired. In other words, three out of every four CPU cycles retired zero instructions: most were spent waiting for memory. Processor speed has seriously outstripped memory speed.*
>
> *Increasing the width of instruction issue and increasing the number of simultaneous instruction streams only makes the memory bottleneck worse. If a CPU chip today needs to move 2 GBytes/s (say, 16 bytes every 8 ns) across the pins to keep itself busy, imagine a chip in the foreseeable future with twice the clock rate, twice the issue width, and two instruction streams. All these factors multiply together to require about 16 GBytes/s of pin bandwidth to keep this chip busy. It is not clear whether pin bandwidth can keep up—32 bytes every 2ns?*
>
> *I expect that over the coming decade memory subsystems design will be the* ***only*** *important design issue for microprocessors.* [Sites 1996, emphasis Sites']

The title of Sites' article is "It's the Memory, Stupid!" Sites realized in 1996 what we as a community are only now, more than a decade later, beginning to digest and internalize fully: *uh, guys, it really* ***is*** *the memory system ... little else matters right now, so stop wasting time and resources on other facets of the design.* Most of his colleagues designing next-generation Alpha architectures at Digital Equipment Corp. ignored his advice and instead remained focused on building ever faster microprocessors, rather than shifting their focus to the building of ever faster *systems*. It is perhaps worth noting that Digital Equipment Corp. no longer exists.

The increasing gap between processor and memory speeds has rendered the organization, architecture, and design of memory subsystems an increasingly important part of computer-systems design. Today, the divide is so severe we are now in one of those down-cycles where the processor is so good at

number-crunching it has completely sidelined itself; it is too fast for its own good, in a sense. Sites' prediction came true: memory subsystems design is now and has been for several years the *only* important design issue for microprocessors and systems. Memory-hierarchy parameters affect system performance *significantly* more than processor parameters (e.g., they are responsible for 2–10× changes in execution time, as opposed to 2–10%), making it absolutely essential for any designer of computer systems to exhibit an in-depth knowledge of the memory system's organization, its operation, its not-so-obvious behavior, and its range of performance characteristics. This is true now, and it is likely to remain true in the near future.

Thus this book, which is intended to provide exactly that type of in-depth coverage over a wide range of topics.

Topics Covered

In the following chapters we address the logical design and operation, the physical design and operation, the performance characteristics (i.e., design trade-offs), and, to a limited extent, the energy consumption of modern memory hierarchies.

In the cache section, we present topics and perspectives that will be new (or at least interesting) to even veterans in the field. What this implies is that the cache section is *not* an overview of processor-cache organization and its effect on performance—instead, we build up the concept of cache from first principles and discuss topics that are incompletely covered in the computer-engineering literature. The section discusses a significant degree of historical development in cache-management techniques, the physical design of modern SRAM structures, the operating system's role in cache coherence, and the continuum of cache architectures from those that are fully transparent (to application software and/or the operating system) to those that are fully visible.

DRAM and disk are interesting technologies because, unlike caches, they are not typically integrated onto the microprocessor die. Thus any discussion of these topics necessarily deals with the issue of communication: e.g., channels, signalling, protocols, and request scheduling.

DRAM involves one or more chip-to-chip crossings, and so signalling and signal integrity are as fundamental as circuit design to the technology. In the DRAM section, we present an intuitive understanding of exactly what happens inside the DRAM so that the ubiquitous parameters of the interface (e.g., t_{RC}, t_{RCD}, t_{CAS}, etc.) will make sense. We survey the various DRAM architectures that have appeared over the years and give an in-depth description of the technologies in the next generation memory-system architecture. We discuss memory-controller issues and investigate performance issues of modern systems.

The disk section builds from the bottom up, providing a view of the disk from physical recording principles to the configuration and operation of disks within system settings. We discuss the operation of the disk's read/write heads; the arrangement of recording media within the enclosure; and the organization-level view of blocks, sectors, tracks, and cylinders, as well as various protocols used to encode data. We discuss performance issues and techniques used to improve performance, including caching and buffering, prefetching, request scheduling, and data reorganization. We discuss the various disk interfaces available today (e.g., ATA, serial ATA, SCSI, fibre channel, etc.) as well as system configurations such as RAID, SAN, and NAS.

The last section of the book, *Cross-Cutting Issues*, covers topics that apply to all levels of the memory hierarchy, such as the tools of analysis and how to use them correctly, subthreshold leakage power in CMOS devices and circuits, a look at power breakdowns in future SRAMs, codes for error detection and error correction, the design and operation of virtual memory systems, and the hardware mechanisms that are required in microprocessors to support virtual memory.

Goals and Audience

The primary goal of this book is to bring the reader to a level of understanding at which the physical design and/or detailed software emulation of the entire hierarchy is possible, from cache to disk. As we argue in the initial chapter, this level of understanding

is important now and will become increasingly necessary over time. Another goal of the book is to discuss techniques of analysis, so that the next generation of design engineers is prepared to tackle the nontrivial multidimensional optimization problems that result from considering detailed side-effects that can manifest themselves at any point in the entire hierarchy.

Accordingly, our target audience are those planning to build and/or optimize memory systems: i.e., computer-engineering and computer-science faculty and graduate students (and perhaps advanced undergraduates) and developers in the computer design, peripheral design, and embedded systems industries.

As an educational textbook, this is targeted at graduate and undergraduate students with a solid background in computer organization and architecture. It could serve to support an advanced senior-level undergraduate course or a second-year graduate course specializing in computer-systems design. There is clearly far too much material here for any single course; the book provides depth on enough topics to support two to three separate courses. For example, at the University of Maryland we use the DRAM section to teach a graduate class called *High-Speed Memory Systems,* and we supplement both our general and advanced architecture classes with material from the sections on *Caches* and *Cross-Cutting Issues.* The *Disk* section could support a class focused solely on disks, and it is also possible to create for advanced students a survey class that lightly touches on all the topics in the book.

As a reference, this book is targeted toward both academics and professionals alike. It provides the breadth necessary to understand the wide scope of behaviors that appear in modern memory systems, and most of the topics are addressed in enough depth that a reader should be able to build (or at least model in significant detail) caches, DRAMs, disks, their controllers, their subsystems ... and understand their interactions.

What this means is that the book should not only be useful to developers, but it should also be useful to those responsible for long-range planning and forecasting for future product developments and their issues.

Acknowledgments and Thanks

Beyond the kind folks named in the book's Dedication, we have a lot of people to thank. The following people were enormously helpful in creating the contents of this book:

- Prof. Rajeev Barua, ECE Maryland, provided text to explain scratch-pad memories and their accompanying partitioning problem (see Chapter 3).
- Dr. Brinda Ganesh, a former graduate student in Bruce Jacob's research group, now at Intel DAP, wrote the latency-oriented section of the DRAM-performance chapter (see Section 15.5).
- Joseph Gross, a graduate student in Bruce Jacob's research group, updated the numbers in David Wang's Ph.D. proposal to produce the tables comparing the characteristics of modern DRAMs in the book's *Overview* chapter.
- Dr. Ed Growchowski, formerly of Hitachi Global Storage Technologies and currently a consultant for IDEMA, provided some of the disk recording density history charts.
- Dr. Martin Hassner, Dr. Bruce Wilson, Dr. Matt White, Chuck Cox (now with IBM), Frank Chu, and Tony Dunn of Hitatchi Global Storage Technologies provided important consultation on various details of disk drive design.
- Dr. Windsor Hsu, formerly of IBM Almaden Research Center and now with Data Domain, Inc., provided important consultation on system issues related to disk drives.
- Dr. Aamer Jaleel, a former graduate student in Bruce Jacob's research group, now at Intel VSSAD, is responsible for Chapter 4's sections on cache coherence.
- Michael Martin, a graduate student in Bruce Jacob's research group, executed simulations for the final table (Ov. 5) and formatted the four large time-sequence graphs in the *Overview* chapter.
- Rami Nasr, a graduate student at Maryland who wrote his M.S. thesis on the Fully

Buffered DIMM architecture, provided much of the contents of Chapter 14.

- Prof. Marty Peckerar, ECE Maryland, provided brilliant text and illustrations to explain the subthreshold leakage problem (see the book's *Overview* and Section 29.2).
- Profs. Yan Solihin, ECE NCSU, and Donald Yeung, ECE Maryland, provided much of the material on hardware and software prefetching (Sections 3.1.2 and 3.2.2); this came from a book chapter on the topic written by the pair for Kaeli and Yew's *Speculative Execution in High Performance Computer Architectures* (CRC Press, 2005).
- Sadagopan Srinivasan, a graduate student in Bruce Jacob's research group, performed the study at the end of Chapter 1 and provided much insight on the memory behavior of both streaming applications and multi-core systems.
- Dr. Nuengwong Tuaycharoen, a former graduate student in Bruce Jacob's research group, now at Thailand's Dhurakijpundit University, performed the experiments, and wrote the example holistic analysis at the end of the book's *Overview* chapter, directly relating to the behavior of caches, DRAMs, and disks in a single study.
- Patricia Wadkins and others in the Rochester SITLab of Hitachi Global Storage Technologies provided test results and measurement data for the book's Section III on *Disks*.
- Mr. Yong Wang, a signal integrity expert, formerly of HP and Intel, now with Meta-RAM, contributed extensively to the Chapter on *DRAM System Signaling and Timing* (Chapter 9).
- Michael Xu at Hitachi Global Storage Technologies drew the beautiful, complex illustrations, as well as provided some of the photographs, in the book's Section III on Disks.

In addition, several students involved in tool support over the years deserve special recognition:

- Brinda Ganesh took the reins from David Wang to maintain *DRAMsim*; among other things, she is largely responsible for the FB-DIMM support in that tool.
- Joseph Gross also supports *DRAMsim* and is leading the development of the second generation version of the software, which is object-oriented and significantly streamlined.
- Nuengwong Tuaycharoen integrated the various disparate software modules to produce *SYSim*, a full-system simulator that gave us the wonderful time-sequence graphs at the end of the *Overview*.

Numerous reviewers spent considerable time reading early drafts of the manuscript and providing excellent critiques or our direction, approach, and raw content. The following reviewers directly contributed to the book in this way: Ashraf Aboulnaga, *University of Waterloo*; Al Davis, *University of Utah*; Diana Franklin, *California Polytechnic University, San Luis Obispo*; Yiming Hu, *University of Cincinnati*; David Kaeli, *Northeastern University*; Nagi Mekhiel, *Ryerson University, Toronto*; Michael Schuette, Ph.D., VP of Technology Development at *OCZ Technology*; Jim Smith, *University of Wisconsin—Madison*; Yan Solin, *North Carolina State University*; and Several Anonymous reviewers (i.e., we the authors were told not to use their names).

The editorial and production staff at Morgan Kaufmann/Elsevier was amazing to work with: Denise Penrose pitched the idea to us in the first place (and enormous thanks go to Jim Smith, who pointed her in our direction) and Nate McFadden and Paul Gottehrer made the process of writing, editing, and proofing go incredibly smoothly.

Lastly, Dr. Richard Matick, the author of the original memory-systems book (*Computer Storage Systems and Technology*, John Wiley & Sons, 1976) and currently a leading researcher in embedded DRAM at IBM T. J. Watson, provided enormous help in direction, focus, and approach.

Dave, Spencer, Sam, and I are indebted to all of these writers, illustrators, coders, editors, and reviewers alike; they all helped to make this book what it is. To those contributors: thank you. You rock.

Bruce Jacob, Summer 2007
College Park, Maryland

On Memory Systems and Their Design

Memory is essential to the operation of a computer system, and nothing is more important to the development of the modern memory system than the concept of the memory hierarchy. While a flat memory system built of a single technology is attractive for its simplicity, a well-implemented hierarchy allows a memory system to approach simultaneously the performance of the fastest component, the cost per bit of the cheapest component, and the energy consumption of the most energy-efficient component.

For years, the use of a memory hierarchy has been very convenient, in that it has simplified the process of designing memory systems. The use of a hierarchy allowed designers to treat system design as a modularized process—to treat the memory system as an abstraction and to optimize individual subsystems (caches, DRAMs [dynamic RAM], disks) in isolation.

However, we are finding that treating the hierarchy in this way—as a set of disparate subsystems that interact only through well-defined functional interfaces and that can be optimized in isolation—no longer suffices for the design of modern memory systems. One trend becoming apparent is that many of the underlying implementation issues are becoming significant. These include the physics of device and interconnect scaling, the choice of signaling protocols and topologies to ensure signal integrity, design parameters such as granularity of access and support for concurrency, and communication-related issues such as scheduling algorithms and queueing. These low-level details have begun to affect the higher level design process quite dramatically, whereas they were considered transparent only a design-generation ago. Cache architectures are appearing that play to the limitations imposed by interconnect physics in deep submicron processes; modern DRAM design is driven by circuit-level limitations that create system-level headaches; and modern disk performance is dominated by the on-board caching and scheduling policies. This is a non-trivial environment in which to attempt optimal design.

This trend will undoubtedly become more important as time goes on, and even now it has tremendous impact on design results. As hierarchies and their components grow more complex, *systemic* behaviors—those arising from the complex interaction of the memory system's parts—have begun to dominate. The real loss of performance is not seen in the CPU or caches or DRAM devices or disk assemblies themselves, but in the subtle interactions between these subsystems and in the manner in which these subsystems are connected. Consequently, it is becoming increasingly foolhardy to attempt system-level optimization by designing/optimizing each of the parts in isolation (which, unfortunately, is often the approach taken in modern computer design). No longer can a designer remain oblivious to issues "outside the scope" and focus solely on designing a subsystem. It has now become the case that a memory-systems designer, wishing to build a properly behaved memory hierarchy, must be intimately familiar with issues involved at all levels of an implementation, from cache to DRAM to disk. Thus, we wrote this book.

Ov.1 Memory Systems

A memory hierarchy is designed to provide multiple functions that are seemingly mutually exclusive. We start at random-access memory (RAM): all microprocessors (and computer systems in general) expect a random-access memory out of which they operate. This is fundamental to the structure of modern software, built upon the von Neumann model in which code and data are essentially the same and reside in the same place (i.e., memory). All requests, whether for instructions or for data, go to this random-access memory. At any given moment, any particular datum in memory may be needed; there is no requirement that data reside next to the code that manipulates it, and there is no requirement that two instructions executed one after the other need to be adjacent in memory. Thus, the memory system must be able to handle randomly addressed[1] requests in a manner that favors no particular request. For instance, using a tape drive for this primary memory is unacceptable for performance reasons, though it might be acceptable in the Turing-machine sense.

Where does the mutually exclusive part come in? As we said, all microprocessors are built to expect a random-access memory out of which they can operate. Moreover, this memory must be *fast*, matching the machine's processing speed; otherwise, the machine will spend most of its time tapping its foot and staring at its watch. In addition, modern software is written to expect gigabytes of storage for data, and the modern consumer expects this storage to be cheap. How many memory technologies provide both tremendous speed and tremendous storage capacity at a low price? Modern processors execute instructions both out of order and speculatively—put simply, they execute instructions that, in some cases, are not meant to get executed—and system software is typically built to expect that certain changes to memory are permanent. How many memory technologies provide non-volatility and an *undo* operation?

While it might be elegant to provide all of these competing demands with a single technology (say, for example, a gigantic battery-backed SRAM [static RAM]), and though there is no engineering problem that cannot be solved (if ever in doubt about this, simply query a room full of engineers), the reality is that building a full memory system out of such a technology would be prohibitively expensive today.[2] The good news is that it is not necessary. Specialization and division of labor make possible all of these competing goals simultaneously. Modern memory systems often have a terabyte of storage on the desktop and provide instruction-fetch and data-access bandwidths of 128 GB/s or more. Nearly all of the storage in the system is non-volatile, and speculative execution on the part of the microprocessor is supported. All of this can be found in a memory system that has an average cost of roughly 1/100,000,000 pennies per bit of storage.

The reason all of this is possible is because of a phenomenon called *locality of reference* [Belady 1966, Denning 1970]. This is an observed behavior that computer applications tend to exhibit and that, when exploited properly, allows a small memory to serve in place of a larger one.

Ov.1.1 Locality of Reference Breeds the Memory Hierarchy

We think linearly (in steps), and so we program the computer to solve problems by working in steps. The practical implications of this are that a computer's use of the memory system tends to be non-random and highly predictable. Thus is born the concept of *locality of reference*, so named because memory references tend to be localized in time and space:

- If you use something once, you are likely to use it again.
- If you use something once, you are likely to use its neighbor.

The first of these principles is called *temporal locality*; the second is called *spatial locality*. We will discuss them (and another type of locality) in more detail in *Part I: Cache* of this book, but for now it suffices to

[1] Though "random" addressing is the commonly used term, authors actually mean *arbitrarily* addressed requests because, in most memory systems, a *randomly* addressed sequence is one of the most efficiently handled events.
[2] Even Cray machines, which were famous for using SRAM as their main memory, today are built upon DRAM for their main memory.

say that one can exploit the locality principle and render a single-level memory system, which we just said was expensive, unnecessary. If a computer's use of the memory system, given a small time window, is both predictable and limited in spatial extent, then it stands to reason that a program does not need all of its data immediately accessible. A program would perform nearly as well if it had, for instance, a *two-level* store, in which the first level provides immediate access to a subset of the program's data, the second level holds the remainder of the data but is slower and therefore cheaper, and some appropriate heuristic is used to manage the movement of data back and forth between the levels, thereby ensuring that the most-needed data is usually in the first-level store.

This generalizes to the *memory hierarchy*: multiple levels of storage, each optimized for its assigned task. By choosing these levels wisely a designer can produce a system that has the best of all worlds: performance approaching that of the fastest component, cost per bit approaching that of the cheapest component, and energy consumption per access approaching that of the least power-hungry component.

The modern hierarchy is comprised of the following components, each performing a particular function or filling a functional niche within the system:

- **Cache (SRAM):** Cache provides access to program instructions and data that has very low latency (e.g., 1/4 nanosecond per access) and very high bandwidth (e.g., a 16-byte instruction block and a 16-byte data block per cycle => 32 bytes per 1/4 nanosecond, or 128 bytes per nanosecond, or 128 GB/s). It is also important to note that cache, on a per-access basis, also has relatively low energy requirements compared to other technologies.
- **DRAM:** DRAM provides a random-access storage that is relatively large, relatively fast, and relatively cheap. It is large and cheap compared to cache, and it is fast compared to disk. Its main strength is that it is just fast enough and just cheap enough to act as an operating store.
- **Disk:** Disk provides permanent storage at an ultra-low cost per bit. As mentioned, nearly all computer systems expect *some* data to be modifiable yet permanent, so the memory system must have, at some level, a permanent store. Disk's advantage is its very reasonable cost (currently less than 50¢ per gigabyte), which is low enough for users to buy enough of it to store thousands of songs, video clips, photos, and other memory hogs that users are wont to accumulate in their accounts (authors included).

Table Ov.1 lists some rough order-of-magnitude comparisons for access time and energy consumption per access.

Why is it not feasible to build a flat memory system out of these technologies? Cache is far too expensive to be used as permanent storage, and its cost to store a single album's worth of audio would exceed that of the

TABLE Ov.1 Cost-performance for various memory technologies

Technology	Bytes per Access (typ.)	Latency per Access	Cost per Megabyte[a]	Energy per Access
On-chip Cache	10	100 of picoseconds	$1–100	1 nJ
Off-chip Cache	100	Nanoseconds	$1–10	10–100 nJ
DRAM	1000 (internally fetched)	10–100 nanoseconds	$0.1	1–100 nJ (per device)
Disk	1000	Milliseconds	$0.001	100–1000 mJ

[a]Cost of semiconductor memory is extremely variable, dependent much more on economic factors and sales volume than on manufacturing issues. In particular, on-chip caches (i.e., those integrated with a microprocessor core) can take up half of the die area, in which case their "cost" would be half of the selling price of that microprocessor. Depending on the market (e.g., embedded versus high end) and sales volume, microprocessor costs cover an enormous range of prices, from pennies per square millimeter to several dollars per square millimeter.

original music CD by several orders of magnitude. Disk is far too slow to be used as an operating store, and its average seek time for random accesses is measured in milliseconds. Of the three, DRAM is the closest to providing a flat memory system. DRAM is sufficiently fast enough that, without the support of a cache front-end, it can act as an operating store for many embedded systems, and with battery back-up it can be made to function as a permanent store. However, DRAM alone is not cheap enough to serve the needs of human users, who often want nearly a terabyte of permanent storage, and, even with random access times in the tens of nanoseconds, DRAM is not quite fast enough to serve as the only memory for modern general-purpose microprocessors, which would prefer a new block of instructions every fraction of a nanosecond.

So far, no technology has appeared that provides every desired characteristic: low cost, non-volatility, high bandwidth, low latency, etc. So instead we build a system in which each component is designed to offer one or more characteristics, and we manage the operation of the system so that the poorer characteristics of the various technologies are "hidden." For example, if most of the memory references made by the microprocessor are handled by the cache and/or DRAM subsystems, then the disk will be used only rarely, and, therefore, its extremely long latency will contribute very little to the average access time. If most of the data resides in the disk subsystem, and very little of it is needed at any given moment in time, then the cache and DRAM subsystems will not need much storage, and, therefore, their higher costs per bit will contribute very little to the average cost of the system. If done right, a memory system has an average cost approaching that of bottom-most layer and an average access time and bandwidth approaching that of topmost layer.

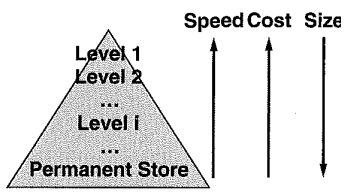


FIGURE Ov.1: A memory hierachy.

The memory hierarchy is usually pictured as a pyramid, as shown in Figure Ov.1. The higher levels in the hierarchy have better performance characteristics than the lower levels in the hierarchy; the higher levels have a higher cost per bit than the lower levels; and the system uses fewer bits of storage in the higher levels than found in the lower levels.

Though modern memory systems are comprised of SRAM, DRAM, and disk, these are simply technologies chosen to serve particular needs of the system, namely permanent store, operating store, and a fast store. Any technology set would suffice if it (a) provides permanent and operating stores and (b) satisfies the given computer system's performance, cost, and power requirements.

Permanent Store

The system's permanent store is where everything lives ... meaning it is home to data that can be modified (potentially), but whose modifications must be remembered across invocations of the system (power-ups and power-downs). In general-purpose systems, this data typically includes the operating system's files, such as boot program, OS (operating system) executable, libraries, utilities, applications, etc., and the users' files, such as graphics, word-processing documents, spreadsheets, digital photographs, digital audio and video, email, etc. In embedded systems, this data typically includes the system's executable image and any installation-specific configuration information that it requires. Some embedded systems also maintain in permanent store the state of any partially completed transactions to withstand worst-case scenarios such as the system going down before the transaction is finished (e.g., financial transactions).

These all represent data that should not disappear when the machine shuts down, such as a user's saved email messages, the operating system's code and configuration information, and applications and their saved documents. Thus, the storage must be *non-volatile*, which in this context means not susceptible to power outages. Storage technologies chosen for permanent store include magnetic disk, flash memory, and even EEPROM (electrically erasable programmable read-only memory), of which flash memory is a special type. Other forms of programmable ROM (read-only memory) such as ROM, PROM (programmable ROM),

or EPROM (erasable programmable ROM) are suitable for non-writable permanent information such as the executable image of an embedded system or a general-purpose system's boot code and BIOS.[3] Numerous exotic non-volatile technologies are in development, including magnetic RAM (MRAM), FeRAM (ferroelectric RAM), and phase-change RAM (PCRAM).

In most systems, the cost per bit of this technology is a very important consideration. In general-purpose systems, this is the case because these systems tend to have an enormous amount of permanent storage. A desktop can easily have more than 500 GB of permanent store, and a departmental server can have one hundred times that amount. The enormous number of bits in these systems translates even modest cost-per-bit increases into significant dollar amounts. In embedded systems, the cost per bit is important because of the significant number of units shipped. Embedded systems are often consumer devices that are manufactured and sold in vast quantities, e.g., cell phones, digital cameras, MP3 players, programmable thermostats, and disk drives. Each embedded system might not require more than a handful of megabytes of storage, yet a tiny 1¢ increase in the cost per megabyte of memory can translate to a $100,000 increase in cost per million units manufactured.

Operating (Random-Access) Store

As mentioned earlier, a typical microprocessor expects a new instruction or set of instructions on every clock cycle, and it can perform a data-read or data-write every clock cycle. Because the addresses of these instructions and data need not be sequential (or, in fact, related in any detectable way), the memory system must be able to handle *random access*—it must be able to provide instant access to any datum in the memory system.

The machine's operating store is the level of memory that provides random access at the microprocessor's data granularity. It is the storage level out of which the microprocessor could conceivably operate, i.e., it is the storage level that can provide random access to its storage, one data word at a time. This storage level is typically called "main memory." Disks cannot serve as main memory or operating store and cannot provide random access for two reasons: instant access is provided for only the data underneath the disk's head at any given moment, and the granularity of access is not what a typical processor requires. Disks are block-oriented devices, which means they read and write data only in large chunks; the typical granularity is 512 B. Processors, in contrast, typically operate at the granularity of 4 B or 8 B data words. To use a disk, a microprocessor must have additional buffering memory out of which it can read one instruction at a time and read or write one datum at a time. This buffering memory would become the *de facto* operating store of the system.

Flash memory and EEPROM (as well as the exotic non-volatile technologies mentioned earlier) are potentially viable as an operating store for systems that have small permanent-storage needs, and the non-volatility of these technologies provides them with a distinct advantage. However, not all are set up as an ideal operating store; for example, flash memory supports word-sized reads but supports only block-sized writes. If this type of issue can be handled in a manner that is transparent to the processor (e.g., in this case through additional data buffering), then the memory technology can still serve as a reasonable hybrid operating store.

Though the non-volatile technologies seem positioned perfectly to serve as operating store in all manner of devices and systems, DRAM is the most commonly used technology. Note that the only requirement of a memory system's operating store is that it provide random access with a small access granularity. Non-volatility is not a requirement, so long as it is provided by another level in the hierarchy. DRAM is a popular choice for operating store for several reasons: DRAM is faster than the various non-volatile technologies (in some cases *much* faster); DRAM supports an unlimited number of writes, whereas some non-volatile technologies start to fail after being erased and rewritten too many times (in some technologies, as few as 1–10,000 erase/write cycles); and DRAM processes are very similar to those used to build logic devices.

[3]BIOS = basic input/output system, the code that provides to software low-level access to much of the hardware.

DRAM can be fabricated using similar materials and (relatively) similar silicon-based process technologies as most microprocessors, whereas many of the various non-volatile technologies require new materials and (relatively) different process technologies.

Fast (and Relatively Low-Power) Store

If these storage technologies provide such reasonable operating store, why, then, do modern systems use cache? Cache is inserted between the processor and the main memory system whenever the access behavior of the main memory is not sufficient for the needs or goals of the system. Typical figures of merit include performance and energy consumption (or power dissipation). If the performance when operating out of main memory is insufficient, cache is interposed between the processor and main memory to decrease the average access time for data. Similarly, if the energy consumed when operating out of main memory is too high, cache is interposed between the processor and main memory to decrease the system's energy consumption.

The data in Table Ov.1 should give some intuition about the design choice. If a cache can reduce the number of accesses made to the next level down in the hierarchy, then it potentially reduces both execution time and energy consumption for an application. The gain is only potential because these numbers are valid only for certain technology parameters. For example, many designs use large SRAM caches that consume much more energy than several DRAM chips combined, but because the caches can reduce execution time they are used in systems where performance is critical, even at the expense of energy consumption.

It is important to note at this point that, even though the term "cache" is usually interpreted to mean SRAM, a cache is merely a concept and as such imposes no expectations on its implementation. Caches are best thought of as compact databases, as shown in Figure Ov.2. They contain data and, optionally, metadata such as the unique ID (address) of each data block in the array, whether it has been updated recently, etc. Caches can be built from SRAM, DRAM, disk, or virtually any storage technology. They can be managed completely in hardware and thus can be transparent to the running application and even to the memory system itself; and at the other extreme they can be explicitly managed by the running application. For instance, Figure Ov.2 shows that there is an optional block of metadata, which if implemented in hardware would be called the cache's *tags*. In that instance, a key is passed to the tags array, which produces either the location of the corresponding item in the data array (a *cache hit*) or an indication that the item is not in the data array (a *cache miss*). Alternatively, software can be written to index the array explicitly, using direct cache-array addresses, in which case the key lookup (as well as its associated tags array) is unnecessary. The configuration chosen for the cache is called its *organization*. Cache organizations exist at all spots along the continuum between these two extremes. Clearly, the choice of organization will significantly impact the cache's performance and energy consumption.

Predictability of access time is another common figure of merit. It is a special aspect of performance that is very important when building real-time systems or systems with highly orchestrated data movement. DRAM is occasionally in a state where it needs to ignore external requests so that it can guarantee the integrity of its stored data (this is called *refresh* and will be discussed in detail in Part II of the book). Such hiccups in data movement can be disastrous for some applications. For this reason, many microprocessors, such as digital signal processors (DSPs) and processors used in embedded control applications (called *microcontrollers*), often

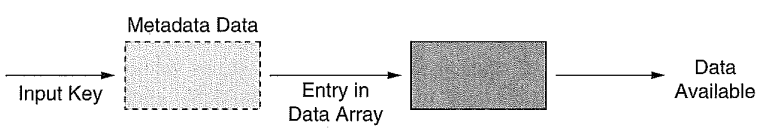


FIGURE Ov.2: An idealized cache lookup. A cache is logically comprised of two elements: the data array and some management information that indicates what is in the data array (labeled "metadata"). Note that the key information may be virtual, i.e., data addresses can be embedded in the software using the cache, in which case there is no explicit key lookup, and only the data array is needed.

have special caches that look like small main memories. These are *scratch-pad RAMs* whose implementation lies toward the end of the spectrum at which the running application manages the cache explicitly. DSPs typically have two of these scratch-pad SRAMs so that they can issue on every cycle a new *multiply-accumulate (MAC)* operation, an important DSP instruction whose repeated operation on a pair of data arrays produces its dot product. Performing a new MAC operation every cycle requires the memory system to load new elements from two different arrays simultaneously in the same cycle. This is most easily accomplished by having two separate data busses, each with its own independent data memory and each holding the elements of a different array.

Perhaps the most familiar example of a software-managed memory is the processor's *register file*, an array of storage locations that is indexed directly by bits within the instruction and whose contents are dictated entirely by software. Values are brought into the register file explicitly by software instructions, and old values are only overwritten if done so explicitly by software. Moreover, the register file is significantly smaller than most on-chip caches and typically consumes far less energy. Accordingly, software's best bet is often to optimize its use of the register file [Postiff & Mudge 1999].

Ov.1.2 Important Figures of Merit

The following issues have been touched on during the previous discussion, but at this point it would be valuable to formally present the various figures of merit that are important to a designer of memory systems. Depending on the environment in which the memory system will be used (supercomputer, departmental server, desktop, laptop, signal-processing system, embedded control system, etc.), each metric will carry more or less weight. Though most academic studies tend to focus on one axis at a time (e.g., performance), the design of a memory system is a multi-dimensional optimization problem, with all the adherent complexities of analysis. For instance, to analyze something in this design space or to consider one memory system over another, a designer should be familiar with concepts such as Pareto optimality (described later in this chapter). The various figures of merit, in no particular order other than performance being first due to its popularity, are performance, energy consumption and power dissipation, predictability of behavior (i.e., real time), manufacturing costs, and system reliability. This section describes them briefly, collectively. Later sections will treat them in more detail.

Performance

The term "performance" means many things to many people. The performance of a system is typically measured in the time it takes to execute a task (i.e., task *latency*), but it can also be measured in the number of tasks that can be handled in a unit time period (i.e., task *bandwidth*). Popular figures of merit for performance include the following:[4]

- Cycles per Instruction (CPI)
  $$= \frac{\text{Total execution cycles}}{\text{Total user-level instructions committed}}$$
- Memory-system CPI overhead
  = Real CPI – CPI assuming perfect memory
- Memory Cycles per Instruction (MCPI)
  $$= \frac{\text{Total cycles spent in memory system}}{\text{Total user-level instructions committed}}$$
- Cache miss rate $= \frac{\text{Total cache misses}}{\text{Total cache accesses}}$
- Cache hit rate = 1 – Cache miss rate
- Average access time
  = (hit rate · average to service hit) + (miss rate · average to service miss)
- Million Instructions per Second (MIPS)
  $$= \frac{\text{Instructions executed (seconds)}}{10^6 \cdot \text{Average required for execution}}$$

[4]Note that the MIPS metric is easily abused. For instance, it is inappropriate for comparing different instruction-set architectures, and marketing literature often takes the definition of "instructions executed" to mean any particular given window of time as opposed to the full execution of an application. In such cases, the metric can mean the highest possible issue rate of instructions that the machine can achieve (but not necessarily sustain for any realistic period of time).

A cautionary note: using a metric of performance for the memory system that is independent of a processing context can be very deceptive. For instance, the MCPI metric does not take into account how much of the memory system's activity can be overlapped with processor activity, and, as a result, memory system A which has a worse MCPI than memory system B might actually yield a computer system with better total performance. As Figure Ov.5 in a later section shows, there can be significantly different amounts of overlapping activity between the memory system and CPU execution.

How to average a set of performance metrics correctly is still a poorly understood topic, and it is very sensitive to the weights chosen (either explicitly or implicitly) for the various benchmarks considered [John 2004]. Comparing performance is always the least ambiguous when it means the amount of time saved by using one design over another. When we ask the question *this machine is how much faster than that machine?* the implication is that we have been using *that* machine for some time and wish to know how much time we would save by using *this* machine instead. The true measure of performance is to compare the total execution time of one machine to another, with each machine running the benchmark programs that represent the user's typical workload as often as a user expects to run them. For instance, if a user compiles a large software application ten times per day and runs a series of regression tests once per day, then the total execution time should count the compiler's execution ten times more than the regression test.

Energy Consumption and Power Dissipation

Energy consumption is related to work accomplished (e.g., how much computing can be done with a given battery), whereas power dissipation is the rate of consumption. The instantaneous power dissipation of CMOS (complementary metal-oxide-semiconductor) devices, such as microprocessors, is measured in watts (W) and represents the sum of two components: *active power*, due to switching activity, and *static power*, due primarily to subthreshold leakage. To a first approximation, average power dissipation is equal to the following (we will present a more detailed model later):

$$P_{avg} = (P_{dynamic} + P_{static}) \equiv C_{tot} V^2_{dd} f + I_{leak} V_{dd} \quad \textbf{(EQ Ov.1)}$$

where C_{tot} is the total capacitance switched, V_{dd} is the power supply, f is the switching frequency, and I_{leak} is the leakage current, which includes such sources as subthreshold and gate leakage. With each generation in process technology, active power is decreasing on a device level and remaining roughly constant on a chip level. Leakage power, which used to be insignificant relative to switching power, increases as devices become smaller and has recently caught up to switching power in magnitude [Grove 2002]. In the future, leakage will be the primary concern.

Energy is related to power through time. The energy consumed by a computation that requires T seconds is measured in joules (J) and is equal to the integral of the instantaneous power over time T. If the power dissipation remains constant over T, the resultant energy consumption is simply the product of power and time.

$$E = (P_{avg} \cdot T) \equiv C_{tot} V^2_{dd} N + I_{leak} V_{dd} T \quad \textbf{(EQ Ov.2)}$$

where N is the number of switching events that occurs during the computation.

In general, if one is interested in extending battery life or reducing the electricity costs of an enterprise computing center, then *energy* is the appropriate metric to use in an analysis comparing approaches. If one is concerned with heat removal from a system or the thermal effects that a functional block can create, then *power* is the appropriate metric. In informal discussions (i.e., in common-parlance prose rather than in equations where units of measurement are inescapable), the two terms "power" and "energy" are frequently used interchangeably, though such use is technically incorrect. Beware, because this can lead to ambiguity and even misconception, which is usually unintentional, but not always so. For instance, microprocessor manufacturers will occasionally claim to have a "low-power" microprocessor that beats its predecessor by a factor of, say, two. This is easily accomplished by running the microprocessor at half the clock rate, which does reduce its power dissipation,

but remember that power is the rate at which energy is consumed. However, to a first order, doing so doubles the time over which the processor dissipates that power. The net result is a processor that consumes the same amount of *energy* as before, though it is branded as having lower *power*, which is technically not a lie.

Popular figures of merit that incorporate both energy/power and performance include the following:

- Energy-Delay Product
$$= \left(\begin{array}{l}\text{Energy required}\\ \text{to perform task}\end{array}\right) \cdot \left(\begin{array}{l}\text{Time required}\\ \text{to perform task}\end{array}\right)$$
- Power-Delay Product
$$= \left(\begin{array}{l}\text{Power required}\\ \text{to perform task}\end{array}\right)^m \cdot \left(\begin{array}{l}\text{Time required}\\ \text{to perform task}\end{array}\right)^n$$
- MIPS per watt
$$= \frac{\text{Performance of benchmark in MIPS}}{\text{Average power dissipated by benchmark}}$$

The second equation was offered as a generalized form of the first (note that the two are equivalent when m = 1 and n = 2) so that designers could place more weight on the metric (time or energy/power) that is most important to their design goals [Gonzalez & Horowitz 1996, Brooks et al. 2000a].

Predictable (Real-Time) Behavior

Predictability of behavior is extremely important when analyzing real-time systems, because correctness of operation is often the primary design goal for these systems (consider, for example, medical equipment, navigation systems, anti-lock brakes, flight control systems, etc., in which failure to perform as predicted is not an option).

Popular figures of merit for expressing predictability of behavior include the following:

- Worst-Case Execution Time (WCET), taken to mean the longest amount of time a function could take to execute
- Response time, taken to mean the time between a stimulus to the system and the system's response (e.g., time to respond to an external interrupt)
- Jitter, the amount of deviation from an average timing value

These metrics are typically given as single numbers (average or worst case), but we have found that the probability density function makes a valuable aid in system analysis [Baynes et al. 2001, 2003].

Design (and Fabrication and Test) Costs

Cost is an obvious, but often unstated, design goal. Many consumer devices have cost as their primary consideration: if the cost to design and manufacture an item is not low enough, it is not worth the effort to build and sell it. Cost can be represented in many different ways (note that energy consumption is a measure of cost), but for the purposes of this book, by "cost" we mean the cost of producing an item: to wit, the cost of its design, the cost of testing the item, and/or the cost of the item's manufacture. Popular figures of merit for cost include the following:

- Dollar cost (best, but often hard to even approximate)
- Design size, e.g., die area (cost of manufacturing a VLSI (very large scale integration) design is proportional to its area cubed or more)
- Packaging costs, e.g., pin count
- Design complexity (can be expressed in terms of number of logic gates, number of transistors, lines of code, time to compile or synthesize, time to verify or run DRC (design-rule check), and many others, including a design's impact on clock cycle time [Palacharla et al. 1996])

Cost is often presented in a relative sense, allowing differing technologies or approaches to be placed on equal footing for a comparison.

- Cost per storage bit/byte/KB/MB/etc. (allows cost comparison between different storage technologies)
- Die area per storage bit (allows size-efficiency comparison within same process technology)

In a similar vein, cost is especially informative when combined with performance metrics. The following are variations on the theme:

- Bandwidth per package pin (total sustainable bandwidth to/from part, divided by total number of pins in package)
- Execution-time-dollars (total execution time multiplied by total cost; note that cost can be expressed in other units, e.g., pins, die area, etc.)

An important note: cost should incorporate *all* sources of that cost. Focusing on just one source of cost blinds the analysis in two ways: first, the true cost of the system is not considered, and second, solutions can be unintentionally excluded from the analysis. If cost is expressed in pin count, then all pins should be considered by the analysis; the analysis should not focus solely on data pins, for example. Similarly, if cost is expressed in die area, then all sources of die area should be considered by the analysis; the analysis should not focus solely on the number of banks, for example, but should also consider the cost of building control logic (decoders, muxes, bus lines, etc.) to select among the various banks.

Reliability

Like the term "performance," the term "reliability" means many things to many different people. In this book, we mean reliability of the data stored within the memory system: how easily is our stored data corrupted or lost, and how can it be protected from corruption or loss? Data integrity is dependent upon physical devices, and physical devices can fail.

Approaches to guarantee the integrity of stored data typically operate by storing redundant information in the memory system so that in the case of device failure, some but not all of the data will be lost or corrupted. If enough redundant information is stored, then the missing data can be reconstructed. Popular figures of merit for measuring reliability characterize both device fragility and robustness of a proposed solution. They include the following:

- Mean Time Between Failures (MTBF): [5] given in time (seconds, hours, etc.) or number of uses
- Bit-error tolerance, e.g., how many bit errors in a data word or packet the mechanism can correct, and how many it can detect (but not necessarily correct)
- Error-rate tolerance, e.g., how many errors per second in a data stream the mechanism can correct
- Application-specific metrics, e.g., how much radiation a design can tolerate before failure, etc.

Note that values given for MTBF often seem astronomically high. This is because they are not meant to apply to individual devices, but to system-wide device use, as in a large installation. For instance, if the expected service lifetime of a device is several years, then that device is expected to fail in several years. If an administrator swaps out devices every few years (before the service lifetime is up), then the administrator should expect to see failure frequencies consistent with the MTBF rating.

Ov.1.3 The Goal of a Memory Hierarchy

As already mentioned, a well-implemented hierarchy allows a memory system to approach simultaneously the performance of the fastest component, the cost per bit of the cheapest component, and the energy consumption of the most energy-efficient component. A modern memory system typically has performance close to that of on-chip cache, the fastest component in the system. The rate at which microprocessors fetch and execute their instructions is measured in nanoseconds or fractions of a nanosecond. A modern low-end desktop machine has several hundred gigabytes of storage and sells for under $500, roughly half of which goes to the on-chip caches, off-chip caches, DRAM, and disk. This represents an average cost of

[5] A common variation is "Mean Time To Failure (MTTF)."

several dollars per gigabyte—very close to that of disk, the cheapest component. Modern desktop systems have an energy cost that is typically in the low tens of nanojoules per instruction executed—close to that of on-chip SRAM cache, the least energy-costly component in the system (on a per-access basis).

The goal for a memory-system designer is to create a system that behaves, on average and from the point of view of the processor, like a big cache that has the price tag of a disk. A successful memory hierarchy is much more than the sum of its parts; moreover, successful memory-system design is non-trivial.

How the system is built, how it is used (and what parts of it are used more heavily than others), and on which issues an engineer should focus most of his effort at design time—all these are highly dependent on the target application of the memory system. Two common categories of target applications are (a) general-purpose systems, which are characterized by their need for universal applicability for just about any type of computation, and (b) embedded systems, which are characterized by their tight design restrictions along multiple axes (e.g., cost, correctness of design, energy consumption, reliability) and the fact that each executes only a single, dedicated software application its entire lifespan, which opens up possibilities for optimization that are less appropriate for general-purpose systems.

General-Purpose Computer Systems

General-purpose systems are what people normally think of as "computers." These are the machines on your desktop, the machines in the refrigerated server room at work, and the laptop on the kitchen table. They are designed to handle any and all tasks thrown at them, and the software they run on a day-to-day basis is radically different from machine to machine.

General-purpose systems are typically overbuilt. By definition they are expected by the consumer to run all possible software applications with acceptable speed, and therefore, they are built to handle the average case very well and the worst case at least tolerably well. Were they optimized for any particular task, they could easily become less than optimal for all dissimilar tasks. Therefore, general-purpose systems are optimized for everything, which is another way of saying that they are actually optimized for nothing in particular. However, they make up for this in raw performance, pure number-crunching. The average notebook computer is capable of performing orders of magnitude more operations per second than that required by a word processor or email client, tasks to which the average notebook is frequently relegated, but because the general-purpose system may be expected to handle virtually anything at any time, it must have significant spare number-crunching ability, just in case.

It stands to reason that the memory system of this computer must also be designed in a Swiss-army-knife fashion. Figure Ov.3 shows the organization of a typical personal computer, with the components of the memory system highlighted in grey boxes. The cache levels are found both on-chip (i.e., integrated on the same die as the microprocessor core) and off-chip (i.e., on a separate die). The DRAM system is comprised of a memory controller and a number of DRAM chips organized into DIMMs (dual in-line memory modules, printed circuit boards that contain a handful of DRAMs each). The memory controller can be located on-chip or off-chip, but the DRAMs are always separate from the CPU to allow memory upgrades. The disks in the system are considered peripheral devices, and so their access is made through one or more levels of controllers, each representing a potential chip-to-chip crossing (e.g., here a disk request passes through the system controller to the PCI (peripheral component interconnect) bus controller, to the SCSI (small computer system interface) controller, and finally to the disk itself).

The software that runs on a general-purpose system typically executes in the context of a robust operating system, one that provides virtual memory. Virtual memory is a mechanism whereby the operating system can provide to all running user-level software (i.e., email clients, web browsers, spreadsheets, word-processing packages, graphics and video editing software, etc.) the illusion that the user-level software is in direct control of the computer, when in fact its use of the computer's resources is managed by the operating system. This is a very effective way for an operating system to provide simultaneous access by

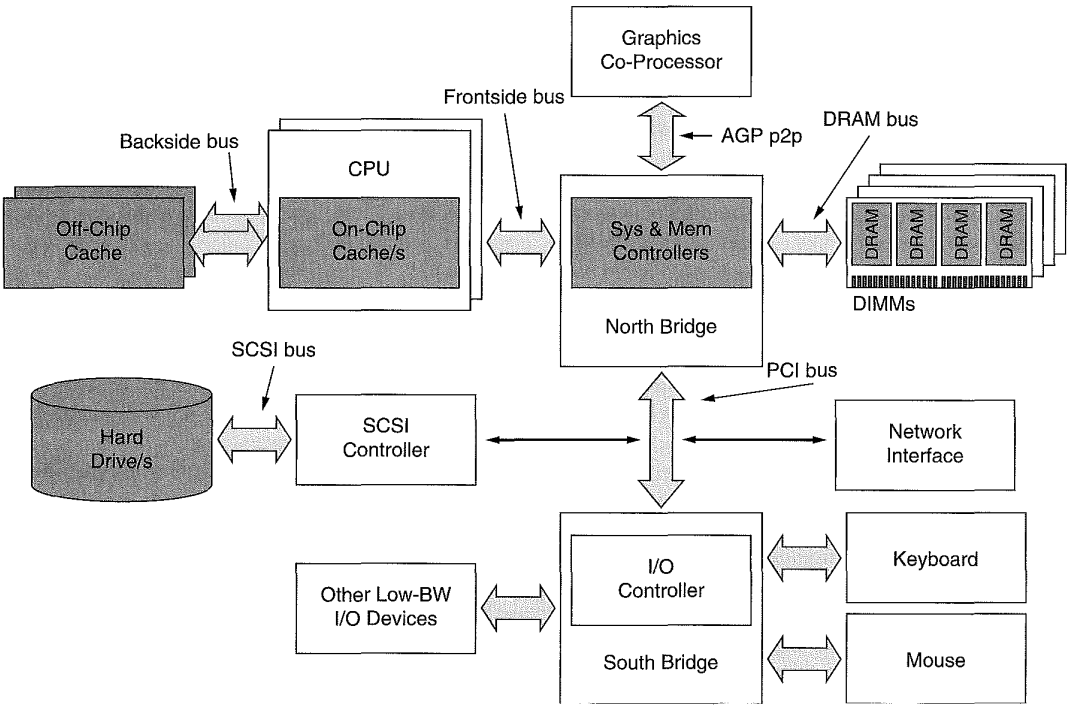


FIGURE Ov.3: Typical PC organization. The memory subsystem is one part of a relatively complex whole. This figure illustrates a two-way multiprocessor, with each processor having its own dedicated off-chip cache. The parts most relevant to this text are shaded in grey: the CPU and its cache system, the system and memory controllers, the DIMMs and their component DRAMs, and the hard drive/s.

large numbers of software packages to small numbers of limited-use resources (e.g., physical memory, the hard disk, the network, etc.).

The virtual memory system is the primary constituent of the memory system, in that it is the primary determinant of the manner/s in which the memory system's components are used by software running on the computer. Permanent data is stored on the disk, and the operating store, DRAM, is used as a cache for this permanent data. This DRAM-based cache is explicitly managed by the operating system. The operating system decides what data from the disk should be kept, what should be discarded, what should be sent back to the disk, and, for data retained, where it should be placed in the DRAM system. The primary and secondary caches are usually transparent to software, which means that they are managed by hardware, not software (note, however, the use of the word "usually"—later sections will delve into this in more detail). In general, the primary and secondary caches hold *demand-fetched* data, i.e., running software demands data, the hardware fetches it from memory, and the caches retain as much of it as possible. The DRAM system contains data that the operating system deems worthy of keeping around, and because fetching data from the disk and writing it back to the disk are such time-consuming processes, the operating system can exploit that lag time (during

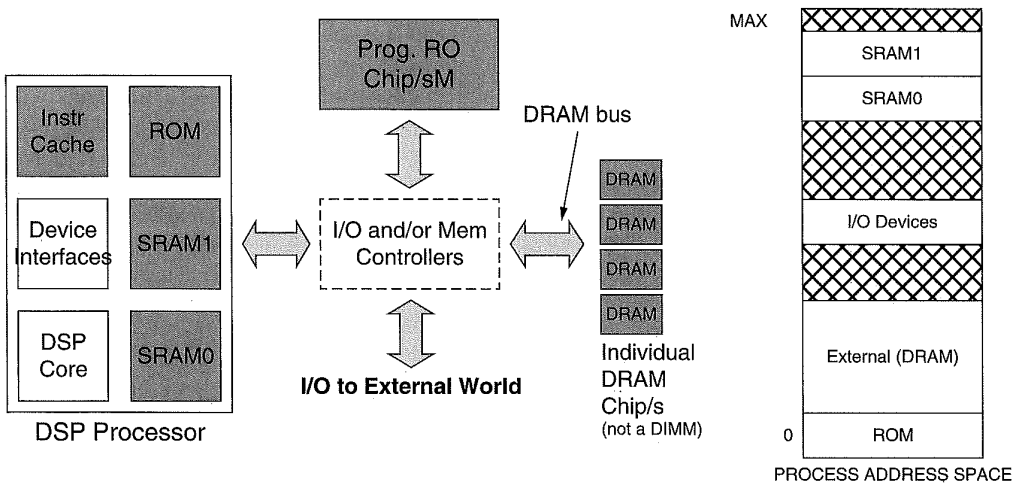


FIGURE Ov.4: DSP-style memory system. Example based on Texas Instruments' TMS320C3x DSP family.

which it would otherwise be stalled, doing nothing) to use sophisticated heuristics to decide what data to retain.

Embedded Computer Systems

Embedded systems differ from general-purpose systems in two main aspects. First and foremost, the two are designed to suit very different purposes. While general-purpose systems run a myriad of unrelated software packages, each having potentially very different performance requirements and dynamic behavior compared to the rest, embedded systems perform a single function their entire lifetime and thus execute the same code day in and day out until the system is discarded or a software upgrade is performed. Second, while performance is the primary (in many instances, the only) figure of merit by which a general-purpose system is judged, optimal embedded-system designs usually represent trade-offs between several goals, including manufacturing cost (e.g., die area), energy consumption, and performance.

As a result, we see two very different design strategies in the two camps. As mentioned, general-purpose systems are typically overbuilt; they are optimized for nothing in particular and must make up for this in raw performance. On the other hand, embedded systems are expected to handle only one task that is known at design time. Thus, it is not only possible, but highly beneficial to optimize an embedded design for its one suited task. If general-purpose systems are *overbuilt*, the goal for an embedded system is to be *appropriately* built. In addition, because effort spent at design time is amortized over the life of a product, and because many embedded systems have long lifetimes (tens of years), many embedded design houses will expend significant resources up front to optimize a design, using techniques not generally used in general-purpose systems (for instance, compiler optimizations that require many days or weeks to perform).

The memory system of a typical embedded system is less complex than that of a general-purpose system.[6] Figure Ov.4 illustrates an average digital signal-processing system with dual tagless SRAMs on-chip,

[6]Note that "less complex" does not necessarily imply "small," e.g., consider a typical iPod (or similar MP3 player), whose primary function is to store gigabytes' worth of a user's music and/or image files.

an off-chip programmable ROM (e.g., PROM, EPROM, flash ROM, etc.) that holds the executable image, and an off-chip DRAM that is used for computation and holding variable data. External memory and device controllers can be used, but many embedded microprocessors already have such controllers integrated onto the CPU die. This cuts down on the system's die count and thus cost. Note that it would be possible for the entire hierarchy to lie on the CPU die, yielding a single-chip solution called a *system-on-chip*. This is relatively common for systems that have limited memory requirements. Many DSPs and microcontrollers have programmable ROM embedded within them. Larger systems that require megabytes of storage (e.g., in Cisco routers, the instruction code alone is more than a 12 MB) will have increasing numbers of memory chips in the system.

On the right side of Figure Ov.4 is the software's view of the memory system. The primary distinction is that, unlike general-purpose systems, is that the SRAM caches are visible as separately addressable memories, whereas they are transparent to software in general-purpose systems.

Memory, whether SRAM or DRAM, usually represents one of the more costly components in an embedded system, especially if the memory is located on-CPU because once the CPU is fabricated, the memory size cannot be increased. In nearly all system-on-chip designs and many microcontrollers as well, memory accounts for the lion's share of available die area. Moreover, memory is one of the primary consumers of energy in a system, both on-CPU and off-CPU. As an example, it has been shown that, in many digital signal-processing applications, the memory system consumes more of both energy and die area than the processor datapath. Clearly, this is a resource on which significant time and energy is spent performing optimization.

Ov.2 Four Anecdotes on Modular Design

It is our observation that computer-system design in general, and memory-hierarchy design in particular, has reached a point at which it is no longer sufficient to design and optimize subsystems in isolation. Because memory systems and their subsystems are so complex, it is now the rule, and not the exception, that the subsystems we thought to be independent actually interact in unanticipated ways. Consequently, our traditional design methodologies no longer work because their underlying assumptions no longer hold. Modular design, one of the most widely adopted design methodologies, is an oft-praised engineering design principle in which clean functional interfaces separate subsystems (i.e., modules) so that subsystem design and optimization can be performed independently and in parallel by different designers. Applying the principles of modular design to produce a complex product can reduce the time and thus the cost for system-level design, integration, and test; optimization at the modular level guarantees optimization at the system level, provided that the system-level architecture and resulting module-to-module interfaces are optimal.

That last part is the sticking point: the principle of modular design assumes no interaction between module-level implementations and the choice of system-level architecture, but that is exactly the kind of interaction that we have observed in the design of modern, high-performance memory systems. Consequently, though modular design has been a staple of memory-systems design for decades, allowing cache designers to focus solely on caches, DRAM designers to focus solely on DRAMs, and disk designers to focus solely on disks, we find that, going forward, modular design is no longer an appropriate methodology.

Earlier we noted that, in the design of memory systems, many of the underlying implementation issues have begun to affect the higher level design process quite significantly: cache design is driven by interconnect physics; DRAM design is driven by circuit-level limitations that have dramatic system-level effects; and modern disk performance is dominated by the on-board caching and scheduling policies. As hierarchies and their components grow more complex, we find that the bulk of performance is lost not in the CPUs or caches or DRAM devices or disk assemblies themselves, but in the subtle interactions between these subsystems and in the manner in which these subsystems are connected. The bulk of lost

performance is due to poor configuration of system-level parameters such as bus widths, granularity of access, scheduling policies, queue organizations, and so forth.

This is extremely important, so it bears repeating: the bulk of lost performance is not due to the number of CPU pipeline stages or functional units or choice of branch prediction algorithm or even CPU clock speed; the bulk of lost performance is due to poor configuration of system-level parameters such as bus widths, granularity of access, scheduling policies, queue organizations, etc. Today's computer-system performance is dominated by the manner in which data is moved between subsystems, i.e., the scheduling of transactions, and so it is not surprising that seemingly insignificant details can cause such a headache, as scheduling is known to be highly sensitive to such details.

Consequently, one can no longer attempt system-level optimization by designing/optimizing each of the parts in isolation (which, unfortunately, is often the approach taken in modern computer design). In subsystem design, nothing can be considered "outside the scope" and thus ignored. Memory-system design must become the purview of architects, and a subsystem designer must consider the system-level ramifications of even the slightest low-level design decision or modification. In addition, a designer must understand the low-level implications of system-level design choices. A simpler form of this maxim is as follows:

> *A designer must consider the system-level ramifications of circuit- and device-level decisions as well as the circuit- and device-level ramifications of system-level decisions.*

To illustrate what we mean and to motivate our point, we present several anecdotes. Though they focus on the DRAM system, their message is global, and we will show over the course of the book that the relationships they uncover are certainly not restricted to the DRAM system alone. We will return to these anecdotes and discuss them in much more detail in Chapter 27, *The Case for Holistic Design*, which follows the technical section of the book.

Ov.2.1 Anecdote I: Systemic Behaviors Exist

In 1999–2001, we performed a study of DRAM systems in which we explicitly studied only system-level effects—those that had nothing to do with the CPU architecture, DRAM architecture, or even DRAM interface protocol. In this study, we held constant the CPU and DRAM architectures and considered only a handful of parameters that would affect how well the two communicate with each other. Figure Ov.5 shows some of the results [Cuppu & Jacob 1999, 2001, Jacob 2003]. The varied parameters in Figure Ov.5 are all seemingly innocuous parameters, certainly not the type that would account for up to 20% differences in system performance (execution time) if one parameter was increased or decreased by a small amount, which is indeed the case. Moreover, considering the top two graphs, all of the choices represent intuitively "good" configurations. None of the displayed values represent strawmen, machine configurations that one would avoid putting on one's own desktop. Nonetheless, the performance variability is significant. When the analysis considers a wider range of bus speeds and burst lengths, the problematic behavior increases. As shown in the bottom graph, the ratio of best to worst execution times can be a factor of three, and the local optima are both more frequent and more exaggerated. Systems with relatively low bandwidth (e.g., 100, 200, 400 MB/s) and relatively slow bus speeds (e.g., 100, 200 MHz), if configured well, can match or exceed the performance of system configurations with much faster hardware that is poorly configured.

Intuitively, one would expect the design space to be relatively smooth: as system bandwidth increases, so should system performance. Yet the design space is far from smooth. Performance variations of 20% or more can be found in design points that are immediately adjacent to one another. The variations from best-performing to worst-performing design exceed a factor of three across the full space studied, and local minima and maxima abound. Moreover, the behaviors are related. Increasing one parameter by a factor of two toward higher expected performance (e.g., increasing the channel width) can move the system off a local optimum, but local optimality can be restored by changing other related parameters to follow suit,

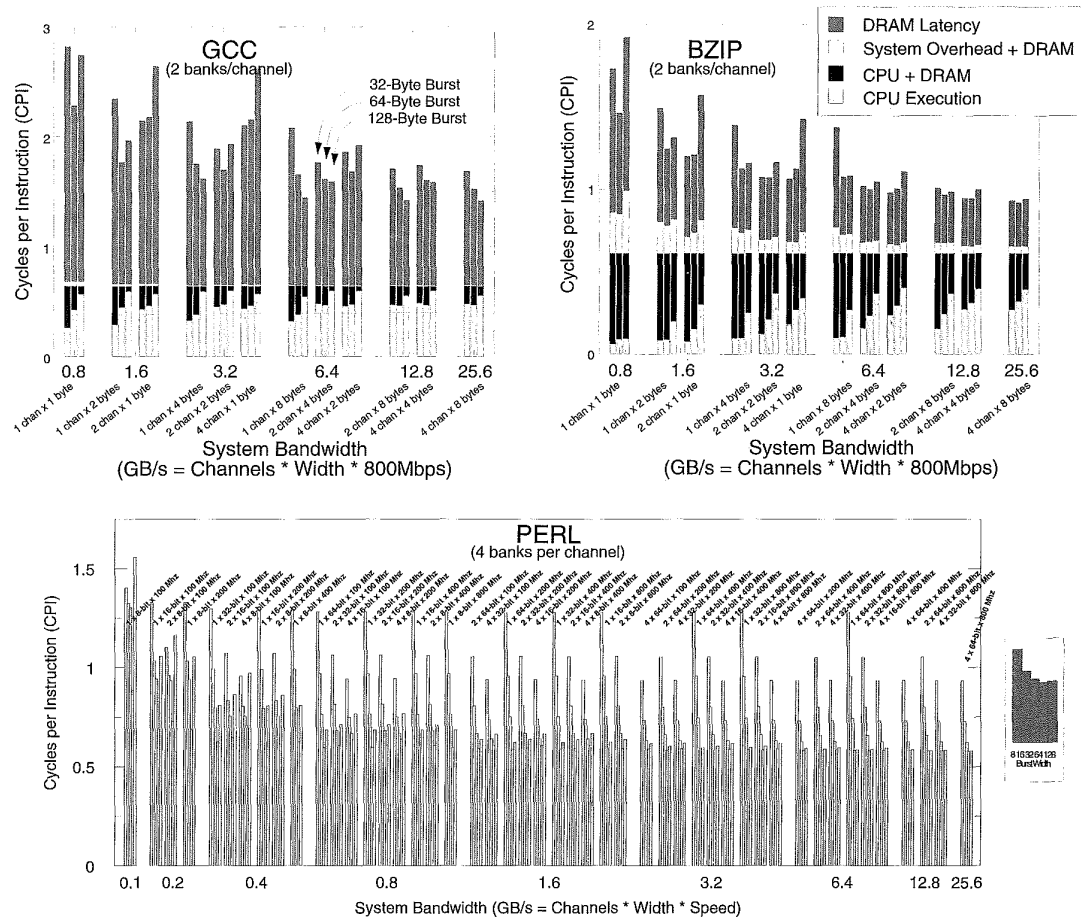


FIGURE 0v.5: Execution time as a function of bandwidth, channel organization, and granularity of access. Top two graphs from Cuppu & Jacob [2001] (© 2001 *IEEE*); bottom graph from Jacob [2003] (© 2003 *IEEE*).

such as increasing the burst length and cache block size to match the new channel width. This complex interaction between parameters previously thought to be independent arises because of the complexity of the system under study, and so we have named these "systemic" behaviors.[7] This study represents the moment we realized that systemic behaviors exist and that they are significant. Note that the behavior

[7]There is a distinction between this type of behavior and what in complex system theory is called "emergent system" behaviors or properties. Emergent system behaviors are those of individuals within a complex system, behaviors that an individual may perform in a group setting that the individual would never perform alone. In our environment, the behaviors are observations we have made of the design space, which is derived from the system as a whole.

is not restricted to the DRAM system. We have seen it in the disk system as well, where the variations in performance from one configuration to the next are even more pronounced.

Recall that this behavior comes from the varying of parameters that are seemingly unimportant in the grand scheme of things—at least they would certainly seem to be far less important than, say, the cache architecture or the number of functional units in the processor core. The bottom line, as we have observed, is that systemic behaviors—unanticipated interactions between seemingly innocuous parameters and mechanisms—cause significant losses in performance, requiring in-depth, detailed design-space exploration to achieve anything close to an optimal design given a set of technologies and limitations.

Ov.2.2 Anecdote II: The DLL in DDR SDRAM

Beginning with their first generation, DDR (double data rate) SDRAM devices have included a circuit-level mechanism that has generated significant controversy within JEDEC (Joint Electron Device Engineering Council), the industry consortium that created the DDR SDRAM standard. The mechanism is a delay-locked loop (DLL), whose purpose is to more precisely align the output of the DDR part with the clock on the system bus. The controversy stems from the cost of the technology versus its benefits.

The system's global clock signal, as it enters the chip, is delayed by the DLL so that the chip's internal clock signal, after amplification and distribution across the chip, is exactly in-phase with the original system clock signal. This more precisely aligns the DRAM part's output with the system clock. The trade-off is extra latency in the datapath as well as a higher power and heat dissipation because the DLL, a dynamic control mechanism, is continuously running. By aligning each DRAM part in a DIMM to the system clock, each DRAM part is effectively de-skewed with respect to the other parts, and the DLLs cancel out timing differences due to process variations and thermal gradients.

Figure Ov.6 illustrates a small handful of alternative solutions considered by JEDEC, who ultimately chose Figure Ov.6(b) for the standard. The interesting thing is that the data strobe is not used to capture data at the memory controller, bringing into question its purpose if the DLL is being used to help with data transfer to the memory controller. There is significant disagreement over the value of the chosen design; an anonymous JEDEC member, when

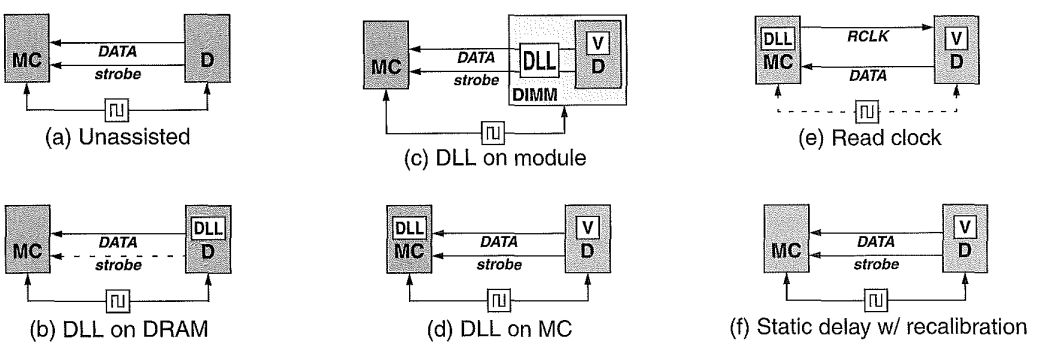


FIGURE Ov.6: Several alternatives to the per-DRAM DLL. The figure illustrates a half dozen different timing conventions (a dotted line indicates a signal is unused for capturing data): (a) the scheme in single data rate SDRAM; (b) the scheme chosen for DDR SDRAM; (c) moving the DLL onto the module, with a per-DRAM static delay element (Vernier); (d) moving the DLL onto the memory controller, with a per-DRAM static delay; (e) using a separate read clock per DRAM or per DIMM; and (f) using only a static delay element and recalibrating periodically to address dynamic changes.

asked "what is the DLL doing on the DDR chip?" answered with a grin, "burning power." In applications that require low latency and low power dissipation, designers turn off the DLL entirely and use only the data strobe for data capture, ignoring the system clock (as in Figure Ov.6(a)) [Kellogg 2002, Lee 2002, Rhoden 2002].

The argument for the DLL is that it de-skews the DRAM devices on a DIMM and provides a path for system design that can use a global clocking scheme, one of the simplest system designs known. The argument against the DLL is that it would be unnecessary if a designer learned to use the data strobe—this would require a more sophisticated system design, but it would achieve better performance at a lower cost. At the very least, it is clear that a DLL is a circuit-oriented solution to the problem of system-level skew, which could explain the controversy.

Ov.2.3 Anecdote III: A Catch-22 in the Search for Bandwidth

With every DRAM generation, timing parameters are added. Several have been added to the DDR specification to address the issues of power dissipation and synchronization.

- t_{FAW} (*Four-bank Activation Window*) and t_{RRD} (*Row-to-Row activation Delay*) put a ceiling on the maximum current draw of a single DRAM part. These are protocol-level limitations whose values are chosen to prevent a memory controller from exceeding circuit-related thresholds.
- t_{DQS} is our own name for the DDR system-bus turnaround time; one can think of it as the DIMM-to-DIMM switching time that has implications only at the system level (i.e., it has no meaning or effect if considering read requests in a system with but a single DIMM). By obeying t_{DQS}, one can ensure that a second DIMM will not drive the data bus at the same time as a first when switching from one DIMM to another for data output.

These are per-device timing parameters that were chosen to improve the behavior (current draw, timing uncertainty) of individual devices. However, they do so at the expense of a significant loss in system-level performance. When reading large amounts of data from the DRAM system, an application will have to read, and thus will have to *activate*, numerous DRAM rows. At this point, the t_{FAW} and t_{RRD} timing parameters kick in and limit the available read bandwidth. The t_{RRD} parameter specifies the minimum time between two successive row activation commands to the same DRAM device (which implies the same DIMM, because all the DRAMs on a DIMM are slaved together[8]). The t_{FAW} parameter represents a sliding window of time during which no more than four row activation commands to the same device may appear.

The parameters are specified in nanoseconds and not bus cycles, so they become increasingly problematic at higher bus frequencies. Their net effect is to limit the bandwidth available from a DIMM by limiting how quickly one can get the data out of the DRAM's storage array, irrespective of how fast the DRAM's I/O circuitry can ship the data back to the memory controller. At around 1 GBps, sustainable bandwidth hits a ceiling and remains flat no matter how fast the bus runs because the memory controller is limited in how quickly it can activate a new row and start reading data from it.

The obvious solution is to interleave data from different DIMMs on the bus. If one DIMM is limited in how quickly it can read data from its arrays, then one should populate the bus with many DIMMs and move through them in a round-robin fashion. This should bring the system bandwidth up to maximum. However, the function of t_{DQS} is to prevent exactly that: t_{DQS} is the bus turnaround time, inserted to account for skew on the bus and to prevent different bus masters from driving the bus at the same time.

[8]This is a minor oversimplification. We would like to avoid having to explain details of DRAM-system organization, such as the concept of *rank*, at this point.

To avoid such collisions, a second DIMM must wait at least t_{DQS} after a first DIMM has finished before driving the bus. So we have a catch:

- One set of parameters limits device-level bandwidth and expects a designer to go to the system level to reclaim performance.
- The other parameter limits system-level bandwidth and expects a designer to go to the device level to reclaim performance.

The good news is that the problem is solvable (see Chapter 15, Section 15.4.3, *DRAM Command Scheduling Algorithms*), but this is nonetheless a very good example of low-level design decisions that create headaches at the system level.

Ov.2.4 Anecdote IV: Proposals to Exploit Variability in Cell Leakage

The last anecdote is an example of a system-level design decision that ignores circuit- and device-level implications. Ever since DRAM was invented, it has been observed that different DRAM cells exhibit different data-retention time characteristics, typically ranging between hundreds of milliseconds to tens of seconds. DRAM manufacturers typically set the refresh requirement conservatively and require that every row in a DRAM device be refreshed at least once every 64 or 32 ms to avoid losing data. Though refresh might not seem to be a significant concern, in mobile devices researchers have observed that refresh can account for one-third of the power in otherwise idle systems, prompting action to address the issue. Several recent papers propose moving the refresh function into the memory controller and refreshing each row only when needed. During an initialization phase, the controller would characterize each row in the memory system, measuring DRAM data-retention time on a row-by-row basis, discarding leaky rows entirely, limiting its DRAM use to only those rows deemed non-leaky, and refreshing once every tens of seconds instead of once every tens of milliseconds.

The problem is that these proposals ignore another, less well-known phenomenon of DRAM cell variability, namely that a cell with a long retention time can suddenly (in the time frame of seconds) exhibit a short retention time [Yaney et al. 1987, Restle et al. 1992, Ueno et al. 1998, Kim 2004]. Such an effect would render these power-efficient proposals functionally erroneous. The phenomenon is called *variable retention time* (VRT), and though its occurrence is infrequent, it is non-zero. The occurrence rate is low enough that a system using one of these reduced-refresh proposals could protect itself against VRT by using error correcting codes (ECC, described in detail in Chapter 30, *Memory Errors and Error Correction*), but none of the proposals so far discuss VRT or ECC.

Ov.2.5 Perspective

To summarize so far:

Anecdote I: Systemic behaviors exist and are significant (they can be responsible for factors of two to three in execution time).

Anecdote II: The DLL in DDR SDRAM is a circuit-level solution chosen to address system-level skew.

Anecdote III: t_{DQS} represents a circuit-level solution chosen to address system-level skew in DDR SDRAM; t_{FAW} and t_{RRD} are circuit-level limitations that significantly limit system-level performance.

Anecdote IV: Several research groups have recently proposed system-level solutions to the DRAM-refresh problem, but fail to account for circuit-level details that might compromise the correctness of the resulting system.

Anecdotes II and III show that a common practice in industry is to focus at the level of devices and circuits, in some cases ignoring their system-level ramifications. Anecdote IV shows that a common practice in research is to design systems that have device- and circuit-level ramifications while abstracting away the details of the devices and circuits involved. Anecdote I

illustrates that both approaches are doomed to failure in future memory-systems design.

It is clear that in the future we will have to move away from modular design; one can no longer safely abstract away details that were previously considered "out of scope." To produce a credible analysis, a designer must consider many different subsystems of a design and many different levels of abstraction—one must consider the forest when designing trees and consider the trees when designing the forest.

Ov.3 Cross-Cutting Issues

Though their implementation details might apply at a local level, most design decisions must be considered in terms of their system-level effects and side-effects before they become part of the system/hierarchy. For instance, power is a cross-cutting, system-level phenomenon, even though most power optimizations are specific to certain technologies and are applied locally; reliability is a system-level issue, even though each level of the hierarchy implements its own techniques for improving it; and, as we have shown, performance optimizations such as widening a bus or increasing support for concurrency rarely result in system performance that is globally optimal. Moreover, design decisions that locally optimize along one axis (e.g., power) can have even larger effects on the system level when all axes are considered. Not only can the global power dissipation be thrown off optimality by blindly making a local decision, it is even easier to throw the system off a global optimum when more than one axis is considered (e.g., power/performance).

Designing the best system given a set of constraints requires an approach that considers multiple axes simultaneously and measures the system-level effects of all design choices. Such a holistic approach requires an understanding of many issues, including cost and performance models, power, reliability, and software structure. The following sections provide overviews of these cross-cutting issues, and Part IV of the book will treat these topics in more detail.

Ov.3.1 Cost/Performance Analysis

To perform a cost/performance analysis correctly, the designer must define the problem correctly, use the appropriate tools for analysis, and apply those tools in the manner for which they were designed. This section provides a brief, intuitive look at the problem. Herein, we will use *cost* as an example of problem definition, *Pareto optimality* as an example of an appropriate tool, and *sampled averages* as an example to illustrate correct tool usage. We will discuss these issues in more detail with more examples in Chapter 28, *Analysis of Cost and Performance.*

Problem Definition: Cost

A designer must think in an all-inclusive manner when accounting for cost. For example, consider a cost-performance analysis of a DRAM system wherein performance is measured in sustainable bandwidth and cost is measured in pin count.

To represent the cost correctly, the analysis should consider *all* pins, including those for control, power, ground, address, and data. Otherwise, the resulting analysis can incorrectly portray the design space, and workable solutions can get left out of the analysis. For example, a designer can reduce latency in some cases by increasing the number of address and command pins, but if the cost analysis only considers data pins, then these optimizations would be cost-free. Consider DRAM addressing, which is done half of an address at a time. A 32-bit physical address is sent to the DRAM system 16 bits at a time in two different commands; one could potentially decrease DRAM latency by using an SRAM-like wide address bus and sending the entire 32 bits at once. This represents a *real* cost in design and manufacturing that would be higher, but an analysis that accounts only for data pins would not consider it as such.

Power and ground pins must also be counted in a cost analysis for similar reasons. High-speed chip-to-chip interfaces typically require more power and ground pins than slower interfaces. The extra power and ground signals help to isolate the I/O drivers from each other and the signal lines

from each other, both improving signal integrity by reducing crosstalk, ground bounce, and related effects. I/O systems with higher switching speeds would have an unfair advantage over those with lower switching speeds (and thus fewer power/ground pins) in a cost-performance analysis if power and ground pins were to be excluded from the analysis. The inclusion of these pins would provide for an effective and easily quantified trade-off between cost and bandwidth.

Failure to include address, control, power, and ground pins in an analysis, meaning failure to be all-inclusive at the conceptual stages of design, would tend to blind a designer to possibilities. For example, an architecturally related family of solutions that at first glance gives up total system bandwidth so as to be more cost-effective might be thrown out at the conceptual stages for its intuitively lower performance. However, considering all sources of cost in the analysis would allow a designer to look more closely at this family and possibly to recover lost bandwidth through the addition of pins.

Comparing SDRAM and Rambus system architectures provides an excellent example of considering cost as the total number of pins leading to a continuum of designs. The Rambus memory system is a narrow-channel architecture, compared to SDRAM's wide-channel architecture, pictured in Figure Ov.7 Rambus uses fewer address and command pins than SDRAM and thus incurs an additional latency at the command level. Rambus also uses fewer data pins and occurs an additional latency when transmitting data as well. The trade-off is the ability to run the bus at a much higher bus frequency, or *pin-bandwidth* in bits per second per pin, than SDRAM. The longer channel of the DRDRAM (direct Rambus DRAM) memory system contributes directly to longer read-command latencies and longer bus turnaround times. However, the longer channel also allows for more devices to be connected to the memory system and reduces the likelihood that consecutive commands access the same device. The width and depth of the memory channels impact the bandwidth, latency, pin count, and various cost components of the respective memory systems. The effect that these organizational differences have on the DRAM access protocol is shown in Figure Ov.8 which illustrates a row activation and column read

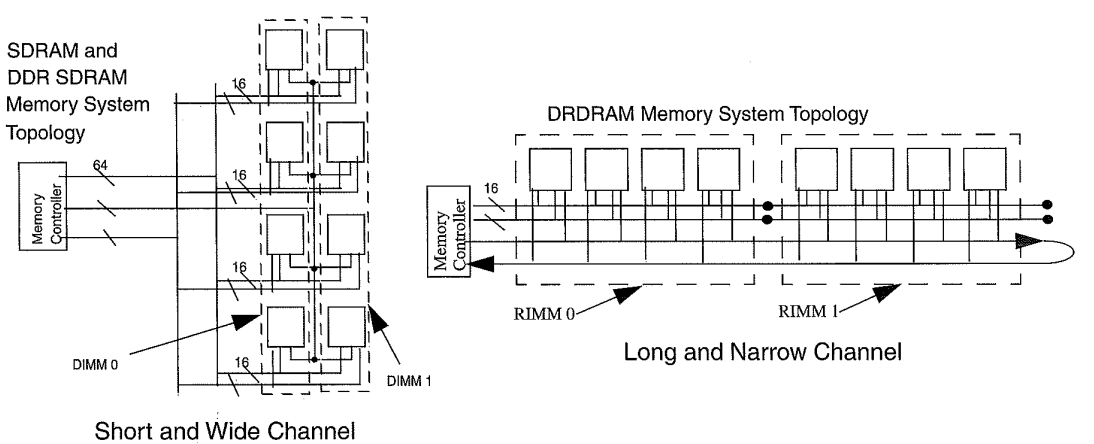


FIGURE Ov.7: Difference in topology between SDRAM and Rambus memory systems.

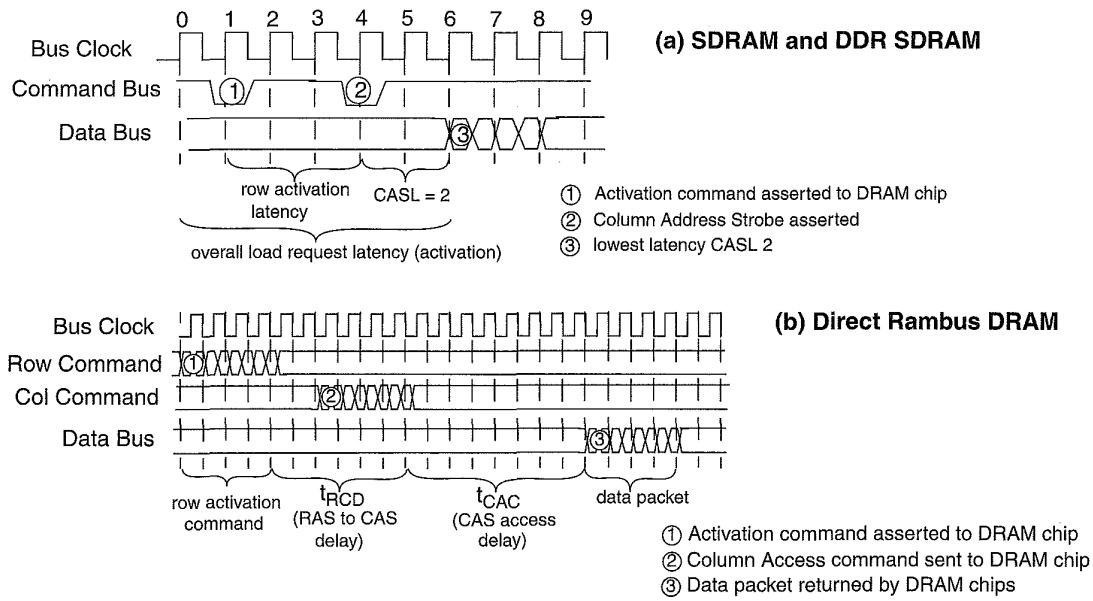


FIGURE Ov.8: Memory access latency in SDRAM and DDR SDRAM memory systems (top) and DRDRAM (bottom).

command for both DDR SDRAM and Direct Rambus DRAM.

Contemporary SDRAM and DDR SDRAM memory chips operating at a frequency of 200 MHz can activate a row in 3 clock cycles. Once the row is activated, memory controllers in SDRAM or DDR SDRAM memory systems can retrieve data using a simple column address strobe command with a latency of 2 or 3 clock cycles. In Figure Ov.8(a), Step 1 shows the assertion of a row activation command, and Step 2 shows the assertion of the column address strobe signal. Step 3 shows the relative timing of a high-performance DDR SDRAM memory module with a CASL (CAS latency) of 2 cycles. For a fair comparison against the DRDRAM memory system, we include the bus cycle that the memory controller uses to assert the load command to the memory chips. With this additional cycle included, a DDR SDRAM memory system has a read latency of 6 clock cycles (to critical data). In a SDRAM or DDR SDRAM memory system that operates at 200 MHz, 6 clock cycles translate to 30 ns of latency for a memory load command with row activation latency inclusive. These latency values are the same for high-performance SDRAM and DDR SDRAM memory systems.

The DRDRAM memory system behaves very differently from SDRAM and DDR SDRAM memory systems. Figure Ov.8(b) shows a row activation command in Step 1, followed by a column access command in Step 2. The requested data is then returned by the memory chip to the memory controller in Step 3. The row activation command in Step 1 is transmitted by the memory controller to the memory chip in a packet format that spans 4 clock cycles. The minimum delay between the row activation and column access is 7 clock cycles, and, after an additional (also minimum) CAS (column address strobe) latency of 8 clock cycles, the DRDRAM chip begins to transmit the data to the memory controller. One caveat to the computation of the access latency in the DRDRAM memory system is that CAS delay in the DRDRAM memory system is a function of the number of devices on a single DRDRAM memory channel. On a DRDRAM memory system with a full load of 32 devices

TABLE Ov.2 Peak bandwidth statistics of SDRAM, DDR SDRAM, and DRDRAM memory systems

	Operating Frequency (Data)	Data Channel Pin Count	Data Channel Bandwidth	Control Channel Pin Count	Command Channel Bandwidth	Address Channel Pin Count	Address Channel Bandwidth
SDRAM controller	133	64	1064 MB/s	28	465 MB/s	30	500 MB/s
DDR SDRAM controller	2 * 200	64	3200 MB/s	42	1050 MB/s	30	750 MB/s
DRDRAM controller	2 * 600	16	2400 MB/s	9	1350 MB/s	8	1200 MB/s
x16 SDRAM chip	133	16	256 MB/s	9	150 MB/s	15	250 MB/s
x16 DDR SDRAM chip	2 *200	16	800 MB/s	11	275 MB/s	15	375 MB/s

TABLE Ov.3 Cross-comparison of SDRAM, DDR SDRAM, and DRDRAM memory systems

DRAM Technology	Operating Frequency (Data Bus)	Pin Count per Channel	Peak Bandwidth	Sustained BW on StreamAdd	Bits per Pin per Cycle (Peak)	Bits per Pin per Cycle (Sustained)
SDRAM	133	152	1064 MB/s	540 MB/s	0.4211	0.2139
DDR SDRAM	2 * 200	171	3200 MB/s	1496 MB/s	0.3743	0.1750
DRDRAm	2 * 600	117	2400 MB/s	1499 MB/s	0.1368	0.0854

on the data bus, the CAS-latency delay may be as large as 12 clock cycles. Finally, it takes 4 clock cycles for the DRDRAM memory system to transport the data packet. Note that we add half the transmission time of the data packet in the computation of the latency of a memory request in a DRDRAM memory system due to the fact that the DRDRAM memory system does not support critical word forwarding, and the critically requested data may exist in the latter parts of the data packet; on average, it will be somewhere in the middle. This yields a total latency of 21 cycles, which, in a DRDRAM memory system operating at 600 MHz, translates to a latency of 35 ns.

The Rambus memory system trades off a longer latency for fewer pins and higher pin bandwidth (in this example, three times higher bandwidth). How do the systems compare in performance?

Peak bandwidth of any interface depends solely on the channel width and the operating frequency of the channel. In Table Ov.2, we summarize the statistics of the interconnects and compute the peak bandwidths of the memory systems at the interface of the memory controller and at the interface of the memory chips as well.

Table Ov.3 compares a 133-MHz SDRAM, a 200-MHz DDR SDRAM system, and a 600-MHz DRDRAM system. The 133-MHz SDRAM system, as represented by a PC-133 compliant SDRAM memory system on an AMD Athlon-based computer system, has a theoretical peak bandwidth of 1064 MB/s. The maximum sustained bandwidth for the single channel of SDRAM, as measured by the use of the add kernel in the STREAM benchmark, reaches 540 MB/s. The maximum sustained bandwidth for DDR SDRAM and DRDRAM was also measured on STREAM, yielding 1496 and 1499 MB/s, respectively. The pin cost of each system is factored in, yielding bandwidth per pin on both a per-cycle basis and a per-nanosecond basis.

Appropriate Tools: Pareto Optimality

It is convenient to represent the "goodness" of a design solution, a particular system configuration,

as a single number so that one can readily compare the number with the goodness ratings of other candidate design solutions and thereby quickly find the "best" system configuration. However, in the design of memory systems, we are inherently dealing with a multi-dimensional design space (e.g., one that encompasses performance, energy consumption, cost, etc.), and so using a single number to represent a solution's worth is not really appropriate, unless we can assign exact weights to the various figures of merit (which is dangerous and will be discussed in more detail later) or we care about one aspect to the exclusion of all others (e.g., performance at any cost).

Assuming that we do not have exact weights for the figures of merit and that we do care about more than one aspect of the system, a very powerful tool to aid in system analysis is the concept of *Pareto optimality* or *Pareto efficiency,* named after the Italian economist Vilfredo Pareto, who invented it in the early 1900s.

Pareto optimality asserts that one candidate solution to a problem is better than another candidate solution only if the first *dominates* the second, i.e., if the first is better than or equal to the second in *all* figures of merit. If one solution has a better value in one dimension but a worse value in another, then the two candidates are Pareto equivalent. The best solution is actually a set

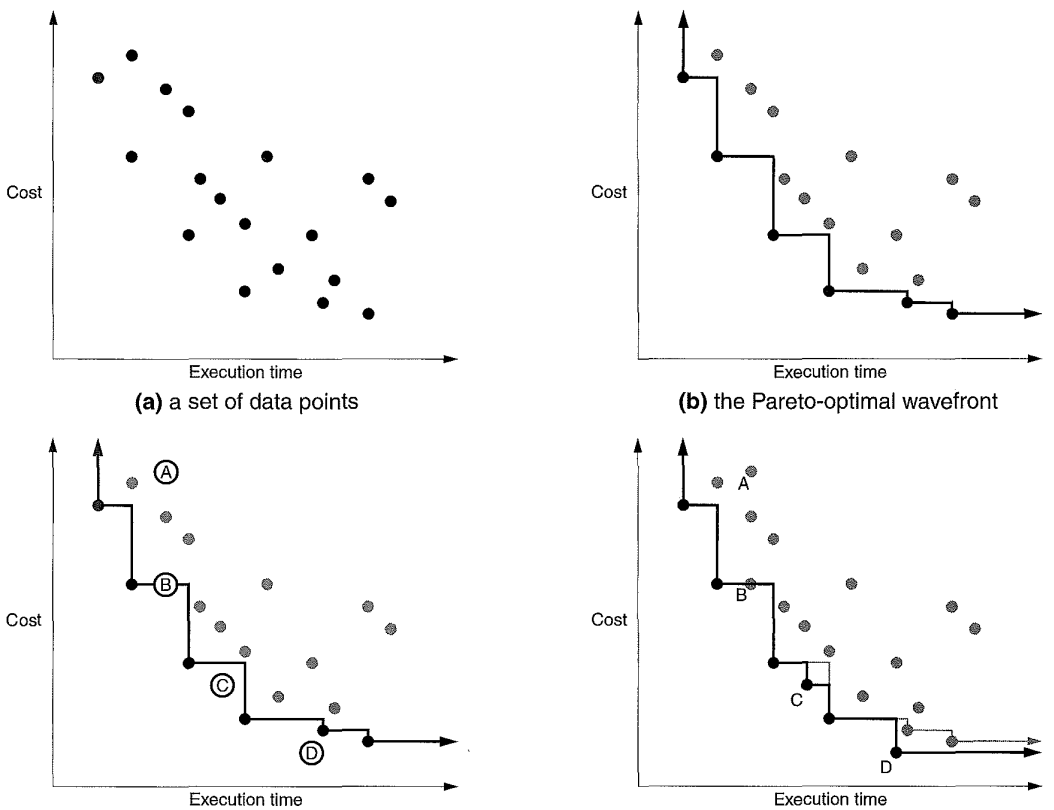


FIGURE 0v.9: Pareto optimality. Members of the Pareto-optimal set are shown in solid black; non-optimal points are grey.

of candidate solutions: the set of Pareto-equivalent solutions that is not dominated by any solution.

Figure Ov.9(a) shows a set of candidate solutions in a two-dimensional space that represent a cost/performance metric. The x-axis represents system performance in execution time (smaller numbers are better), and the y-axis represents system cost in dollars (smaller numbers are better). Figure Ov.9(b) shows the Pareto-optimal set in solid black and connected by a line; non-optimal data points are shown in grey. The Pareto-optimal set forms a wavefront that approaches both axes simultaneously. Figures Ov.9(c) and (d) show the effect of adding four new candidate solutions to the space: one lies inside the wavefront, one lies on the wavefront, and two lie outside the wavefront. The first two new additions, A and B, are both dominated by at least one member of the Pareto-optimal set, and so neither is considered Pareto optimal. Even though B lies on the wavefront, it is not considered Pareto optimal. The point to the left of B has better performance than B at equal cost. Thus, it dominates B.

Point C is not dominated by any member of the Pareto-optimal set, nor does it dominate any member of the Pareto-optimal set. Thus, candidate-solution C is added to the optimal set, and its addition changes the shape of the wavefront slightly. The last of the additional points, D, is dominated by no members of the optimal set, but it *does* dominate several members of the optimal set, so D's inclusion in the optimal set excludes those dominated members from the set. As a result, candidate-solution D changes the shape of the wave front more significantly than candidate-solution C.

Tool Use: Taking Sampled Averages Correctly

In many fields, including the field of computer engineering, it is quite popular to find a *sampled average,* i.e., the average of a sampled set of numbers, rather than the average of the entire set. This is useful when the entire set is unavailable, difficult to obtain, or expensive to obtain. For example, one might want to use this technique to keep a running performance average for a real microprocessor, or one might want to sample several windows of execution in a terabyte-size trace file. Provided that the sampled subset is representative of the set as a whole, and provided that the technique used to collect the samples is correct, this mechanism provides a low-cost alternative that can be very accurate.

The discussion will use as an example a mechanism that samples the miles-per-gallon performance of an automobile under way. The trip we will study is an out and back trip with a brief pit stop, as shown in Figure Ov.10. The automobile will follow a simple course that is easily analyzed:

1. The auto will travel over even ground for 60 miles at 60 mph, and it will achieve 30 mpg during this window of time.
2. The auto will travel uphill for 20 miles at 60 mph, and it will achieve 10 mpg during this window of time.

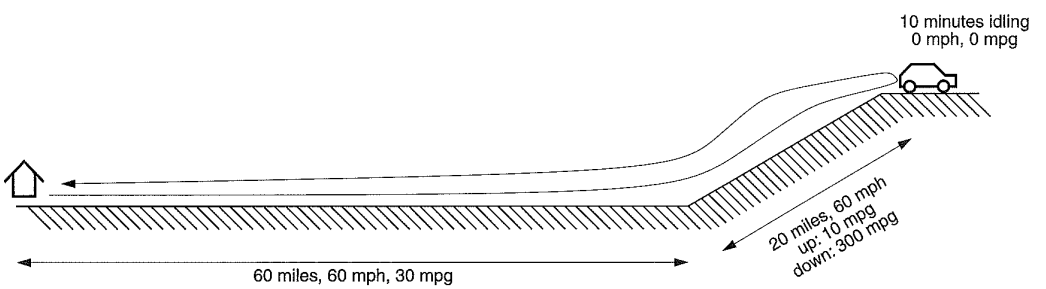


FIGURE Ov.10: Course taken by the automobile in the example.

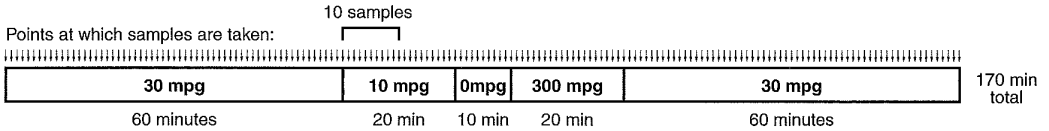


FIGURE Ov.11: Sampling miles-per-gallon (mpg) over time. The figure shows the trip in time, with each segment of time labeled with the average miles-per-gallon for the car during that segment of the trip. Thus, whenever the sampling algorithm samples miles-per-gallon during a window of time, it will add that value to the running average.

3. The auto will travel downhill for 20 miles at 60 mph, and it will achieve 300 mpg during this window of time.
4. The auto will travel back home over even ground for 60 miles at 60 mph, and it will achieve 30 mpg during this window of time.
5. In addition, before returning home, the driver will sit at the top of the hill for 10 minutes, enjoying the view, with the auto idling, consuming gasoline at the rate of 1 gallon every 5 hours. This is equivalent to 1/300 gallon per minute or 1/30 of a gallon during the 10-minute respite. Note that the auto will achieve 0 mpg during this window of time.

Our car's algorithm samples evenly in time, so for our analysis we need to break down the segments of the trip by the amount of time that they take:

- Outbound: 60 minutes
- Uphill: 20 minutes
- Idling: 10 minutes
- Downhill: 20 minutes
- Return: 60 minutes

This is displayed graphically in Figure Ov.11, in which the time for each segment is shown to scale. Assume, for the sake of simplicity, that the sampling algorithm samples the car's miles-per-gallon every minute and adds that sampled value to the running average (it could just as easily sample every second or millisecond). Then the algorithm will sample the value 30 mpg 60 times during the first segment of the trip, the value 10 mpg 20 times during the second segment of the trip, the value 0 mpg 10 times during the third segment of the trip, and so on. Over the trip, the car is operating for a total of 170 minutes. Thus, we can derive the sampling algorithm's results as follows:

$$\frac{60}{170}30 + \frac{20}{170}10 + \frac{10}{170}0 + \frac{20}{170}300 + \frac{60}{170}30 = 57.5\text{mpg} \quad \textbf{(EQ Ov.3)}$$

The sampling algorithm tells us that the auto achieved 57.5 mpg during our trip. However, a quick reality check will demonstrate that this cannot be correct; somewhere in our analysis we have made an invalid assumption. What is the correct answer, the correct approach? In Part IV of the book we will revisit this example and provide a complete picture. In the meantime, the reader is encouraged to figure the answer out for him- or herself.

Ov.3.2 Power and Energy

Power has become a "first-class" design goal in recent years within the computer architecture and design community. Previously, low-power circuit, chip, and system design was considered the purview of specialized communities, but this is no longer the case, as even high-performance chip manufacturers can be blindsided by power dissipation problems.

Power Dissipation in Computer Systems

Power dissipation in CMOS circuits arises from two different mechanisms: *static power*, which is primarily *leakage power* and is caused by the transistor not completely turning off, and *dynamic power*, which is largely the result of switching capacitive loads

between two different voltage states. Dynamic power is dependent on frequency of circuit activity, since no power is dissipated if the node values do not change, while static power is independent of the frequency of activity and exists whenever the chip is powered on. When CMOS circuits were first used, one of their main advantages was the negligible leakage current flowing with the gate at DC or steady state. Practically all of the power consumed by CMOS gates was due to dynamic power consumed during the transition of the gate. But as transistors become increasingly smaller, the CMOS leakage current starts to become significant and is projected to be larger than the dynamic power, as shown in Figure Ov.12.

In charging a load capacitor C up ΔV volts and discharging it to its original voltage, a gate pulls an amount of current equal to $C \cdot \Delta V$ from the V_{dd} supply to charge up the capacitor and then sinks this charge to ground discharging the node. At the end of a charge/discharge cycle, the gate/capacitor combination has moved $C \cdot \Delta V$ of charge from V_{dd} to ground, which uses an amount of energy equal to $C \cdot \Delta V \cdot V_{dd}$ that is independent of the cycle time. The average dynamic power of this node, the average rate of its energy consumption, is given by the following equation [Chandrakasan & Brodersen 1995]:

$$P_{dynamic} = C \cdot \Delta V \cdot V_{dd} \cdot \alpha \cdot f \qquad \textbf{(EQ Ov.4)}$$

Dividing by the charge/discharge period (i.e., multiplying by the clock frequency f) produces the rate of energy consumption over that period. Multiplying by the expected *activity ratio* α, the probability that the node will switch (in which case it dissipates dynamic power; otherwise, it does not), yields an average power dissipation over a larger window of time for which the activity ratio holds (e.g., this can yield average power for an entire hour of computation, not just a nanosecond). The dynamic power for the whole chip is the sum of this equation over all nodes in the circuit.

It is clear from EQ Ov.4 what can be done to reduce the dynamic power dissipation of a system. We can either reduce the capacitance being switched, the voltage swing, the power supply voltage, the activity ratio, or the operating frequency. Most of these options are available to a designer at the architecture level.

Note that, for a specific chip, the voltage swing ΔV is usually proportional to V_{dd}, so EQ Ov.4 is often simplified to the following:

$$P_{dynamic} = C \cdot V^2_{dd} \cdot \alpha \cdot f \qquad \textbf{(EQ Ov.5)}$$

Moreover, the activity ratio α is often approximated as 1/2, giving the following form:

$$P_{dynamic} = \frac{1}{2} \cdot C \cdot V^2_{dd} \cdot f \qquad \textbf{(EQ Ov.6)}$$

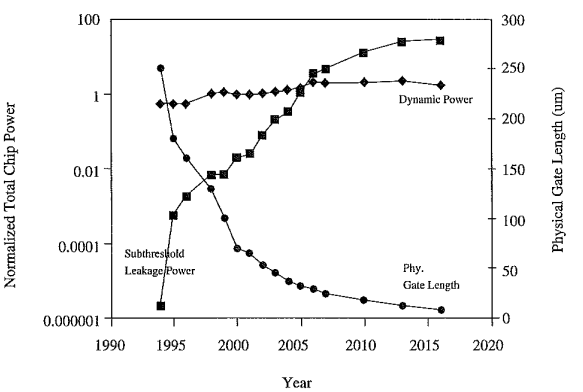


FIGURE Ov.12: Projections for dynamic and leakage, along with gate length. (Figure taken from Kim et al. [2004a]).

Static leakage power is due to our inability to completely turn off the transistor, which leaks current in the subthreshold operating region [Taur & Ning 1998]. The gate couples to the active channel mainly through the gate oxide capacitance, but there are other capacitances in a transistor that couple the gate to a "fixed charge" (charge which cannot move) present in the bulk and not associated with current flow [Peckerar et al. 1979, 1982]. If these extra capacitances are large (note that they increase with each process generation as physical dimensions shrink), then changing the gate bias merely alters the densities of the fixed charge and will not turn the channel off. In this situation, the transistor becomes a leaky faucet; it does not turn off no matter how hard you turn it.

Leakage power is proportional to V_{dd}. It is a linear, not a quadratic, relationship. For a particular process technology, the per-device leakage power is given as follows [Butts & Sohi 2000]:

$$P_{static} = I_{leakage} \cdot V^2_{dd} \qquad \textbf{(EQ Ov.7)}$$

Leakage energy is the product of leakage power times the duration of operation.

It is clear from EQ Ov.7 what can be done to reduce the leakage power dissipation of a system: reduce leakage current and/or reduce the power supply voltage. Both options are available to a designer at the architecture level.

Heat in VLSI circuits is becoming a significant and related problem. The rate at which physical dimensions such as gate length and gate oxide thickness have been reduced is faster than for other parameters, especially voltage, resulting in higher power densities on the chip surface. To lower leakage power and maintain device operation, voltage levels are set according to the silicon bandgap and intrinsic built-in potentials, in spite of the conventional scaling algorithm. Thus, power densities are increasing exponentially for next-generation chips. For instance, the power density of Intel's Pentium chip line has already surpassed that of a hot plate with the introduction of the Pentium Pro [Gelsinger 2001]. The problem of power and heat dissipation now extends to the DRAM system, which traditionally has represented low power densities and low costs. Today, higher end DRAMs are dynamically throttled when, due to repeated high-speed access to the same devices, their operating temperatures surpass design thresholds. The next-generation memory system embraced by the DRAM community, the Fully Buffered DIMM architecture, specifies a per-module controller that, in many implementations, requires a heatsink. This is a cost previously unthinkable in DRAM-system design.

Disks have many components that dissipate power, including the spindle motor driving the platters, the actuator that positions the disk heads, the bus interface circuitry, and the microcontroller/s and memory chips. The spindle motor dissipates the bulk of the power, with the entire disk assembly typically dissipating power in the tens of watts.

Schemes for Reducing Power and Energy

There are numerous mechanisms in the literature that attack the power dissipation and/or energy consumption problem. Here, we will briefly describe three: dynamic voltage scaling, the powering down of unused blocks, and circuit-level approaches for reducing leakage power.

Dynamic Voltage Scaling Recall that total energy is the sum of switching energy and leakage energy, which, to a first approximation, is equal to the following:

$$E_{tot} = [(C_{tot} \cdot V^2_{dd} \cdot \alpha \cdot f) + (N_{tot} \cdot I_{leakage} \cdot V_{dd})] \cdot T \qquad \textbf{(EQ Ov.8)}$$

T is the time required for the computation, and N_{tot} is the total number of devices leaking current. Variations in processor utilization affect the amount of switching activity (the activity ratio α). However, a light workload produces an idle processor that wastes clock cycles and energy because the clock signal continues propagating and the operating voltage remains the same. Gating the clock during idle cycles reduces the switched capacitance C_{tot} during idle cycles. Reducing the frequency f during

periods of low workload eliminates most idle cycles altogether.

None of the approaches, however, affects $C_{tot}V^2_{dd}$ for the actual computation or substantially reduces the energy lost to leakage current. Instead, reducing the supply voltage V_{dd} in conjunction with the frequency f achieves savings in switching energy and reduces leakage energy. For high-speed digital CMOS, a reduction in supply voltage increases the circuit delay as shown by the following equation [Baker et al. 1998, Baker 2005]:

$$T_d = \frac{C_L V_{dd}}{\mu C_{ox}(W/L)(V_{dd} - V_t)^2} \qquad \textbf{(EQ Ov.9)}$$

where

- T_d is the delay or the reciprocal of the frequency f
- V_{dd} is the supply voltage
- C_L is the total node capacitance
- μ is the carrier mobility
- C_{ox} is the oxide capacitance
- V_t is the threshold voltage
- W/L is the width-to-length ratio of the transistors in the circuit

This can be simplified to the following form, which gives the maximum operating frequency as a function of supply and threshold voltages:

$$f_{MAX} \sim \frac{(V_{dd} - V_t)^2}{V_{dd}} \qquad \textbf{(EQ Ov.10)}$$

As mentioned earlier, the threshold voltage is closely tied to the problem of leakage power, so it cannot be arbitrarily lowered. Thus, the right-hand side of the relation ends up being a constant proportion of the operating voltage for a given process technology. Microprocessors typically operate at the maximum speed at which their operating voltage level will allow, so there is not much headroom to arbitrarily lower V_{dd} by itself. However, V_{dd} can be lowered if the clock frequency is also lowered in the same proportion. This mechanism is called *dynamic voltage scaling (DVS)* [Pering & Broderson 1998] and is appearing in nearly every modern microprocessor. The technique sets the microprocessor's frequency to the most appropriate level for performing each task at hand, thus avoiding hurry-up-and-wait scenarios that consume more energy than is required for the computation (see Figure Ov.13). As Weiser points out,

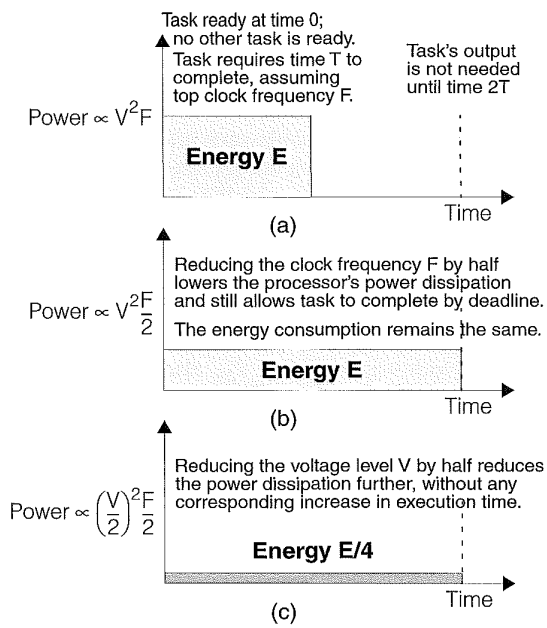


FIGURE Ov.13: Dynamic voltage scaling. Not every task needs the CPU's full computational power. In many cases, for example, the processing of video and audio streams, the only performance requirement is that the task meet a deadline, see (a). Such cases create opportunities to run the CPU at a lower performance level and achieve the same perceived performance while consuming less energy. As (b) shows, reducing the clock frequency of a processor reduces power dissipation but simply spreads a computation out over time, thereby consuming the same total energy as before. As (c) shows,reducing the voltage level as well as the clock frequency achieves the desired goal of reduced energy consumption and appropriate performance level. Figure and caption from Varma et al. [2003].

idle time represents wasted energy, even if the CPU is stopped [Weiser et al. 1994].

Note that it is not sufficient to merely have a chip that *supports* voltage scaling. There must be a heuristic, either implemented in hardware or software, that decides when to scale the voltage and by how much to scale it. This decision is essentially a prediction of the near-future computational needs of the system and is generally made on the basis of the recent computing requirements of all tasks and threads running at the time. The development of good heuristics is a tricky problem (pointed out by Weiser et al. [1994]). Heuristics that closely track performance requirements save little energy, while those that save the most energy tend to do so at the expense of performance, resulting in poor response time, for example.

Most research quantifies the effect that DVS has on reducing dynamic power dissipation because dynamic power follows V_{dd} in a quadratic relationship: reducing V_{dd} can significantly reduce dynamic power. However, lowering V_{dd} also reduces leakage power, which is becoming just as significant as dynamic power. Though the reduction is only linear, it is nonetheless a reduction.

Note also that even though DVS is commonly applied to microprocessors, it is perfectly well suited to the memory system as well. As a processor's speed is decreased through application of DVS, it requires less speed out of its associated SRAM caches, whose power supply can be scaled to keep pace. This will reduce both the dynamic and the static power dissipation of the memory circuits.

Powering-Down Unused Blocks A popular mechanism for reducing power is simply to turn off functional blocks that are not needed. This is done at both the circuit level and the chip or I/O-device level.

At the circuit level, the technique is called *clock gating*. The clock signal to a functional block (e.g., an adder, multiplier, or predictor) passes through a gate, and whenever a control circuit determines that the functional block will be unused for several cycles, the gate halts the clock signal and sends a non-oscillating voltage level to the functional block instead. The latches in the functional block retain their information; do not change their outputs; and, because the data is held constant to the combinational logic in the circuit, do not switch. Therefore, it does not draw current or consume energy.

Note that, in the naïve implementation, the circuits in this instance are still powered up, so they still dissipate static power; clock gating is a technique that only reduces dynamic power. Other gating techniques can reduce leakage as well. For example, in caches, unused blocks can be powered down using Gated-V_{dd} [Powell et al. 2000] or Gated-ground [Powell et al. 2000] techniques. Gated-V_{dd} puts the power supply of the SRAM in a series with a transistor as shown in Figure Ov.14. With the stacking effect introduced by this transistor, the leakage current is reduced drastically. This technique benefits from having both low-leakage current and a simpler fabrication process requirement since only a single threshold voltage is conceptually required (although, as shown in Figure Ov.14, the gating transistor can also have a high threshold to decrease the leakage even further at the expense of process complexity).

At the device level, for instance in DRAM chips or disk assemblies, the mechanism puts the device into a low-activity, low-voltage, and/or low-frequency mode such as *sleep* or *doze* or, in the case of disks, *spin-down*. For example, microprocessors can dissipate anywhere from a fraction of a watt to over 100 W of power; when not in use, they can be put into a low-power sleep or doze mode that consumes milli-watts. The processor typically expects an interrupt to cause it to resume normal operation, for instance, a clock interrupt, the interrupt output of a watchdog timer, or an external device interrupt. DRAM chips typically consume on the order of 1 W each; they have a low-power mode that will reduce this by more than an order of magnitude. Disks typically dissipate power in the tens of watts, the bulk of which is in the spindle motor. When the disk is placed in the "spin-down" mode (i.e., it is not rotating, but it is still responding to the disk controller),

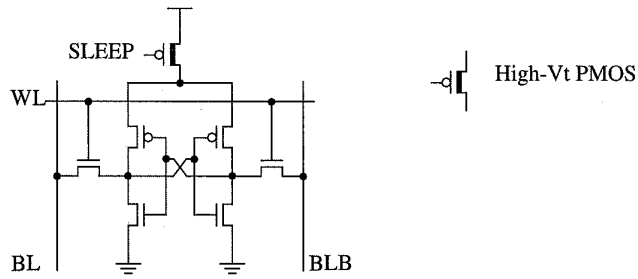


FIGURE Ov.14: Gated-V_{dd} technique using a high-V_t transistor to gate V_{dd}.

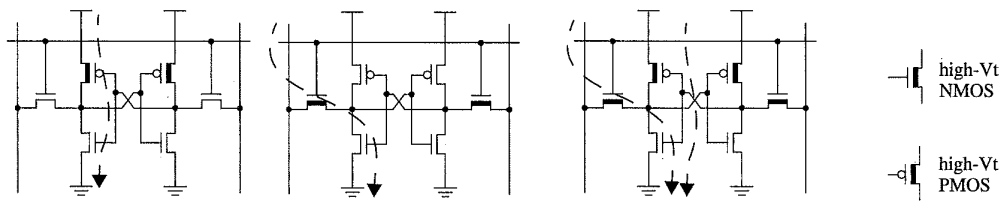


FIGURE Ov.15: Different multi-V_t configurations for the 6T memory cell showing which leakage currents are reduced for each configuration.

the disk assembly consumes a total of a handful of watts [Gurumurthi et al. 2003].

Leakage Power in SRAMs Low-power SRAM techniques provide good examples of approaches for lowering leakage power. SRAM designs targeted for low power have begun to account for the increasingly larger amount of power consumed by leakage currents.

One conceptually simple solution is the use of multi-threshold CMOS circuits. This involves using process-level techniques to increase the threshold voltage of transistors to reduce the leakage current. Increasing this threshold serves to reduce the gate overdrive and reduces the gate's drive strength, resulting in increased delay. Because of this, the technique is mostly used on the non-critical paths of the logic, and fast, low-V_t transistors are used for the critical paths. In this way the delay penalty involved in using higher V_t transistors can be hidden in the non-critical paths, while reducing the leakage currents drastically. For example, multi-V_t transistors are selectively used for memory cells since they represent a majority of the circuit, reaping the most benefit in leakage power consumption with a minor penalty in the access time. Different multi-V_t configurations are shown in Figure Ov.15, along with the leakage current path that each configuration is designed to minimize.

Another technique that reduces leakage power in SRAMs is the Drowsy technique [Kim et al. 2004a]. This is similar to gated-V_{dd} and gated-ground techniques in that it uses a transistor to conditionally enable the power supply to a given part of the SRAM. The difference is that this technique puts infrequently accessed parts of the SRAM into a

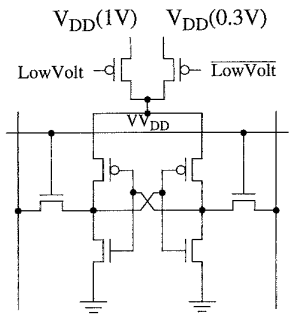


FIGURE Ov.16: A drowsy SRAM cell containing the transistors that gate the desired power supply.

state-preserving, low-power mode. A second power supply with a lower voltage than the regular supply provides power to memory cells in the "drowsy" mode. Leakage power is effectively reduced because of its dependence on the value of the power supply. An SRAM cell of a drowsy cache is shown in Figure Ov.16.

Ov.3.3 Reliability

Like performance, reliability means many things to many people. For example, embedded systems are computer systems, typically small, that run dedicated software and are embedded within the context of a larger system. They are increasingly appearing in the place of traditional electromechanical systems, whose function they are replacing because one can now find chip-level computer systems which can be programmed to perform virtually any function at a price of pennies per system. The reliability problem stems from the fact that the embedded system is a state machine (piece of software) executing within the context of a relatively complex state machine (real-time operating system) executing within the context of an extremely complex state machine (microprocessor and its memory system). We are replacing simple electromechanical systems with ultra-complex systems whose correct function cannot be guaranteed. This presents an enormous problem for the future, in which systems will only get more complex and will be used increasingly in safety-critical situations, where incorrect functioning can cause great harm.

This is a very deep problem, and one that is not likely to be solved soon. A smaller problem that we *can* solve right now—one that engineers currently do—is to increase the reliability of data within the memory system. If a datum is stored in the memory system, whether in a cache, in a DRAM, or on disk, it is reasonable to expect that the next time a processor reads that datum, the processor will get the value that was written.

How could the datum's value change? Solid-state memory devices (e.g., SRAMs and DRAMs) are susceptible to both hard failures and soft errors in the same manner that other semiconductor-based electronic devices are susceptible to both hard failures and soft failures. Hard failures can be caused by electromigration, corrosion, thermal cycling, or electrostatic shock. In contrast to hard failures, soft errors are failures where the physical device remains functional, but random and transient electronic noises corrupt the value of the stored information in the memory system. Transient noise and upset comes from a multitude of sources, including circuit noise (e.g., crosstalk, ground bounce, etc.), ambient radiation (e.g., even from sources within the computer chassis), clock jitter, or substrate interactions with high-energy particles. Which of these is the most common is obviously very dependent on the operating environment.

Figure Ov.17 illustrates the last of these examples. It pictures the interactions between high-energy alpha particles and neutrons with the silicon lattice. The figure shows that when high-energy alpha particles pass through silicon, the alpha particle leaves an ionized trail, and the length of that ionized trail depends on the energy of the alpha particle. The figure also illustrates that when high-energy neutrons pass through silicon, some neutrons pass through without affecting operations of the semiconductor device, but some neutrons collide with nuclei in the silicon lattice. The atomic collision can result in the creation of multiple ionized trails as the secondary particles generated in the collision scatter in the silicon lattice. In the presence of an electric field, the ionized trails of

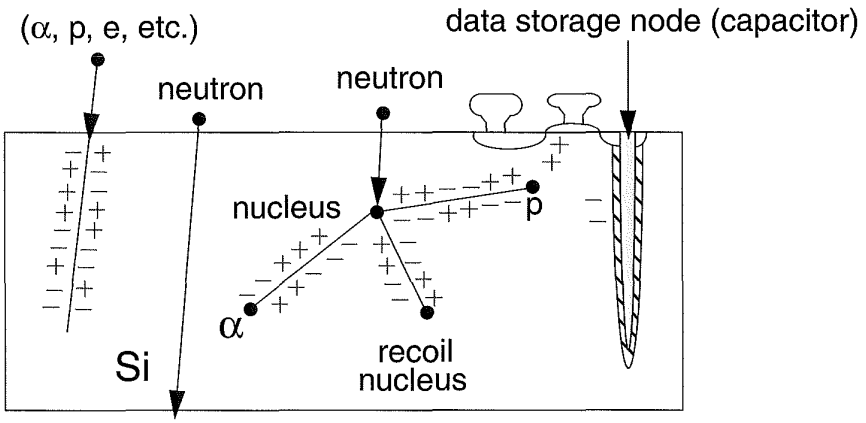


FIGURE Ov.17: Generation of electron-hole pairs in silicon by alpha particles and high-energy neutrons.

TABLE Ov.4 Cross-comparison of failure rates for SRAM, DRAM, and disk

Technology	Failure Rate[a] (SRAM & DRAM: at 0.13 μm)	Frequency of Multi-bit Errors (Relative to Single-bit Errors)	Expected Service Life
SRAM	100 per million device-hours		Several years
DRAM	1 per million device-hours	10–20%	Several years
Disk	1 per million device-hours		Several years

[a]Note that failure rate, i.e., a variation of mean-time-between-failures, says nothing about the expected performance of a single device. However, taken with the expected service life of a device, it can give a designer or administrator an idea of expected performance. If the service life of a device is 5 years, then the part will last about 5 years. A very large installation of those devices (e.g., in the case of disks or DRAMs, hundreds or more) will collectively see the expected failure rate: i.e., several hundred disks will collectively see several million device hours of operation before a single disk fails.

electron-hole pairs behave as temporary surges in current or as charges that can change the data values in storage cells. In addition, charge from the ionized trails of electron-hole pairs can impact the voltage level of bit lines as the value of the stored data is resolved by the sense amplifiers. The result is that the *soft error rate (SER)* of a memory-storage device depends on a combination of factors including the type, number, and energy distribution of the incident particles as well as the process technology design of the storage cells, design of the bit lines and sense amplifiers, voltage level of the device, as well as the design of the logic circuits that control the movement of data in the DRAM device.

Table Ov.4 compares the failure rates for SRAM, DRAM, and disk. SRAM device error rates have historically tracked DRAM devices and did so up until the 180-nm process generation. The combination of reduced supply voltage and reduced critical cell charge means that SRAM SERs have climbed dramatically for the 180-nm and 130-nm process generations. In a recent publication, Monolithic System

Technology, Inc. (MoSys) claimed that for the 250-nm process generation, SRAM SERs were reported to be in the range of 100 failures per million device-hours per megabit, while SERs were reported to be in the range of 100,000 failures per megabit for the 130-nm process generation. The generalized trend is expected to continue to increase as the demand for low power dissipation forces a continued reduction in supply voltage and reduced critical charge per cell.

Solid-state memory devices (SRAMs and DRAMs) are typically protected by error detection codes and/or ECC. These are mechanisms wherein data redundancy is used to detect and/or recover from single- and even multi-bit errors. For instance, parity is a simple scheme that adds a bit to a protected word, indicating the number of even or odd bits in the word. If the read value of the word does not match the parity value, then the processor knows that the read value does not equal the value that was initially written, and an error has occurred. Error correction is achieved by encoding a word such that a bit error moves the resulting word some distance away from the original word (in the Hamming-distance sense) into an invalid encoding. The encoding space is chosen such that the new, invalid word is closest in the space to the original, valid word. Thus, the original word can always be derived from an invalid codeword, assuming a maximum number of bit errors.

Due to SRAM's extreme sensitivity to soft errors, modern processors now ship with parity and single-bit error correction for the SRAM caches. Typically, the tag arrays are protected by parity, whereas the data arrays are protected by single-bit error correction. More sophisticated multi-bit ECC algorithms are typically not deployed for on-chip SRAM caches in modern processors since the addition of sophisticated computation circuitry can add to the die size and cause significant delay relative to the timing demands of the on-chip caches. Moreover, caches store frequently accessed data, and in case an uncorrectable error is detected, a processor simply has to re-fetch the data from memory. In this sense, it can be considered unnecessary to detect and correct multi-bit errors, but sufficient to simply detect multi-bit errors. However, in the physical design of modern SRAMs, often designers will intentionally place capacitors above the SRAM cell to improve SER.

Disk reliability is a more-researched area than data reliability in disks, because data stored in magnetic disks tends to be more resistant to transient errors than data stored in solid-state memories. In other words, whereas reliability in solid-state memories is largely concerned with correcting soft errors, reliability in hard disks is concerned with the fact that disks occasionally die, taking most or all of their data with them. Given that the disk drive performs the function of permanent store, its reliability is paramount, and, as Table Ov.4 shows, disks tend to last several years. This data is corroborated by a recent study from researchers at Google [Pinheiro et al. 2007]. The study tracks the behavior and environmental parameters of a fleet of over 100,000 disks for five years.

Reliability in the disk system is improved in much the same manner as ECC: data stored in the disk system is done so in a redundant fashion. RAID (redundant array of inexpensive disks) is a technique wherein encoded data is striped across multiple disks, so that even in the case of a disk's total failure the data will always be available.

Ov.3.4 Virtual Memory

Virtual memory is the mechanism by which the operating system provides executing software access to the memory system. In this regard, it is the primary consumer of the memory system: its procedures, data structures, and protocols dictate how the components of the memory system are used by all software that runs on the computer. It therefore behooves the reader to know what the virtual memory system does and how it does it. This section provides a brief overview of the mechanics of virtual memory. More detailed treatments of the topic can also be found on-line in articles by the author [Jacob & Mudge 1998a–c].

In general, programs today are written to run on no particular hardware configuration. They have no knowledge of the underlying memory system. Processes execute in imaginary address spaces that

are mapped onto the memory system (including the DRAM system and disk system) by the operating system. Processes generate instruction fetches and loads and stores using imaginary or "virtual" names for their instructions and data. The ultimate home for the process's address space is nonvolatile *permanent store*, usually a disk drive; this is where the process's instructions and data come from and where all of its permanent changes go to. Every hardware memory structure between the CPU and the permanent store is a cache for the instructions and data in the process's address space. This includes main memory—main memory is really nothing more than a cache for a process's virtual address space. A cache operates on the principle that a small, fast storage device can hold the most important data found on a larger, slower storage device, effectively making the slower device look fast. The large storage area in this case is the process address space, which can range from kilobytes to gigabytes or more in size. Everything in the address space initially comes from the program file stored on disk or is created on demand and defined to be zero. This is illustrated in Figure Ov.18.

Address Translation

Translating addresses from virtual space to physical space is depicted in Figure Ov.19. Addresses are mapped at the granularity of *pages*. Virtual memory is

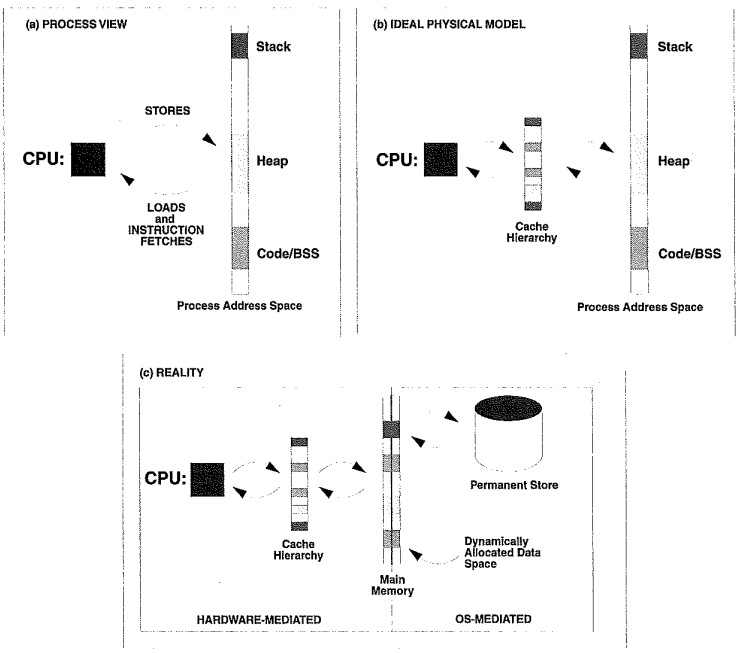


FIGURE Ov.18: Caching the process address space. In the first view, a process is shown referencing locations in its address space. Note that all loads, stores, and fetches use virtual names for objects. The second view illustrates that a process references locations in its address space indirectly through a hierarchy of caches. The third view shows that the address space is not a linear object stored on some device, but is instead scattered across hard drives and dynamically allocated when necessary.

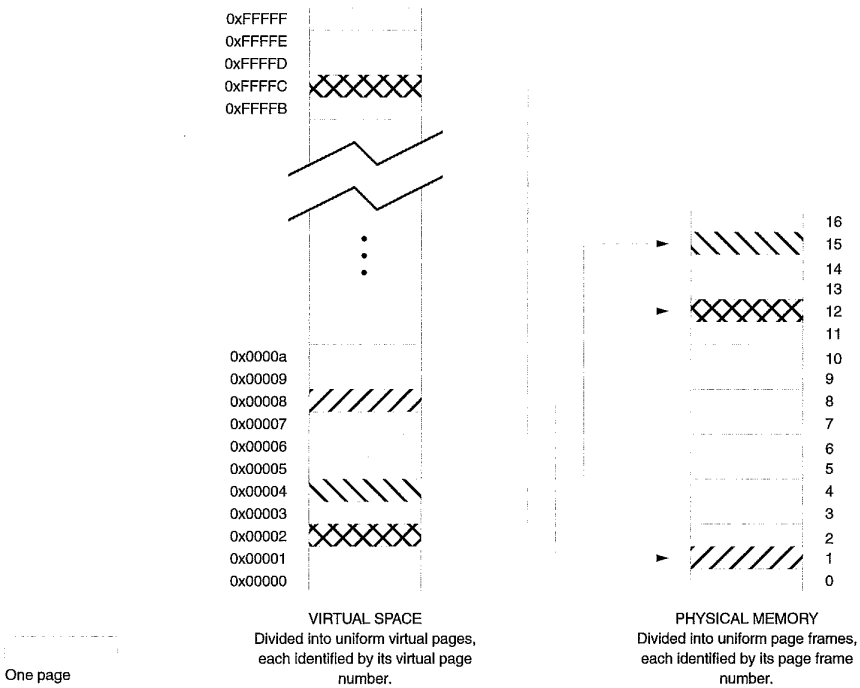


FIGURE Ov.19: Mapping virtual pages into physical page frames.

essentially a mapping of *virtual page numbers* (*VPNs*) to *page frame numbers* (*PFNs*). The mapping is a function, and any virtual page can have only one location. However, the inverse map is not necessarily a function. It is possible and sometimes advantageous to have several virtual pages mapped to the same page frame (to share memory between processes or threads or to allow different views of data with different protections, for example). This is depicted in Figure Ov.19 by mapping two virtual pages (0x00002 and 0xFFFFC) to PFN 12.

If DRAM is a cache, what is its organization? For example, an idealized *fully associative* cache (one in which any datum can reside at any location within the cache's data array) is pictured in Figure Ov.20. A data tag is fed into the cache. The first stage compares the input tag to the tag of every piece of data in the cache. The matching tag points to the data's location in the cache. However, DRAM is not physically built like a cache. For example, it has no inherent concept of a tags array: one merely tells memory what data location one wishes to read or write, and the datum at that location is read out or overwritten. There is no attempt to match the address against a tag to verify the contents of the data location. However, if main memory is to be an effective cache for the virtual address space, the tags mechanism must be implemented *somewhere*. There is clearly a myriad of possibilities, from special DRAM designs that include a hardware tag feature to software algorithms that make several memory references to look up one datum. Traditional virtual memory has the tags array implemented in software, and this software structure often holds more entries than there are entries in the data array (i.e., pages in main memory). The software

structure is called a *page table;* it is a database of mapping information.

The page table performs the function of the tags array depicted in Figure Ov.20. For any given memory reference, it indicates where in main memory (corresponding to "data array" in the figure) that page can be found. There are many different possible organizations for page tables, most of which require only a few memory references to find the appropriate tag entry. However, requiring more than one memory reference for a page table lookup can be very costly, and so access to the page table is sped up by caching its entries in a special cache called the *translation lookaside buffer* (*TLB*) [Lee 1960], a hardware structure that typically has far fewer entries than there are pages in main memory. The TLB is a hardware cache which is usually implemented as a content addressable memory (CAM), also called a fully associative cache.

The TLB takes as input a VPN, possibly extended by an address-space identifier, and returns the corresponding PFN and protection information. This is illustrated in Figure Ov.21. The address-space identifier, if used, extends the virtual address to distinguish it from similar virtual addresses produced by other processes. For a load or store to complete successfully, the

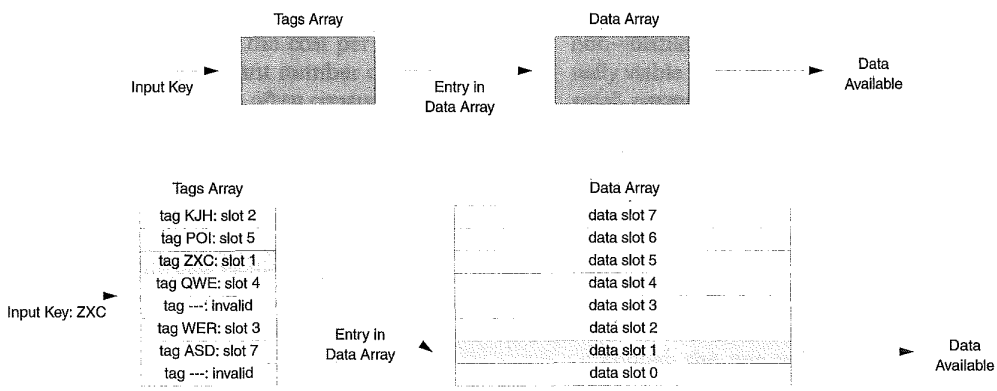


FIGURE Ov.20: An idealized cache lookup. A cache is comprised of two parts: the tag's array and the data array. In the example organization, the tags act as a database. They accept as input a key (an address) and output either the location of the item in the data array or an indication that the item is not in the data array.

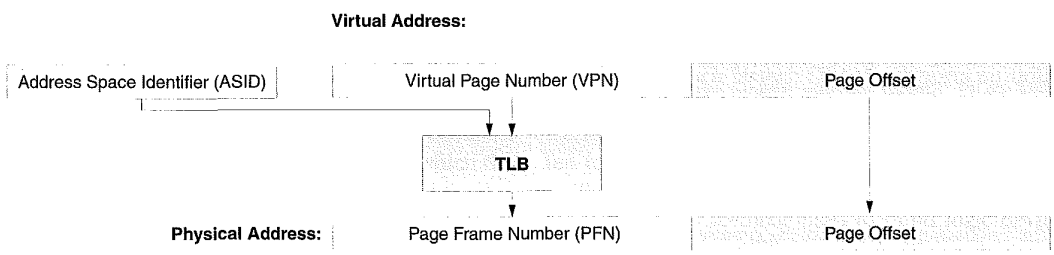


FIGURE Ov.21: Virtual-to-physical address translation using a TLB.

TLB must contain the mapping information for that virtual location. If it does not, a *TLB miss* occurs, and the system[9] must search the page table for the appropriate entry and place it into the TLB. If the system fails to find the mapping information in the page table, or if it finds the mapping but it indicates that the desired page is on disk, a *page fault* occurs. A page fault interrupts the OS, which must then retrieve the page from disk and place it into memory, create a new page if the page does not yet exist (as when a process allocates a new stack frame in virgin territory), or send the process an error signal if the access is to illegal space.

Shared Memory

Shared memory is a feature supported by virtual memory that causes many problems and gives rise to cache-management issues. It is a mechanism whereby two address spaces that are normally protected from each other are allowed to intersect at points, still retaining protection over the non-intersecting regions. Several processes sharing portions of their address spaces are pictured in Figure Ov.22. The shared memory mechanism only opens up a pre-defined portion of a process's address space; the rest of the address space is still protected, and even the shared portion is only unprotected for those processes sharing the memory. For instance, in Figure Ov.22, the region of A's address space that is shared with process B is unprotected from whatever actions B might want to take, but it is safe from the actions of any other processes. It is therefore useful as a simple, secure means for inter-process communication. Shared memory also reduces requirements for physical memory, as when the text regions of processes are shared whenever multiple instances of a single program are run or when multiple instances of a common library are used in different programs.

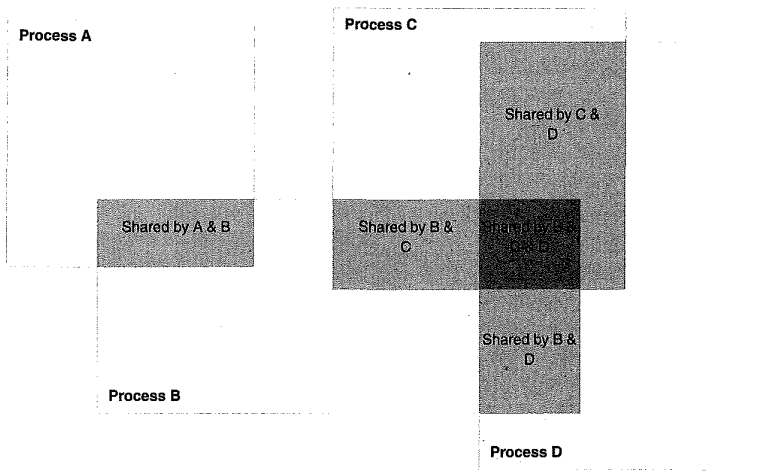


FIGURE Ov.22: Shared memory. Shared memory allows processes to overlap portions of their address space while retaining protection for the nonintersecting regions. This is a simple and effective method for inter-process communication. Pictured are four process address spaces that have overlapped. The darker regions are shared by more than one process, while the lightest regions are still protected from other processes.

[9]In the discussions, we will use the generic term "system" when the acting agent is implementation-dependent and can refer to either a hardware state machine or the operating system. For example, in some implementations, the page table search immediately following a TLB miss is performed by the operating system (MIPS, Alpha); in other implementations, it is performed by the hardware (PowerPC, x86).

The mechanism works by ensuring that shared pages map to the same physical page. This can be done by simply placing the same PFN in the page tables of two processes sharing a page. An example is shown in Figure Ov.23. Here, two very small address spaces are shown overlapping at several places, and one address space overlaps with itself; two of its virtual pages map to the same physical page. This is not just a contrived example. Many operating systems allow this, and it is useful, for example, in the implementation of user-level threads.

Some Commercial Examples

A few examples of what has been done in industry can help to illustrate some of the issues involved.

MIPS Page Table Design MIPS [Heinrich 1995, Kane & Heinrich 1992] eliminated the page table-walking hardware found in traditional memory management units and, in doing so, demonstrated that software can table-walk with reasonable efficiency. It also presented a simple hierarchical page table design, shown in Figure Ov.24. On a TLB miss, the VPN of the

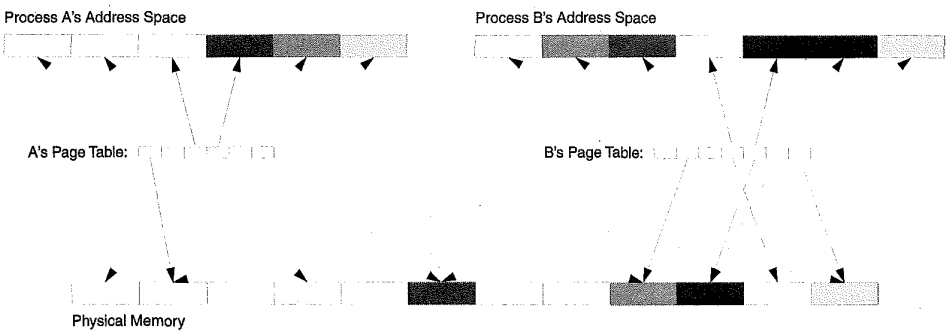


FIGURE Ov.23: An example of shared memory. Two process address spaces—one comprised of six virtual pages and the other of seven virtual pages—are shown sharing several pages. Their page tables maintain information on where virtual pages are located in physical memory. The darkened pages are mapped to several locations; note that the darkest page is mapped at two locations in the same address space.

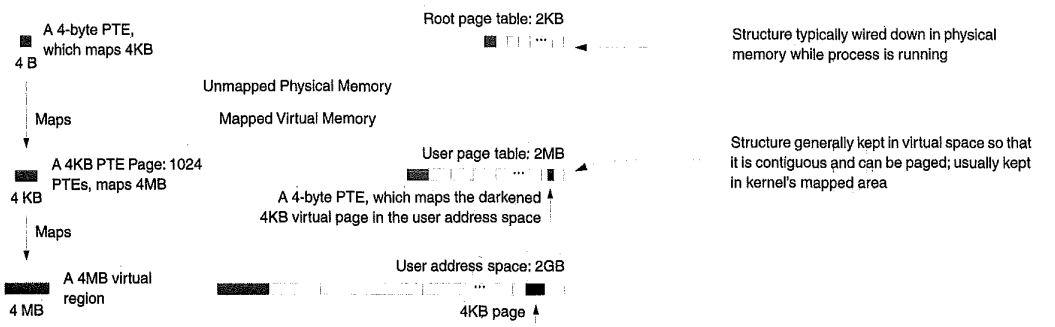


FIGURE Ov.24: The MIPS 32-bit hierarchical page table. MIPS hardware provides support for a 2-MB linear virtual page table that maps the 2-GB user address space by constructing a virtual address from a faulting virtual address that indexes the mapping PTE (page-table entry) in the user page table. This 2-MB page table can easily be mapped by a 2-KB user root page table.

address that missed the TLB is used as an index into the user page table, which is accessed using a virtual address. The architecture provides hardware support for this activity, storing the virtual address of the base of the user-level page table in a hardware register and forming the concatenation of the base address with the VPN. This is illustrated in Figure Ov.25. On a TLB miss, the hardware creates a virtual address for the mapping PTE in the user page table, which must be aligned on a 2-MB virtual boundary for the hardware's lookup address to work. The base pointer, called *PTEBase*, is stored in a hardware register and is usually changed on context switch.

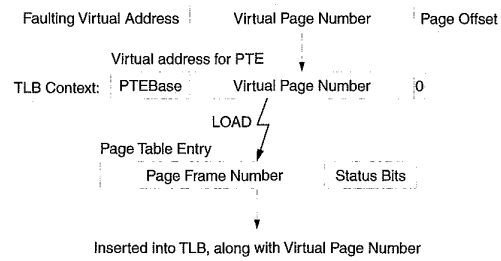


FIGURE Ov.25: The use of the MIPS TLB context register. The VPN of the faulting virtual address is placed into the context register, creating the virtual address of the mapping PTE. This PTE goes directly into the TLB.

PowerPC Segmented Translation The IBM 801 introduced a segmented design that persisted through the POWER and PowerPC architectures [Chang & Mergen 1988, IBM & Motorola 1993, May et al. 1994, Weiss & Smith 1994]. It is illustrated in Figure Ov.26. Applications generate 32-bit "effective" addresses that are mapped onto a larger "virtual" address space at the granularity of *segments*, 256-MB virtual regions. Sixteen segments comprise an application's address space. The top four bits of the effective address select a segment identifier from a set of 16 registers. This segment ID is concatenated with the bottom 28 bits of the effective address to form an extended virtual address. This extended address is used in the TLB and page table. The operating system performs data movement and relocation at the granularity of pages, not segments.

The architecture does not use explicit address-space identifiers; the segment registers ensure address space protection. If two processes duplicate an identifier in their segment registers, they share that virtual segment by definition. Similarly, protection is guaranteed if identifiers are *not* duplicated. If memory is shared through global addresses, the TLB and cache need not be flushed on context switch[10] because the system behaves like a single address space operating system. For more details, see Chapter 31, Section 31.1.7, *Perspective: Segmented Addressing Solves the Synonym Problem*.

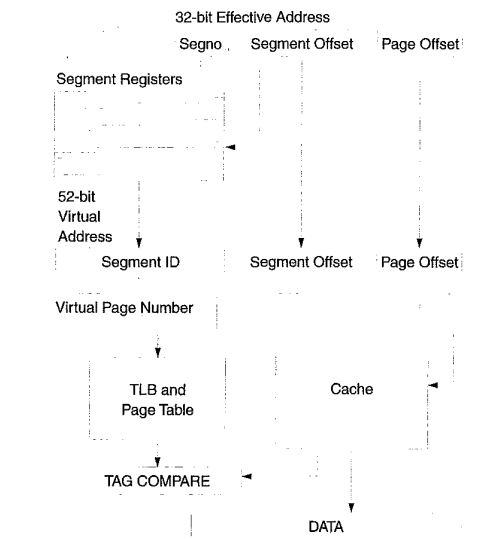


FIGURE Ov.26: PowerPC segmented address translation. Processes generate 32-bit effective addresses that are mapped onto a 52-bit address space via 16 segment registers, using the top 4 bits of the effective address as an index. It is this extended virtual address that is mapped by the TLB and page table. The segments provide address space protection and can be used for shared memory.

[10]Flushing is avoided until the system runs out of identifiers and must reuse them. For example, the address-space identifiers on the MIPS R3000 and Alpha 21064 are six bits wide, with a maximum of 64 active processes [Digital 1994, Kane & Heinrich 1992]. If more processes are desired, identifiers must be constantly reassigned, requiring TLB and virtual-cache flushes.

Ov.4 An Example Holistic Analysis

Disk I/O accounts for a substantial fraction of an application's execution time and power dissipation. A new DRAM technology called *Fully Buffered DIMM* (*FB-DIMM*) has been in development in the industry [Vogt 2004a, b, Haas & Vogt 2005], and, though it provides storage scalability significantly beyond the current DDRx architecture, FB-DIMM has met with some resistance due to its high power dissipation. Our modeling results show that the energy consumed in a moderate-size FB-DIMM system is indeed quite large, and it can easily approach the energy consumed by a disk.

This analysis looks at a trade-off between storage in the DRAM system and in the disk system, focusing on the disk-side write buffer; if configured and managed correctly, the write buffer enables a system to approach the performance of a large DRAM installation at half the energy. Disk-side caches and write buffers have been proposed and studied, but their effect upon total system behavior has not been studied. We present the impact on total system execution time, CPI, and memory-system power, including the effects of the operating system. Using a full-system, execution-based simulator that combines Bochs, Wattch, CACTI, DRAMsim, and DiskSim and boots the RedHat Linux 6.0 kernel, we have investigated the memory-system behavior of the SPEC CPU2000 applications. We study the disk-side cache in both single-disk and RAID-5 organizations. Cache parameters include size, organization, whether the cache supports write caching or not, and whether it prefetches read blocks or not. Our results are given in terms of L1/L2 cache accesses, power dissipation, and energy consumption; DRAM-system accesses, power dissipation, and energy consumption; disk-system accesses, power dissipation, and energy consumption; and execution time of the application plus operating system, in seconds. The results are not from sampling, but rather from a simulator that calculates these values on a cycle-by-cycle basis over the entire execution of the application.

Ov.4.1 Fully-Buffered DIMM vs. the Disk Cache

It is common knowledge that disk I/O is expensive in both power dissipated and time spent waiting on it. What is less well known is the system-wide breakdown of disk power versus cache power versus DRAM power, especially in light of the newest DRAM architecture adopted by industry, the FB-DIMM. This new DRAM standard replaces the conventional memory bus with a narrow, high-speed interface between the memory controller and the DIMMs. It has been shown to provide performance similar to that of DDRx systems, and thus, it represents a relatively low-overhead mechanism (in terms of execution time) for scaling DRAM-system capacity. FB-DIMM's latency degradation is not severe. It provides a noticeable bandwidth improvement, and it is relatively insensitive to scheduling policies [Ganesh et al. 2007].

FB-DIMM was designed to solve the problem of storage scalability in the DRAM system, and it provides scalability well beyond the current JEDEC-style DDRx architecture, which supports at most two to four DIMMs in a fully populated dual-channel system (DDR2 supports up to two DIMMs per channel; proposals for DDR3 include limiting a channel to a single DIMM). The daisy-chained architecture of FB-DIMM supports up to eight DIMMs per channel, and its narrow bus requires roughly one-third the pins of a DDRx SDRAM system. Thus, an FB-DIMM system supports an order of magnitude more DIMMs than DDRx. This scalability comes at a cost, however. The DIMM itself dissipates almost an order of magnitude more power than a traditional DDRx DIMM. Couple this with an order-of-magnitude increase in DIMMs per system, and one faces a serious problem.

To give an idea of the problem, Figure Ov.27 shows the simulation results of an entire execution of the *gzip* benchmark from SPEC CPU2000 on a complete-system simulator. The memory system is only moderate in size: one channel and four DIMMs, totalling a half-gigabyte. The graphs demonstrate numerous important issues, but in this book we are concerned with two items in particular:

- Program initialization is lengthy and represents a significant portion of an application's run time. As the CPI graph shows, the first two-thirds of execution time are spent dealing with the disk, and the corresponding CPI (both average and instantaneous) ranges from the 100s to the 1000s. After this initialization phase, the application settles into a

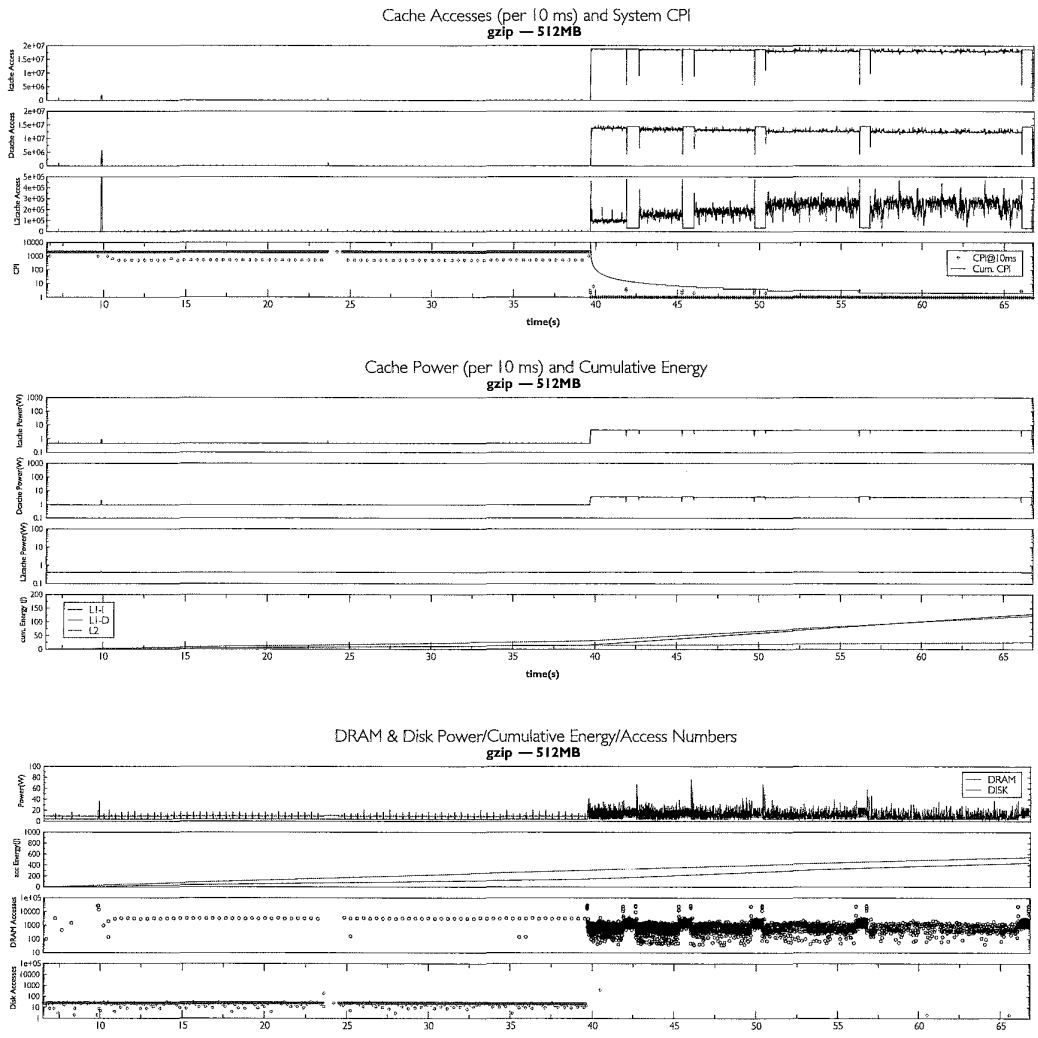


FIGURE 0v.27: Full execution of Gzip. The figure shows the entire run of gzip. System configuration is a 2-GHz Pentium processor with 512 MB of DDR2-533 FB-DIMM main memory and a 12k-RPM disk drive with built-in disk cache. The figure shows the interaction between all components of the memory system, including the L1 instruction and data caches, the unified L2 cache, the DRAM system, and the disk drive. All graphs use the same x-axis, which represents execution time in seconds. The x-axis does not start at zero; the measurements exclude system boot time, invocation of the shell, etc. Each data point represents aggregated (not sampled) activity within a 10-ms epoch. The CPI graph shows two system CPI values: one is the average CPI for each 10-ms epoch, and the other is the cumulative average CPI. A duration with no CPI data point indicates that no instructions were executed due to I/O latency. During such a window the CPI is essentially infinite, and thus, it is possible for the cumulative average to range higher than the displayed instantaneous CPI. Note that the CPI, the DRAM accesses, and the disk accesses are plotted on log scales.

more compute-intensive phase in which the CPI asymptotes down to the theoretical sustainable performance, the single-digit values that architecture research typically reports.

- By the end of execution, the total energy consumed in the FB-DIMM DRAM system (a half a kilojoule) almost equals that of the energy consumed by the disk, and it is twice that of the L1 data cache, L1 instruction cache, and unified L2 cache combined.

Currently, there is substantial work happening in both industry and academia to address the latter issue, with much of the work focusing on access scheduling, architecture improvements, and data migration. To complement this work, we look at a wide range of organizational approaches, i.e., attacking the problem from a parameter point of view rather than a system-redesign, component-redesign, or new-proposed-mechanism point of view, and find significant synergy between the disk cache and the memory system. Choices in the disk-side cache affect both system-level performance and system-level (in particular, DRAM-subsystem-level) energy consumption. Though disk-side caches have been proposed and studied, their effect upon the total system behavior, namely execution time or CPI or total memory-system power including the effects of the operating system, is as yet unreported. For example, Zhu and Hu [2002] evaluate disk built-in cache using both real and synthetic workloads and report the results in terms of average response time. Smith [1985a and b] evaluates a disk cache mechanism with real traces collected in real IBM mainframes on a disk cache simulator and reports the results in terms of miss rate. Huh and Chang [2003] evaluate their RAID controller cache organization with a synthetic trace. Varma and Jacobson [1998] and Solworth and Orji [1990] evaluate destaging algorithms and write caches, respectively, with synthetic workloads. This study represents the first time that the effects of the disk-side cache can be viewed at a system level (considering both application and operating-system effects) and compared directly to all the other components of the memory system.

We use a full-system, execution-based simulator combining Bochs [Bochs 2006], Wattch [Brooks et al. 2000], CACTI [Wilton & Jouppi 1994], DRAMsim [Wang et al. 2005, September], and DiskSim [Ganger et al. 2006]. It boots the RedHat Linux 6.0 kernel and therefore can capture all application behavior, and all operating-system behavior, including I/O activity, disk-block buffering, system-call overhead, and virtual memory overhead such as translation, table walking, and page swapping. We investigate the disk-side cache in both single-disk and RAID-5 organizations. Cache parameters include size, organization, whether the cache supports write caching or not, and whether it prefetches read blocks or not. Additional parameters include disk rotational speed and DRAM-system capacity.

We find a complex trade-off between the disk cache, the DRAM system, and disk parameters like rotational speed. The disk cache, particularly its write-buffering feature, represents a very powerful tool enabling significant savings in both energy and execution time. This is important because, though the cache's support for write buffering is often enabled in desktop operating systems (e.g., Windows and some but not all flavors of Unix/Linux [Ng 2006]), it is typically disabled in enterprise computing applications [Ng 2006], and these are the applications most likely to use FB-DIMMs [Haas & Vogt 2005]. We find substantial improvement between existing implementations and an ideal write buffer (i.e., this is a limit study). In particular, the disk cache's write-buffering ability can offset the total energy consumption of the memory system (including caches, DRAMs, and disks) by nearly a factor of two, while sacrificing a small amount of performance.

Ov.4.2 Fully Buffered DIMM: Basics

The relation between a traditional organization and a FB-DIMM organization is shown in Figure Ov.28, which motivates the design in terms of a graphics-card organization. The first two drawings show a multi-drop DRAM bus next to a DRAM bus organization typical of graphics cards, which use point-to-point soldered connections between the DRAM and memory controller to achieve higher speeds. This arrangement is used in FB-DIMM.

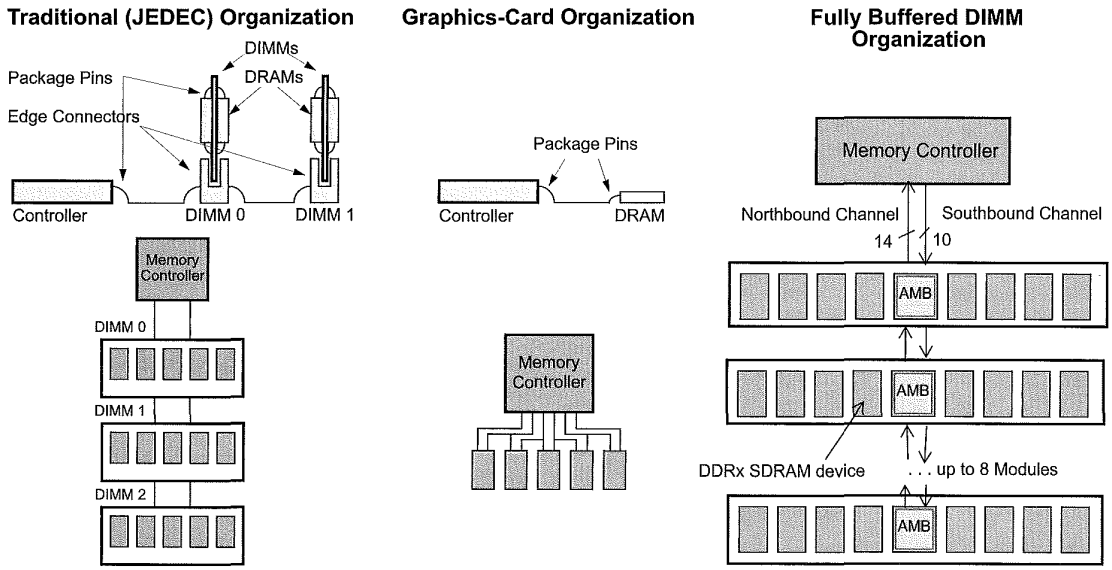


FIGURE Ov.28: FB-DIMM and its motivation. The first two pictures compare the memory organizations of a JEDEC SDRAM system and a graphics card. Above each design is its side-profile, indicating potential impedance mismatches (sources of reflections). The organization on the far right shows how the FB-DIMM takes the graphics-card organization as its *de facto* DIMM. In the FB-DIMM organization, there are no multi-drop busses; DIMM-to-DIMM connections are point to point. The memory controller is connected to the nearest AMB via two unidirectional links. The AMB is, in turn, connected to its southern neighbor via the same two links.

A slave memory controller has been added onto each DIMM, and all connections in the system are point to point. A narrow, high-speed channel connects the master memory controller to the DIMM-level memory controllers (called *Advanced Memory Buffers* or AMBs). Since each DIMM-to-DIMM connection is a point-to-point connection, a channel becomes a *de facto* multi-hop store and forward network. The FB-DIMM architecture limits the channel length to eight DIMMs, and the narrower inter-module bus requires roughly one-third as many pins as a traditional organization. As a result, an FB-DIMM organization can handle roughly 24 times the storage capacity of a single-DIMM DDR3-based system, without sacrificing any bandwidth and even leaving headroom for increased intra-module bandwidth.

The AMB acts like a pass-through switch, directly forwarding the requests it receives from the controller to successive DIMMs and forwarding frames from southerly DIMMs to northerly DIMMs or the memory controller. All frames are processed to determine whether the data and commands are for the local DIMM. The FB-DIMM system uses a serial packet-based protocol to communicate between the memory controller and the DIMMs. Frames may contain data and/or commands. Commands include DRAM commands such as row activate (RAS), column read (CAS), refresh (REF) and so on, as well as channel commands such as write to configuration registers, synchronization commands, etc. Frame scheduling is performed exclusively by the memory controller. The AMB only converts the serial protocol to DDRx-based commands without implementing any scheduling functionality.

The AMB is connected to the memory controller and/or adjacent DIMMs via unidirectional links: the southbound channel which transmits both data

and commands and the northbound channel which transmits data and status information. The southbound and northbound datapaths are 10 bits and 14 bits wide, respectively. The FB-DIMM channel clock operates at six times the speed of the DIMM clock; i.e., the link speed is 4 Gbps for a 667-Mbps DDRx system. Frames on the north- and southbound channel require 12 transfers (6 FB-DIMM channel clock cycles) for transmission. This 6:1 ratio ensures that the FB-DIMM frame rate matches the DRAM command clock rate.

Southbound frames comprise both data and commands and are 120 bits long; northbound frames are data only and are 168 bits long. In addition to the data and command information, the frames also carry header information and a frame CRC (cyclic redundancy check) checksum that is used to check for transmission errors. A northbound read-data frame transports 18 bytes of data in 6 FB-DIMM clocks or 1 DIMM clock. A DDRx system can burst back the same amount of data to the memory controller in two successive beats lasting an entire DRAM clock cycle. Thus, the read bandwidth of an FB-DIMM system is the same as that of a single channel of a DDRx system. Due to the narrower southbound channel, the write bandwidth in FB-DIMM systems is one-half that available in a DDRx system. However, this makes the *total* bandwidth available in an FB-DIMM system 1.5 times that of a DDRx system.

Figure Ov.29 shows the processing of a read transaction in an FB-DIMM system. Initially, a command frame is used to transmit a command that will perform row activation. The AMB translates the request and relays it to the DIMM. The memory controller schedules the CAS command in a following frame. The AMB relays the CAS command to the DRAM devices which burst the data back to the AMB. The AMB bundles two consecutive bursts of data into a single northbound frame and transmits it to the memory controller. In this example, we assume a burst length of four corresponding to two FB-DIMM data frames. Note that although the figures do not identify parameters like t_CAS, t_RCD, and t_CWD, the memory controller must ensure that these constraints are met.

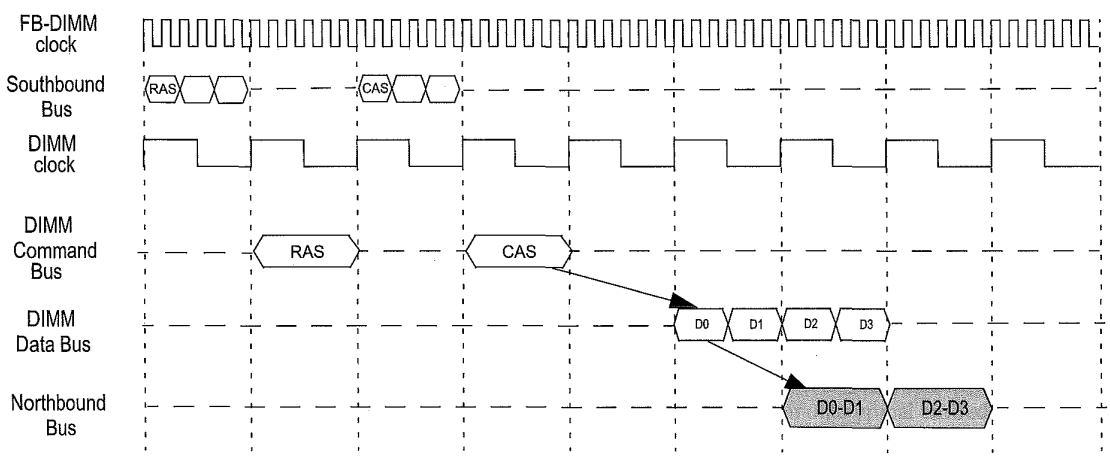


FIGURE Ov.29: Read transaction in an FB-DIMM system. The figure shows how a read transaction is performed in an FB-DIMM system. The FB-DIMM serial busses are clocked at six times the DIMM busses. Each FB-DIMM frame on the southbound bus takes six FB-DIMM clock periods to transmit. On the northbound bus a frame comprises two DDRx data bursts.

The primary dissipater of power in an FB-DIMM channel is the AMB, and its power depends on its position within the channel. The AMB nearest to the memory controller must handle its own traffic and repeat all packets to and from all downstream AMBs, and this dissipates the most power. The AMB in DDR2-533 FB-DIMM dissipates 6 W, and it is currently 10 W for 800 Mbps DDR2 [Staktek 2006]. Even if one averages out the activity on the AMB in a long channel, the eight AMBs in a single 800-Mbps channel can easily dissipate 50 W. Note that this number is for the AMBs only; it does not include power dissipated by the DRAM devices.

Ov.4.3 Disk Caches: Basics

Today's disk drives all come with a built-in cache as part of the drive controller electronics, ranging in size from 512 KB for the micro-drive to 16 MB for the largest server drives. Figure Ov.30 shows the cache and its place within a system. The earliest drives had no cache memory, as they had little control electronics. As the control of data transfer migrated from the host-side control logic to the drive's own controller, a small amount of memory was needed to act as a speed-matching buffer, because the disk's media data rate is different from that of the interface. Buffering is also needed because when the head is at a position ready to do data transfer, the host or the interface may be busy and not ready to receive read data. DRAM is usually used as this buffer memory.

In a system, the host typically has some memory dedicated for caching disk data, and if a drive is attached to the host via some external controller, that controller also typically has a cache. Both the system cache and the external cache are much larger than the disk drive's internal cache. Hence, for most workloads, the drive's cache is not likely to see too many reuse cache hits. However, the disk-side cache is very effective in opportunistically prefetching data, as only the controller inside the drive knows the state the drive is in and when and how it can prefetch without adding any cost in time. Finally, the drive needs cache memory if it is to support write caching/buffering.

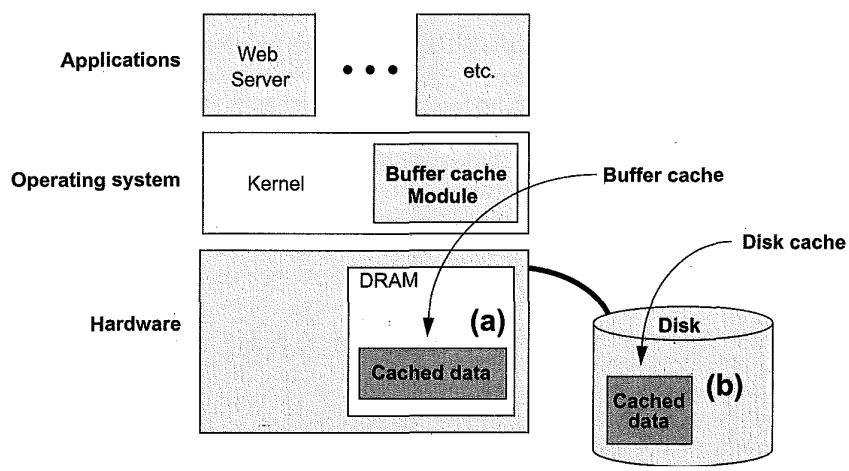


FIGURE Ov.30: Buffer caches and disk caches. Disk blocks are cached in several places, including (a) the operating system's *buffer cache* in main memory and (b), on the disk, in another DRAM buffer, called a *disk cache*.

With write caching, the drive controller services a write request by transferring the write data from the host to the drive's cache memory and then reports back to the host that the write is "done," even though the data has not yet been written to the disk media (data not yet written out to disk is referred to as *dirty*). Thus, the service time for a cached write is about the same as that for a read cache hit, involving only some drive controller overhead and electronic data transfer time but no mechanical time. Clearly, write caching does not need to depend on having the right content in the cache memory for it to work, unlike read caching. Write caching will always work, i.e., a write command will always be a cache hit, as long as there is available space in the cache memory. When the cache becomes full, some or all of the dirty data are written out to the disk media to free up space. This process is commonly referred to as *destage*.

Ideally, destage should be done while the drive is idle so that it does not affect the servicing of read requests. However, this may not be always possible. The drive may be operating in a high-usage system with little idle time ever, or the writes often arrive in bursts which quickly fill up the limited memory space of the cache. When destage must take place while the drive is busy, such activity adds to the load of drive at that time, and a user will notice a longer response time for his requests. Instead of providing the full benefit of cache hits, write caching in this case merely delays the disk writes.

Zhu and Hu [2002] have suggested that large disk built-in caches will not significantly benefit the overall system performance because all modern operating systems already use large file system caches to cache reads and writes. As suggested by Przybylski [1990], the reference stream missing a first-level cache and being handled by a second-level cache tends to exhibit relatively low locality. In a real system, the reference stream to the disk system has missed the operating system's buffer cache, and the locality in the stream tends to be low. Thus, our simulation captures all of this activity. In our experiments, we investigate the disk cache, including the full effects of the operating system's file-system caching.

Ov.4.4 Experimental Results

Figure Ov.27 showed the execution of the GZIP benchmark with a moderate-sized FB-DIMM DRAM system: half a gigabyte of storage. At 512 MB, there is no page swapping for this application. When the storage size is cut in half to 256 MB, page swapping begins but does not affect the execution time significatly. When the storage size is cut to one-quarter of its original size (128 MB), the page swapping is significant enough to slow the application down by an order of magnitude. This represents the hard type of decision that a memory-systems designer would have to face: if one can reduce power dissipation by cutting the amount of storage and feel negligible impact on performance, then one has too much storage to begin with.

Figure Ov.31 shows the behavior of the system when storage is cut to 128 MB. Note that all aspects of system behavior have degraded; execution time is longer, *and* the system consumes more energy. Though the DRAM system's energy has decreased from 440 J to just under 410 J, the execution time has increased from 67 to 170 seconds, the total cache energy has increased from 275 to 450 J, the disk energy has increased from 540 to 1635 J, and the total energy has doubled from 1260 to 2515 J. This is the result of swapping activity—not enough to bring the system to its knees, but enough to be relatively painful.

We noticed that there exists in the disk subsystem the same sort of activity observed in a microprocessor's load/store queue: reads are often stalled waiting for writes to finish, despite the fact that the disk has a 4-MB read/write cache on board. The disk's cache is typically organized to prioritize prefetch activity over write activity because this tends to give the best performance results and because the write buffering is often disabled by the operating system. The solution to the write-stall problem in microprocessors has been to use write buffers; we therefore modified DiskSim to implement an ideal write buffer on the disk side that would not interfere with the disk cache. Figure Ov.32 indicates that the size of the cache seems to make little difference to the behavior of the system. The important thing is that a cache is present. Thus, we should not expect read performance to suddenly increase as a result of moving writes into a separate write buffer.

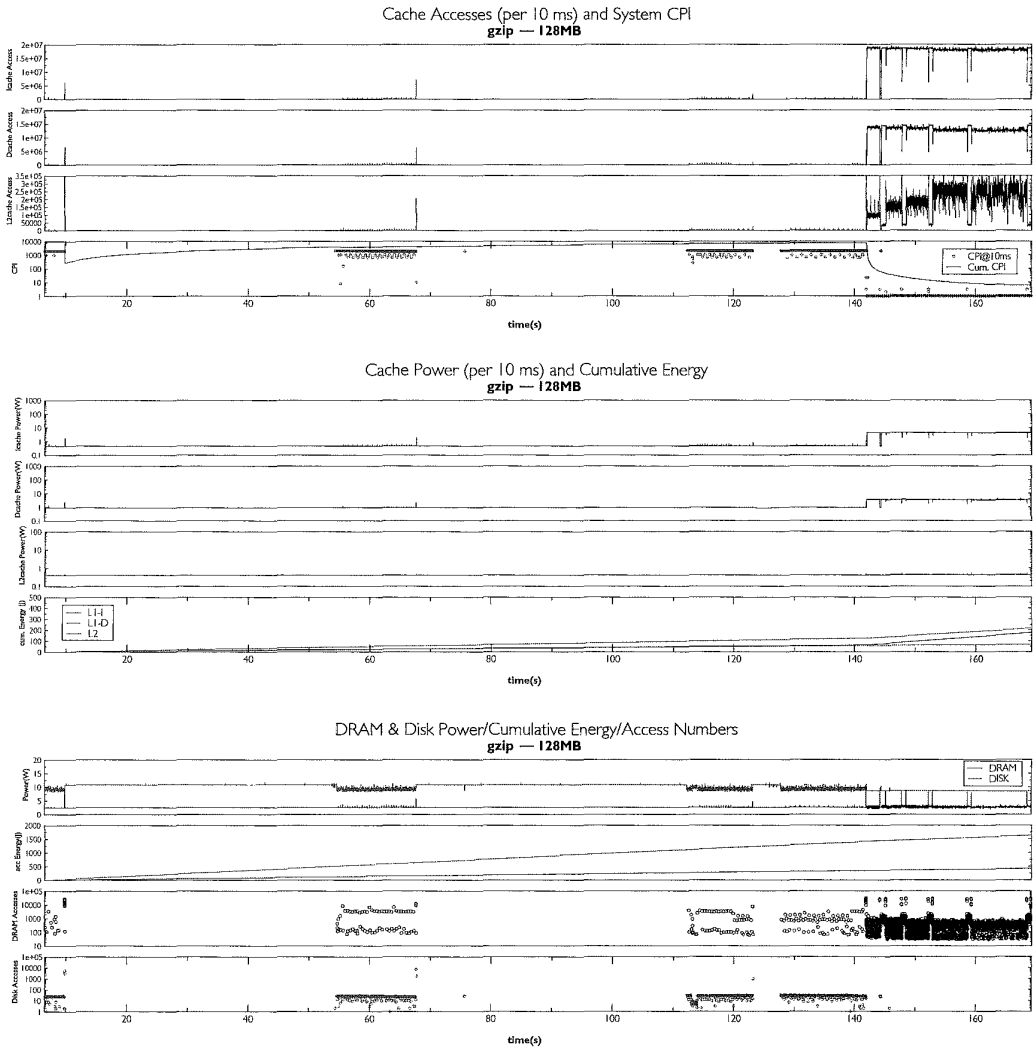


FIGURE 0v.31: Full execution of GZIP, 128 MB DRAM. The figure shows the entire run of GZIP. System configuration is a 2 GHz Pentium processor with 128 MB of FB-DIMM main memory and a 12 K-RPM disk drive with built-in disk cache. The figure shows the interaction between all components of the memory system, including the L1 instruction cache, the L1 data cache, the unified L2 cache, the DRAM system, and the disk drive. All graphs use the same *x*-axis, which represents the execution time in seconds. The *x*-axis does not start at zero; the measurements exclude system boot time, invocation of the shell, etc. Each data point represents aggregated (not sampled) activity within a 10-ms epoch. The CPI graph shows 2 system CPI values: one is the average CPI for each 10-ms epoch, the other is the cumulative average CPI. A duration with no CPI data point indicates that no instructions were executed due to I/O latency. The application is run in single-user mode, as is common for SPEC measurements; therefore, disk delay shows up as stall time. Note that the CPI, the DRAM accesses, and the Disk accesses are plotted on log scales.

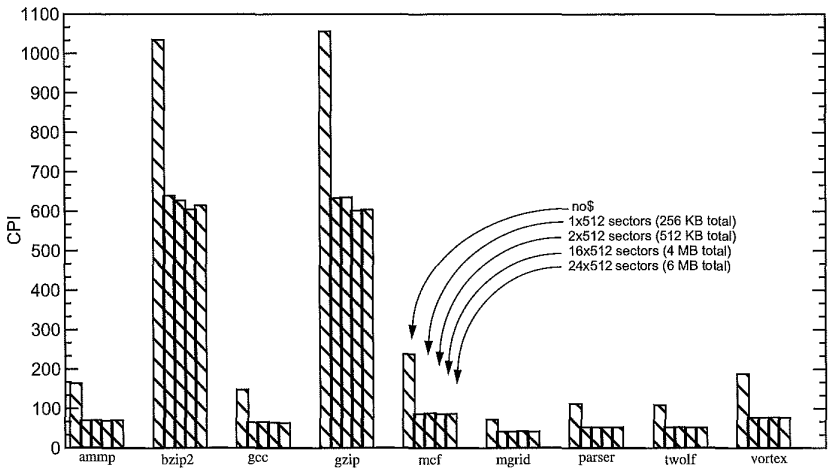


FIGURE Ov.32: The effects of disk cache size by varying the number of segments. The figure shows the effects of a different number of segments with the same segment size in the disk cache. The system configuration is 128 MB of DDR SDRAM with a 12k-RPM disk. There are five bars for each benchmark, which are (1) no cache, (2) 1 segment of 512 sectors each, (3) 2 segments of 512 sectors each, (4) 16 segment of 512 sectors each, and (5) 24 segment of 512 sectors each. Note that the CPI values are for the disk-intensive portion of application execution, not the CPU-intensive portion of application execution (which could otherwise blur distinctions).

Figure Ov.33 shows the behavior of the system with 128 MB and an ideal write buffer. As mentioned, the performance increase and energy decrease is due to the writes being buffered, allowing read requests to progress. Execution time is 75 seconds (compared to 67 seconds for a 512 MB system); and total energy is 1100 J (compared to 1260 J for a 512-MB system). For comparison, to show the effect of faster read and write throughput, Figure Ov.34 shows the behavior of the system with 128 MB and an 8-disk RAID-5 system. Execution time is 115 seconds, and energy consumption is 8.5 KJ. This achieves part of the performance effect as write buffering by improving write time, thereby freeing up read bandwidth sooner. However, the benefit comes at a significant cost in energy.

Table Ov.5 gives breakdowns for **gzip** in tabular form, and the graphs beneath the table give the breakdowns for **gzip**, **bzip2**, and **ammp** in graphical form and for a wider range of parameters (different disk RPMs). The applications all demonstrate the same trends: to cut down the energy of a 512-MB system by reducing the memory to 128 MB which causes both the performance and the energy to get worse. Performance degrades by a factor of 5–10; energy increases by 1.5× to 10×. Ideal write buffering can give the best of both worlds (performance of a large memory system and energy consumption of a small memory system), and its benefit is independent of the disk's RPM. Using a RAID system does not gain significant performance improvement, but it consumes energy proportionally to the number of disks. Note, however, that this is a uniprocessor model running in single-user mode, so RAID is not expected to shine.

Figure Ov.35 shows the effects of disk caching and prefetching on both single-disk and RAID systems. In RAID systems, disk caching has only marginal effects to both the CPI and the disk average response time. However, disk caching with prefetching has significant benefits. In a slow disk system (i.e., 5400 RPM), RAID has more tangible benefits over a non-RAID system. Nevertheless, the combination of using RAID, disk cache, and fast disks can improve the overall performance up to a factor of 10. For the

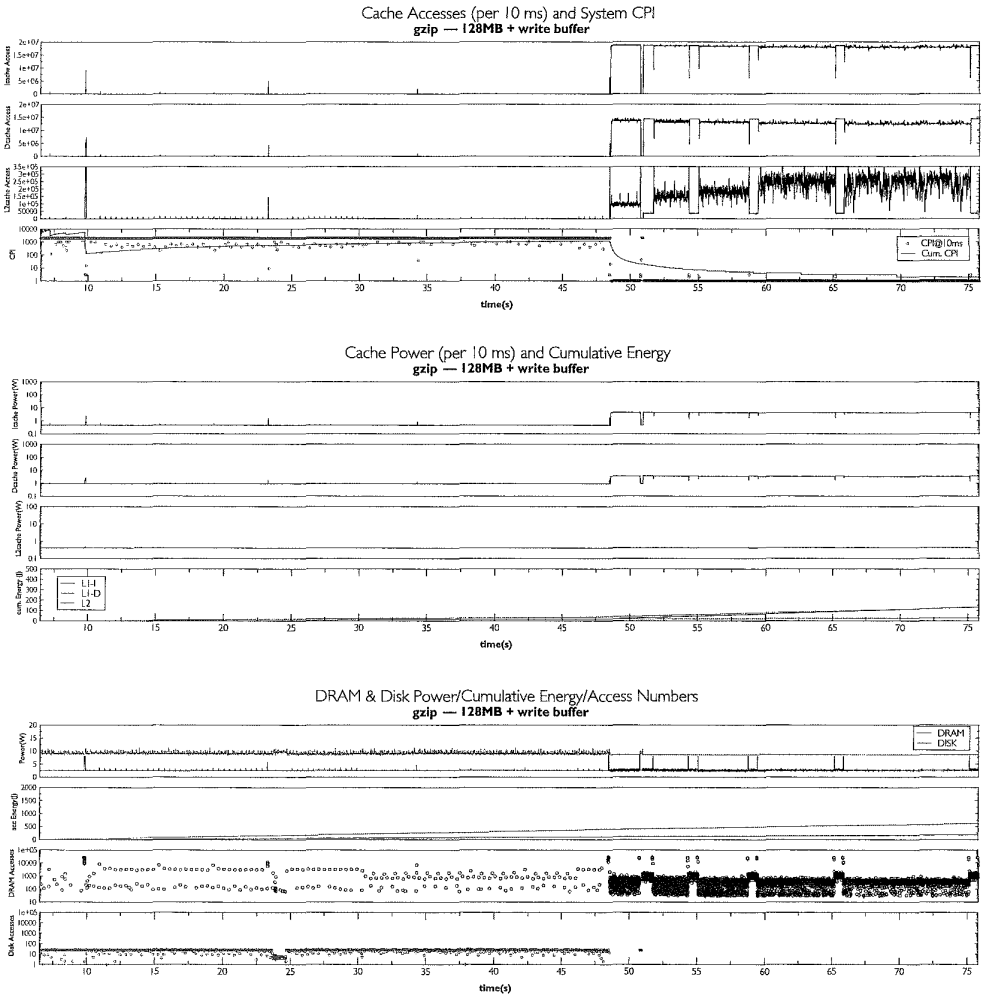


FIGURE Ov.33: Full execution of GZIP, 128 MB DRAM and ideal write buffer. The figure shows the entire run of GZIP. System configuration is a 2 GHz Pentium processor with 128 MB of FB-DIMM main memory and a 12 K-RPM disk drive with built-in disk cache. The figure shows the interaction between all components of the memory system, including the L1 instruction cache, the L1 data cache, the unified L2 cache, the DRAM system, and the disk drive. All graphs use the same *x*-axis, which represents the execution time in seconds. The *x*-axis does not start at zero; the measurements exclude system boot time, invocation of the shell, etc. Each data point represents aggregated (not sampled) activity within a 10-ms epoch. The CPI graph shows two system CPI values: one is the average CPI for each 10-ms epoch, the other is the cumulative average CPI. A duration with no CPI data point indicates that no instructions were executed due to I/O latency. The application is run in single-user mode, as is common for SPEC measurements; therefore, disk delay shows up as stall time. Note that the CPI, the DRAM accesses, and the Disk accesses are plotted on log scales.

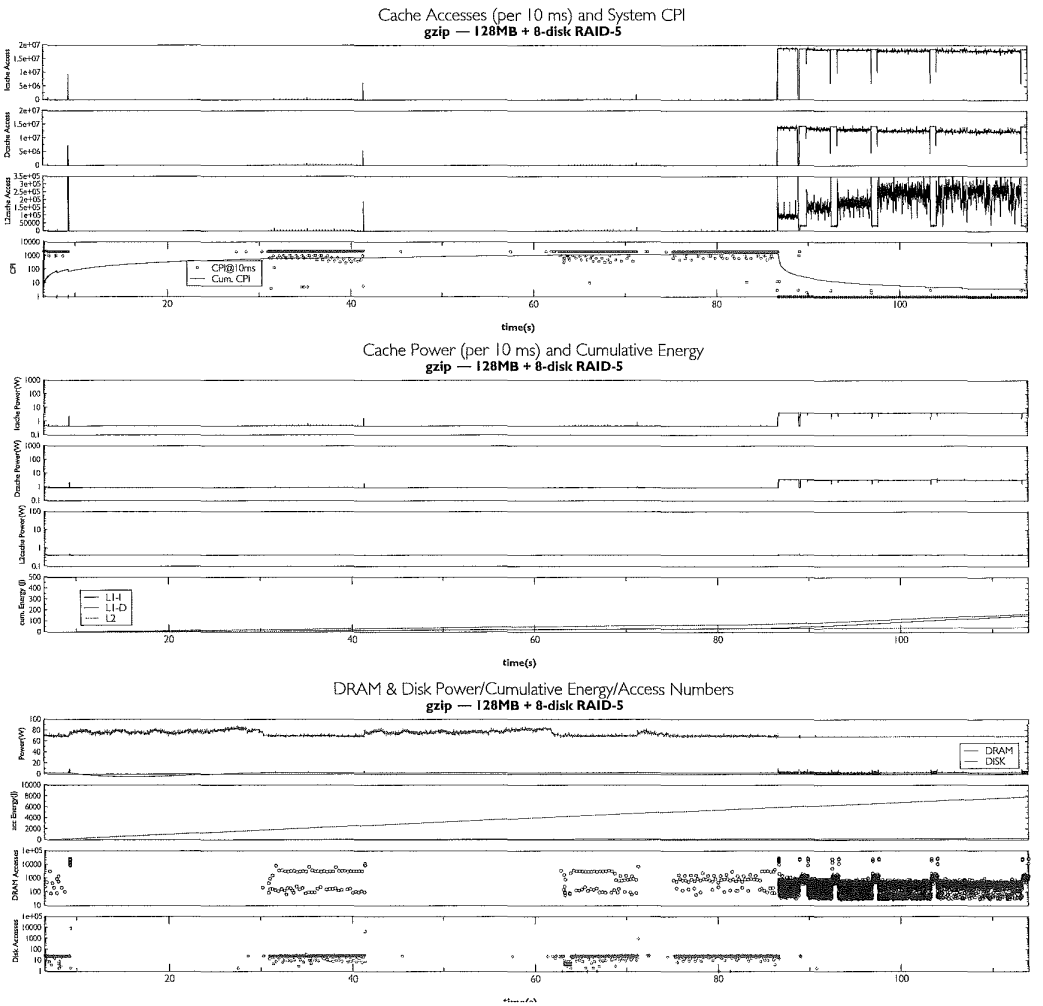
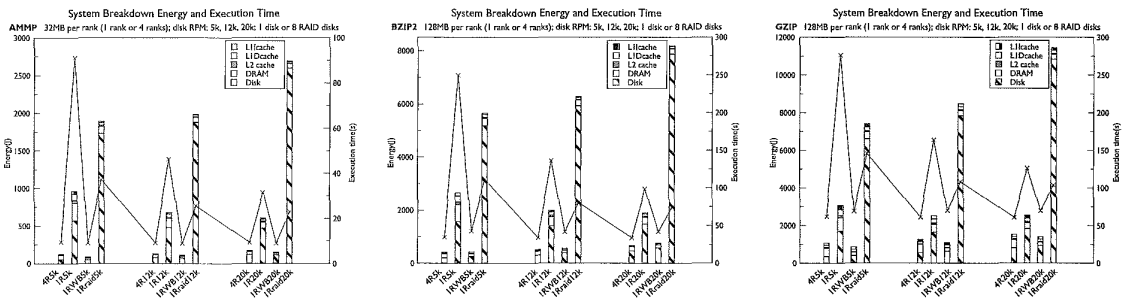


FIGURE Ov.34: Full execution of GZIP, 128 MB DRAM and RAID-5 disk system. The figure shows the entire run of GZIP. System configuration is a 2 GHz Pentium processor with 128 MB of FB-DIMM main memory and a RAID-5 system of eight 12-K-RPM disk drives with built-in disk cache. The figure shows the interaction between all components of the memory system, including the L1 instruction cache, the L1 data cache, the unified L2 cache, the DRAM system, and the disk drive. All graphs use the same *x*-axis, which represents the execution time in seconds. The *x*-axis does not start at zero; the measurements exclude system boot time, invocation of the shell, etc. Each data point represents aggregated (not sampled) activity within a 10-ms epoch. The CPI graph shows two system CPI values: one is the average CPI for each 10-ms epoch, the other is the cumulative average CPI. A duration with no CPI data point indicates that no instructions were executed due to I/O latency. The application is run in single-user mode, as is common for SPEC measurements; therefore, disk delay shows up as stall time. Note that the CPI, the DRAM accesses, and the Disk accesses are plotted on log scales.

TABLE Ov.5 Execution time and energy breakdowns for GZIP and BZIP2

System Configuration (DRAM Size - Disk RPM - Option)	Ex. Time (sec)	L1-I Energy (J)	L1-D Energy (J)	L2 Energy (J)	DRAM Energy (J)	Disk Energy (J)	Total Energy (J)
GZIP							
512 MB–12 K	66.8	129.4	122.1	25.4	440.8	544.1	1261.8
128 MB–12 K	169.3	176.5	216.4	67.7	419.6	1635.4	2515.6
128 MB–12 K–WB	75.8	133.4	130.2	28.7	179.9	622.5	1094.7
128 MB–12 K–RAID	113.9	151	165.5	44.8	277.8	7830	8469.1



average response time, even though the write response time in a RAID system is much higher than the write response time in a single-disk system, this trend does not translate directly into the overall performance. The write response time in a RAID system is higher due to parity calculations, especially the benchmarks with small writes. Despite the improvement in performance, care must be taken in applying RAID because RAID increases the energy proportionally to the number of the disks.

Perhaps the most interesting result in Figure Ov.35 is that the CPI values (top graph) track the disk's *average* ***read*** *response time* (bottom graph) and not the disk's *average response time* (which includes both reads and writes, also bottom graph). This observation holds true for both read-dominated applications and applications with significant write activity (as are **gzip** and **bzip2**). The reason this is interesting is that the disk community tends to report performance numbers in terms of average response time and not average *read* response time, presumably believing the former to be a better indicator of system-level performance than the latter. Our results suggest that the disk community would be better served by continuing to model the effects of write traffic (as it affects read latency) by reporting performance as the average *read* response time.

Ov.4.5 Conclusions

We find that the disk cache can be an effective tool for improving performance at the system level. There is a significant interplay between the DRAM system and the disk's ability to buffer writes and prefetch reads. An ideal write buffer homed within the disk has the potential to move write traffic out of the way and begin working on read requests far sooner, with the result that a system can be made to perform nearly as well as one with four times the amount of main memory, but with roughly half the energy consumption of the configuration with more main memory.

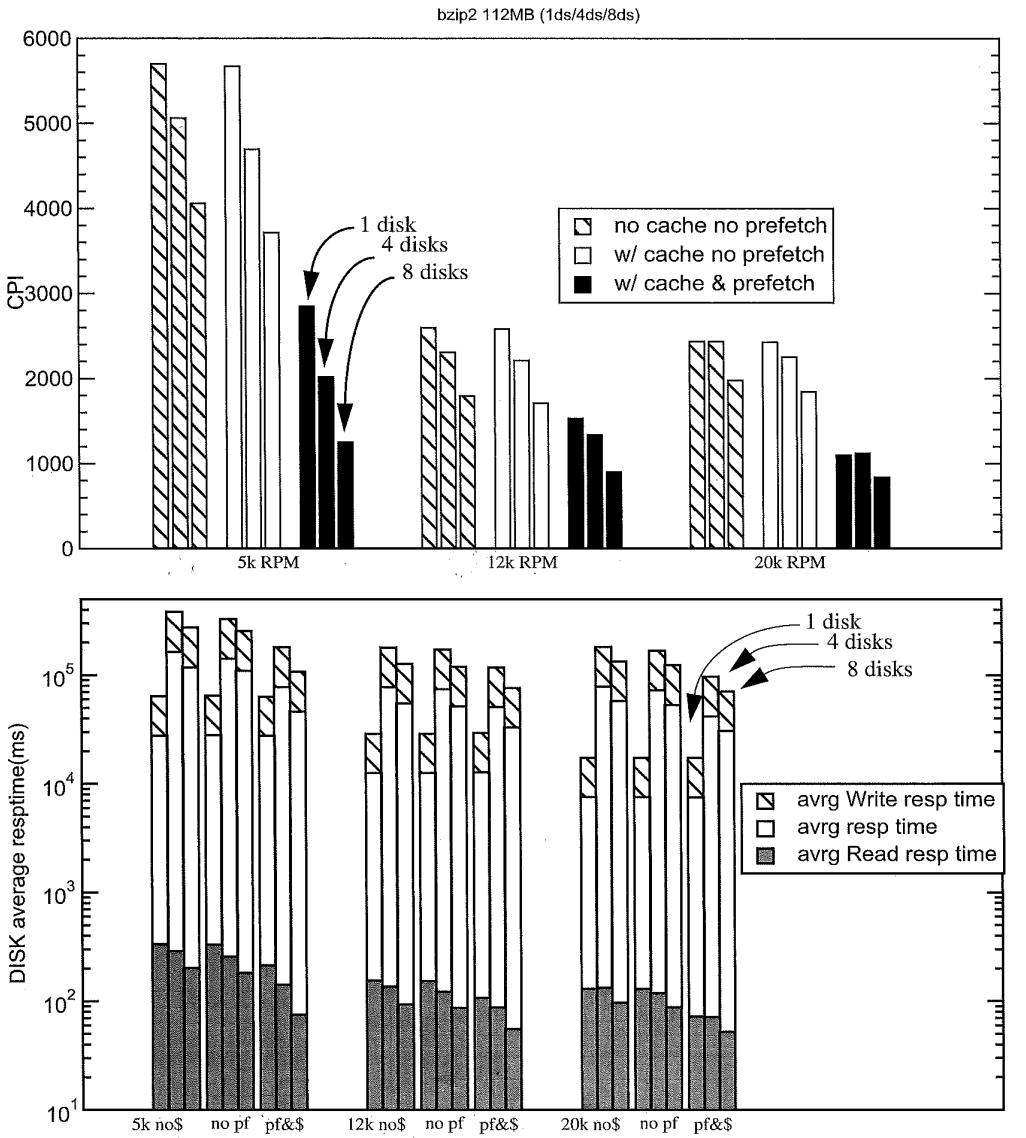


FIGURE Ov.35: The effects of disk prefetching. The experiment tries to identify the effects of prefetching and caching in the disk cache. The configuration is 112 MB of DDR SDRAM running bzip2. The three bars in each group represent a single-disk system, 4-disk RAID-5 system, and 8-disk RAID-5 system. The figure above shows the CPI of each configuration, and the figure below shows the average response time of the disk requests. Note that the CPI axis is in linear scale, but the disk average response time axis is in log scale. The height of the each bar in the average response time graph is the absolute value.

This is extremely important, because FB-DIMM systems are likely to have significant power-dissipation problems, and because of this they will run at the cutting edge of the storage-performance tradeoff. Administrators will configure these systems to use the least amount of storage available to achieve the desired performance, and thus a simple reduction in FB-DIMM storage will result in an unacceptable hit to performance. We have shown that an ideal write buffer in the disk system will solve this problem, transparently to the operating system.

Ov.5 What to Expect

What are the more important architecture-level issues in store for these technologies? On what problems should a designer concentrate?

For caches and SRAMs in particular, power dissipation and reliability are primary issues. A rule of thumb is that SRAMs typically account for at least one-third of the power dissipated by microprocessors, and the reliability for SRAM is the worst of the three technologies.

For DRAMs, power dissipation is becoming an issue with the high I/O speeds expected of future systems. The FB-DIMM, the only proposed architecture seriously being considered for adoption that would solve the capacity-scaling problem facing DRAM systems, dissipates roughly two orders of magnitude more power than a traditional organization (due to an order of magnitude higher per DIMM power dissipation and the ability to put an order of magnitude more DIMMs into a system).

For disks, miniaturization and development of heuristics for control are the primary considerations, but a related issue is the reduction of power dissipation in the drive's electronics and mechanisms. Another point is that some time this year, the industry will be seeing the first generation of hybrid disk drives: those with flash memory to do write caching. Initially, hybrid drives will be available only for mobile applications. One reason for a hybrid drive is to be able to have a disk drive in spin-down mode longer (no need to spin up to do a write). This will save more power and make the battery of a laptop last longer.

For memory systems as a whole, a primary issue is optimization in the face of subsytems that have unanticipated interactions in their design parameters.

From this book, a reader should expect to learn the details of operation and tools of analysis that are necessary for understanding the intricacies and optimizing the behavior of modern memory systems. The designer should expect of the future a memory-system design space that will become increasingly difficult to analyze simply and in which alternative figures of merit (e.g., energy consumption, cost, reliability) will become increasingly important. Future designers of memory systems will have to perform design-space explorations that consider the effects of design parameters in all subsystems of the memory hierarchy, and they will have to consider multiple dimensions of design criteria (e.g., performance, energy consumption, cost, reliability, and real-time behavior).

In short, a holistic approach to design that considers the whole hierarchy is warranted, but this is very hard to do. Among other things, it requires in-depth understanding at all the levels of the hierarchy. It is our goal that this book will enable just such an approach.

PART II

DRAM

The first question I ask myself when something doesn't seem to be beautiful is why do I think it's not beautiful. And very shortly you discover that there is no reason.

— **John Cage**

CHAPTER 7

Overview of DRAMs

DRAM is the "computer memory" that you order through the mail or purchase at the store. It is what you put more of into your computer as an upgrade to improve the computer's performance. It appears in most computers in the form shown in Figure 7.1—the ubiquitous *memory module*, a small computer board (a *printed circuit board*, or *PCB*) that has a handful of chips attached to it. The eight black rectangles on the pictured module are the DRAM chips: plastic packages, each of which encloses a *DRAM die* (a very thin, fragile piece of silicon).

Figure 7.2 illustrates DRAM's place in a typical PC. An individual DRAM device typically connects indirectly to a CPU (i.e., a microprocessor) through a memory controller. In PC systems, the memory controller is part of the *north-bridge* chipset that handles potentially multiple microprocessors, the graphics co-processor, communication to the *south-bridge* chipset (which, in turn, handles all of the system's I/O functions), as well as the interface to the DRAM system. Though still often referred to as "chipsets" these days, the north- and south-bridge chipsets are no longer sets of chips; they are usually implemented as single chips, and in some systems the functions of both are merged into a single die.

Because DRAM is usually an external device by definition, its use, design, and analysis must consider effects of implementation that are often ignored in the use, design, and analysis of on-chip memories such as SRAM caches and scratch-pads. Issues that a designer must consider include the following:

- Pins (e.g., their capacitance and inductance)
- Signaling
- Signal integrity
- Packaging
- Clocking and synchronization
- Timing conventions

Failure to consider these issues when designing a DRAM system is guaranteed to result in a sub-optimal, and quite probably non-functional, design.

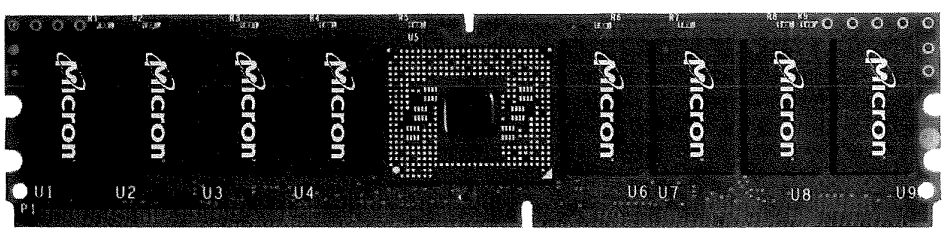


FIGURE 7.1: A memory module. A memory module, or DIMM (dual in-line memory module), is a circuit board with a handful of DRAM chips and associated circuitry attached to it.

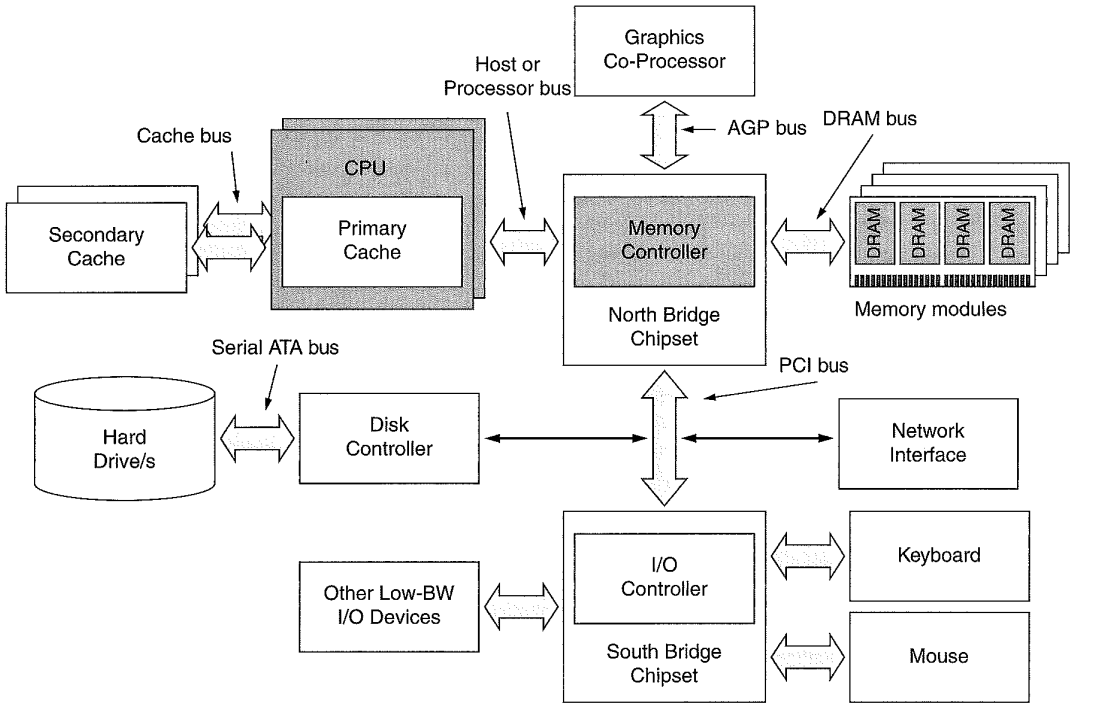


FIGURE 7.2: A typical PC organization. The DRAM subsystem is one part of a relatively complex whole. This figure illustrates a two-way multi-processor, with each processor having its own dedicated secondary cache. The parts most relevant to this report are shaded in darker grey: the CPU, the memory controller, and the individual DRAMs.

Thus, much of this section of the book deals with low-level implementation issues that were not covered in the previous section on caches.

7.1 DRAM Basics: Internals, Operation

A random-access memory (RAM) that uses a single transistor-capacitor pair for each bit is called a dynamic random-access memory or DRAM. Figure 7.3 shows, in the bottom right corner, the circuit for the storage cell in a DRAM. This circuit is *dynamic* because the capacitors storing electrons are not perfect devices, and their eventual leakage requires that, to retain information stored there, each capacitor in the DRAM must be periodically *refreshed* (i.e., read and rewritten).

Each DRAM die contains one or more *memory arrays*, rectangular grids of storage cells with each cell holding one bit of data. Because the arrays are rectangular grids, it is useful to think of them in terms associated with typical grid-like structures. A good example is a Manhattan-like street layout with avenues running north–south and streets running east–west. When one wants to specify a rendezvous location in such a city, one simply designates the intersection of a street and an avenue, and the location is specified without ambiguity. Memory arrays are organized just like this, except where Manhattan is organized into *streets* and *avenues*, memory arrays are organized

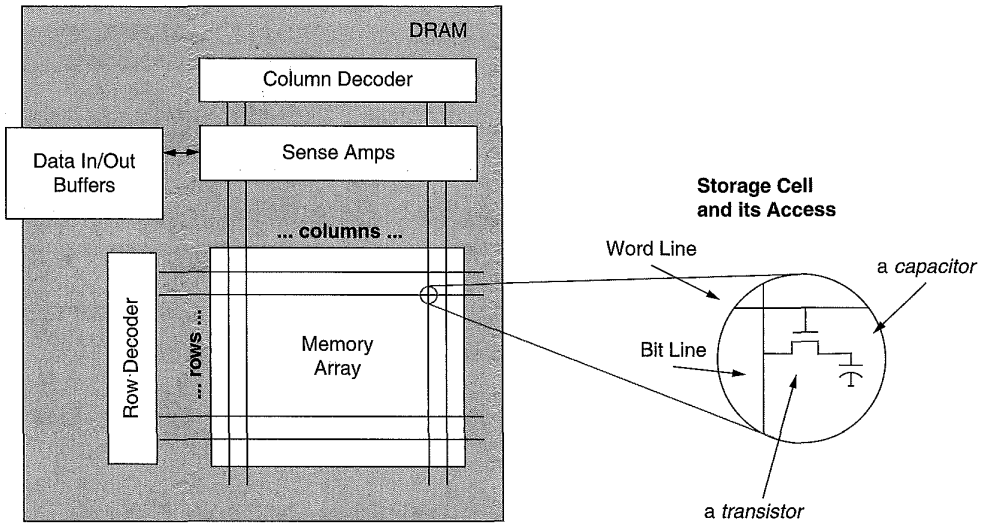


FIGURE 7.3: Basic organization of DRAM internals. The DRAM memory array is a grid of storage cells, where one bit of data is stored at each intersection of a *row* and a *column*.

into *rows* and *columns*. A DRAM chip's memory array with the rows and columns indicated is pictured in Figure 7.3. By identifying the intersection of a row and a column (by specifying a *row address* and a *column address* to the DRAM), a memory controller can access an individual storage cell inside a DRAM chip so as to read or write the data held there.

One way to characterize DRAMs is by the number of memory arrays inside them. Memory arrays within a memory chip can work in several different ways. They can act in unison, they can act completely independently, or they can act in a manner that is somewhere in between the other two. If the memory arrays are designed to act in unison, they operate as a unit, and the memory chip typically transmits or receives a number of bits equal to the number of arrays each time the memory controller accesses the DRAM. For example, in a simple organization, a x4 DRAM (pronounced "by four") indicates that the DRAM has at least four memory arrays and that a column width is 4 bits (each column read or write transmits 4 bits of data). In a x4 DRAM part, four arrays each read 1 data bit in unison, and the part sends out 4 bits of data each time the memory controller makes a column read request. Likewise, a x8 DRAM indicates that the DRAM has at least eight memory arrays and that a column width is 8 bits. Figure 7.4 illustrates the internal organization of x2, x4, and x8 DRAMs. In the past two decades, wider output DRAMs have appeared, and x16 and x32 parts are now common, used primarily in high-performance applications.

Note that each of the DRAM illustrations in Figure 7.4 represents multiple arrays but a single *bank*. Each set of memory arrays that operates independently of other sets is referred to as a bank, not an array. Each bank is independent in that, with only a few restrictions, it can be activated, precharged, read out, etc. at the same time that other banks (on the same DRAM device or on other DRAM devices) are being activated, precharged, etc. The use of multiple independent banks of memory has been a common practice in computer design since DRAMs were invented. In particular, *interleaving* multiple memory banks has been a popular method used to achieve high-bandwidth memory busses using

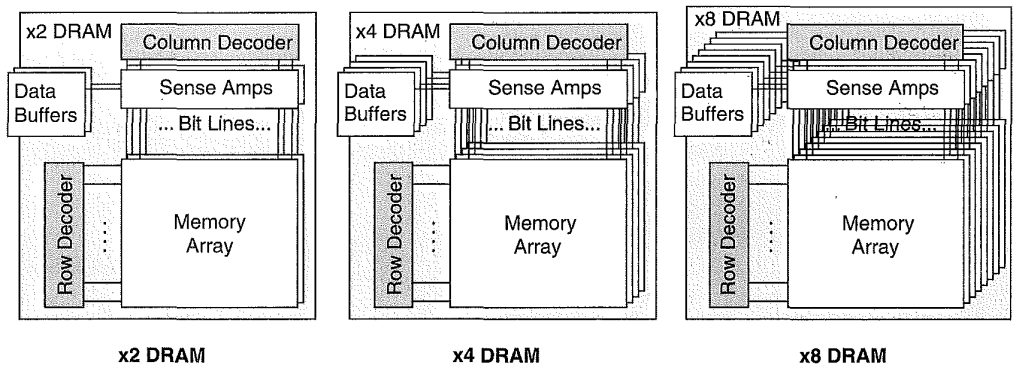


FIGURE 7.4: Logical organization of wide data-out DRAMs. If the DRAM outputs more than one bit at a time, the internal organization is that of multiple arrays, each of which provides one bit toward the aggregate data output.

low-bandwidth devices. In an interleaved memory system, the data bus uses a frequency that is faster than any one DRAM bank can support; the control circuitry toggles back and forth between multiple banks to achieve this data rate. For example, if a DRAM bank can produce a new chunk of data every 10 ns, one can toggle back and forth between two banks to produce a new chunk every 5 ns, or round-robin between four banks to produce a new chunk every 2.5 ns, thereby effectively doubling or quadrupling the data rate achievable by any one bank. This technique goes back at least to the mid-1960s, where it was used in two of the highest performance (and, as it turns out, best documented) computers of the day: the IBM System/360 Model 91 [Anderson et al. 1967] and Seymour Cray's Control Data 6600 [Thornton 1970].

Because a system can have multiple DIMMs, each of which can be thought of as an independent bank, and the DRAM devices on each DIMM can implement internally multiple independent banks, the word "rank" was introduced to distinguish DIMM-level independent operation versus internal-bank-level independent operation. Figure 7.5 illustrates the various levels of organization in a modern DRAM system. A system is composed of potentially many independent DIMMs. Each DIMM may contain one or more independent ranks. Each rank is a set of DRAM devices that operate in unison, and internally each of these DRAM devices implements one or more independent banks. Finally, each bank is composed of slaved memory arrays, where the number of arrays is equal to the data width of the DRAM part (i.e., a x4 part has four slaved arrays per bank). Having concurrency at the rank and bank levels provides bandwidth through the ability to pipeline requests. Having multiple DRAMs acting in unison at the rank level and multiple arrays acting in unison at the bank level provides bandwidth in the form of parallel access.

The busses in a JEDEC-style organization are classified by their function and organization into *data*, *address*, *control*, and *chip-select* busses. An example arrangement is shown in Figure 7.6, which depicts a memory controller connected to two memory modules. The data bus that transmits data to and from the DRAMs is relatively wide. It is often 64 bits wide, and it can be much wider in high-performance systems. A dedicated address bus carries row and column addresses to the DRAMs, and its width grows with the physical storage on a DRAM device (typical widths today are about 15 bits). A control bus is composed of the row and column strobes,[1] output enable, clock, clock enable, and other related signals. These

[1] A "strobe" is a signal that indicates to the recipient that another signal, e.g., data or command, is present and valid.

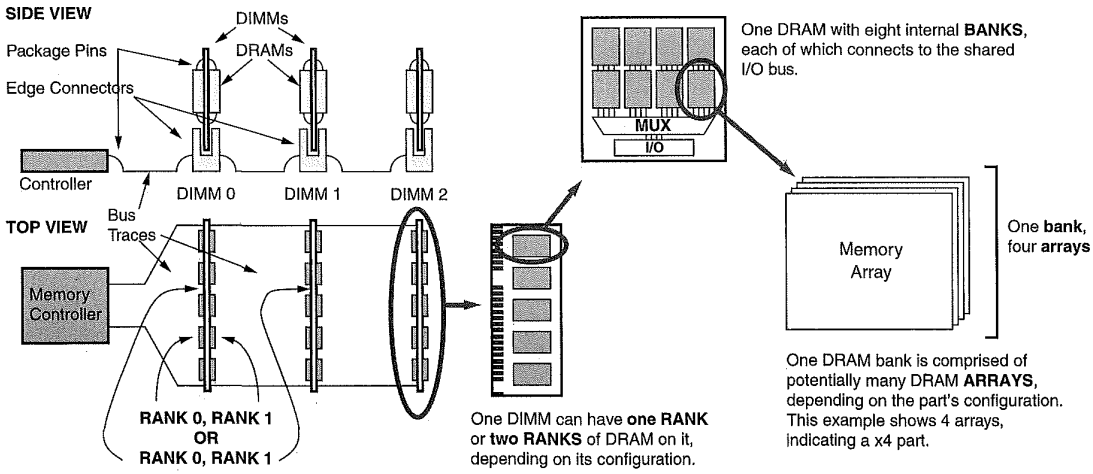


FIGURE 7.5: DIMMs, ranks, banks, and arrays. A system has potentially many DIMMs, each of which may contain one or more ranks. Each rank is a set of ganged DRAM devices, each of which has potentially many banks. Each bank has potentially many constituent arrays, depending on the part's data width.

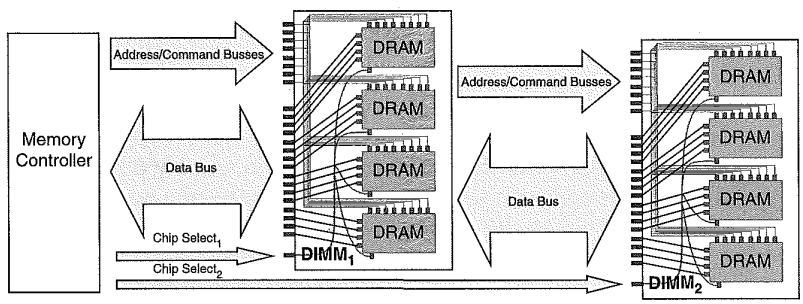
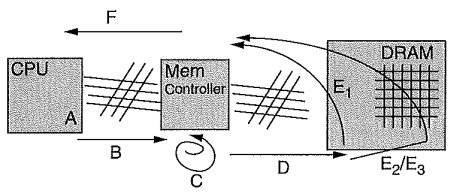


FIGURE 7.6: JEDEC-style memory bus organization. The figure shows a system of a memory controller and two memory modules with a 16-bit data bus and an 8-bit address and command bus.

signals are similar to the address-bus signals in that they all connect from the memory controller to every DRAM in the system. Finally, there is a chip-select network that connects from the memory controller to every DRAM in a *rank* (a separately addressable set of DRAMs). For example, a memory module can contain two ranks of DRAM devices; for every DIMM in the system, there can be two separate chip-select networks, and thus, the size of the chip-select "bus" scales with the maximum amount of physical memory in the system.

This last bus, the chip-select bus, is essential in a JEDEC-style memory system, as it enables the intended recipient of a memory request. A value is asserted on the chip-select bus at the time of a request (e.g., read or write). The chip-select bus



A: Transaction request may be delayed in Queue
B: Transaction request sent to Memory Controller
C: Transaction converted to Command Sequences (may be queued)
D: Command/s Sent to DRAM
E_1: Requires only a **CAS** or
E_2: Requires **RAS + CAS** or
E_3: Requires **PRE + RAS + CAS**
F: Transaction sent back to CPU
DRAM Latency = A + B + C + D + E + F

FIGURE 7.7: System organization and the steps of a DRAM read. Reading data from a DRAM is not as simple as an SRAM, and at several of the stages the request can be stalled.

contains a separate wire for every rank of DRAM in the system. The chip-select signal passes over a wire unique to each small set of DRAMs and enables or disables the DRAMs in that rank so that they, respectively, either handle the request currently on the bus or ignore the request currently on the bus. Thus, only the DRAMs to which the request is directed handle the request. Even though all DRAMs in the system are connected to the same address and control busses and could, in theory, all respond to the same request at the same time, the chip-select bus prevents this from happening.

Figure 7.7 focuses attention on the microprocessor, memory controller, and DRAM device and illustrates the steps involved in a DRAM request. As mentioned previously, a DRAM device connects indirectly to a microprocessor through a memory controller; the microprocessor connects to the memory controller through some form of network (bus, point-to-point, crossbar, etc.); and the memory controller connects to the DRAM through another network (bus, point-to-point, etc.). The memory controller acts as a liaison between the microprocessor and DRAM so that the microprocessor does not need to know the details of the DRAM's operation. The microprocessor presents requests to the memory controller that the memory controller satisfies. The microprocessor connects to potentially many memory controllers at once; alternatively, many microprocessors could be connected to the same memory controller. The simplest case (a *uniprocessor* system) is illustrated in the figure. The memory controller connects to potentially many DRAM devices at once. In particular, DIMMs are the most common physical form in which consumers purchase DRAM, and these are small PCBs with a handful of DRAM devices on each. A memory controller usually connects to at least one DIMM and, therefore, multiple DRAM devices at once.

Figure 7.7 also illustrates the steps of a typical DRAM read operation. After ordering and queueing requests, the microprocessor sends a given request to the memory controller. Once the request arrives at the memory controller, it is queued until the DRAM is ready and all previous and/or higher priority requests have been handled. The memory controller's interface to the DRAM is relatively complex (compared to that of an SRAM, for instance); the row-address strobe (RAS) and column-address strobe (CAS) components are shown in detail in Figure 7.8. Recall from Figure 7.3 that the capacitor lies at the intersection of a wordline and a bitline; it is connected to the bitline through a transistor controlled by the wordline. A transistor is, among other things, a switch, and when the voltage on a wordline goes high, all of the transistors attached to that wordline become closed switches (turned on), connecting their respective capacitors to the associated bitlines. The capacitors at each intersection of wordline and bitline are extremely small and hold a number of electrons that are miniuscule relative to the physical characteristics of those bitlines. Therefore, special circuits called *sense amplifiers* are used to detect the values stored on the capacitors when those capacitors become connected to their associated bitlines. The sense amplifiers first *precharge* the bitlines to a voltage level that is halfway between logic level 0 and logic level 1. When the capacitors are later connected to the bitlines through the transistors, the capacitors change the voltage levels on those bitlines very slightly. The sense amplifiers detect the minute changes and pull

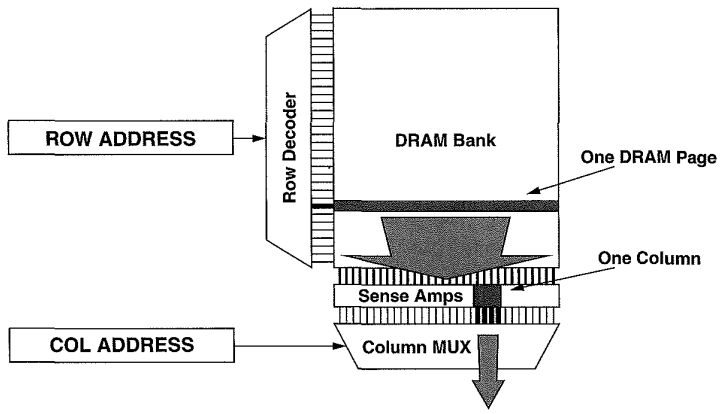


FIGURE 7.8: The multi-phase DRAM-access protocol. The row access drives a DRAM page onto the bitlines to be sensed by the sense amps. The column address drives a subset of the DRAM page onto the bus (e.g., 4 bits).

the bitline voltages all the way to logic level 0 or 1. Bringing the voltage on the bitlines to fully high or fully low, as opposed to the precharged state between high and low, actually recharges the capacitors as long as the transistors remain on.

Returning to the steps in handling the read request. The memory controller must decompose the provided data address into components that identify the appropriate rank within the memory system, the bank within that rank, and the row and column inside the identified bank. The components identifying the row and column are called the *row address* and the *column address*. The bank identifier is typically one or more address bits. The rank number ends up causing a chip-select signal to be sent out over a single one of the separate chip-select lines.

Once the rank, bank, and row are identified, the bitlines in the appropriate bank must be *precharged* (set to a logic level halfway between 0 and 1). Once the appropriate bank has been precharged, the second step is to *activate* the appropriate row inside the identified rank and bank by setting the chip-select signal to activate the set of DRAMs comprising the desired bank, sending the row address and bank identifier over the address bus, and signaling the DRAM's $\overline{\text{RAS}}$ pin (*row-address strobe*—the bar indicates that the signal is active when it is low). This tells the DRAM to send an entire row of data (thousands of bits) into the DRAM's sense amplifiers (circuits that detect and amplify the tiny logic signals represented by the electric charges in the row's storage cells). This typically takes a few tens of nanoseconds, and the step may have already been done (the row or page could already be open or activated, meaning that the sense amps might already have valid data in them).

Once the sense amps have recovered the values, and the bitlines are pulled to the appropriate logic levels, the memory controller performs the last step, which is to *read* the column (*column* being the name given to the data subset of the row that is desired), by setting the chip-select signal to activate the set of DRAMs comprising the desired bank,[2] sending the column address and bank identifier over the address bus, and signaling the DRAM's $\overline{\text{CAS}}$ pin (*column-address strobe*—like $\overline{\text{RAS}}$, the bar indicates that it is

[2]This step is necessary for SDRAMs; it is not performed for older, asynchronous DRAMs (it is subsumed by the earlier chip-select accompanying the RAS).

active when low). This causes only a few select bits[3] in the sense amplifiers to be connected to the output drivers, where they will be driven onto the data bus. Reading the column data takes on the order of tens of nanoseconds. When the memory controller receives the data, it forwards the data to the microprocessor.

The process of transmitting the address in two different steps (i.e., separately transmitted row and column addresses) is unlike that of SRAMs. Initially, DRAMs had minimal I/O pin counts because the manufacturing cost was dominated by the number of I/O pins in the package. This desire to limit I/O pins has had a long-term effect on DRAM architecture; the address pins for most DRAMs are still multiplexed, meaning that two different portions of a data address are sent over the same pins at different times, as opposed to using more address pins and sending the entire address at once.

Most computer systems have a special signal that acts much like a heartbeat and is called the *clock*. A clock transmits a continuous signal with regular intervals of "high" and "low" values. It is usually illustrated as a square wave or semi-square wave with each period identical to the next, as shown in Figure 7.9. The upward portion of the square wave is called the *positive* or *rising edge* of the clock, and the downward portion of the square wave is called the *negative* or *falling edge* of the clock. The primary clock in a computer system is called the system clock or global clock, and it typically resides on the motherboard (the PCB that contains the microprocessor and memory bus). The system clock drives the microprocessor and memory controller and many of the associated peripheral devices directly. If the clock drives the DRAMs directly, the DRAMs are called *synchronous DRAMs*. If the clock does not drive the DRAMs directly, the DRAMs are called *asynchronous DRAMs*. In a synchronous DRAM, steps internal to the DRAM happen in time with one or more edges of this clock. In an asynchronous DRAM, operative steps internal to the DRAM happen when the memory controller commands the DRAM to act, and those commands typically happen in time with one or more edges of the system clock.

7.2 Evolution of the DRAM Architecture

In the 1980s and 1990s, the conventional DRAM interface started to become a performance bottleneck in high-performance as well as desktop systems. The improvement in the speed and performance of microprocessors was significantly outpacing the improvement in speed and performance of DRAM chips. As a consequence, the DRAM interface began to evolve, and a number of revolutionary proposals [Przybylski 1996] were made as well. In most cases, what was considered evolutionary or revolutionary was the proposed *interface* (the mechanism by which the microprocessor accesses the DRAM). The DRAM core (i.e., what is pictured in Figure 7.3) remains essentially unchanged.

Figure 7.10 shows the evolution of the basic DRAM architecture from *clocked* to the conventional *asynchronous* to *fast page mode* (FPM) to *extended data-out* (EDO) to *burst-mode EDO* (BEDO) to *synchronous* (SDRAM). The figure shows each as a stylized DRAM in terms of the memory array, the sense amplifiers, and the column multiplexer (as well as additional components if appropriate).

As far as the first evolutionary path is concerned (asynchronous through SDRAM), the changes have largely been structural in nature, have been relatively minor in terms of cost and physical implementation, and have targeted increased throughput. Since SDRAM, there has been a profusion of designs proffered by the DRAM industry, and we lump these new DRAMs into two categories: those targeting reduced latency and those targeting increased throughput.

7.2.1 Structural Modifications Targeting Throughput

Compared to the conventional DRAM, FPM simply allows the row to remain open across multiple $\overline{\text{CAS}}$ commands, requiring very little additional circuitry. To this, EDO changes the output drivers to become output latches so that they hold the data valid on the

[3]One bit in a "x1" DRAM, two bits in a "x2" DRAM, four bits in a "x4" DRAM, etc.

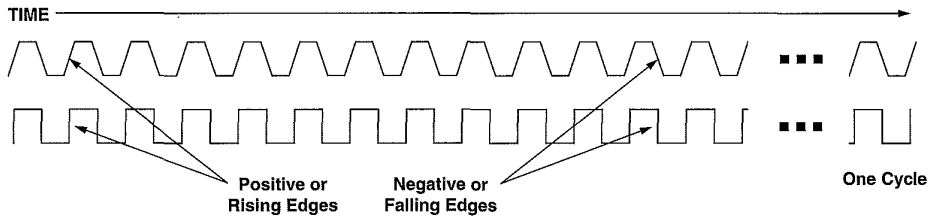


FIGURE 7.9: Example clock signals. Clocks are typically shown as square waves (bottom) or sort of square waves (top). They repeat *ad infinitum*, and the repeating shape is called a clock cycle. The two clocks pictured above have the same frequency—the number of cycles in a given time period.

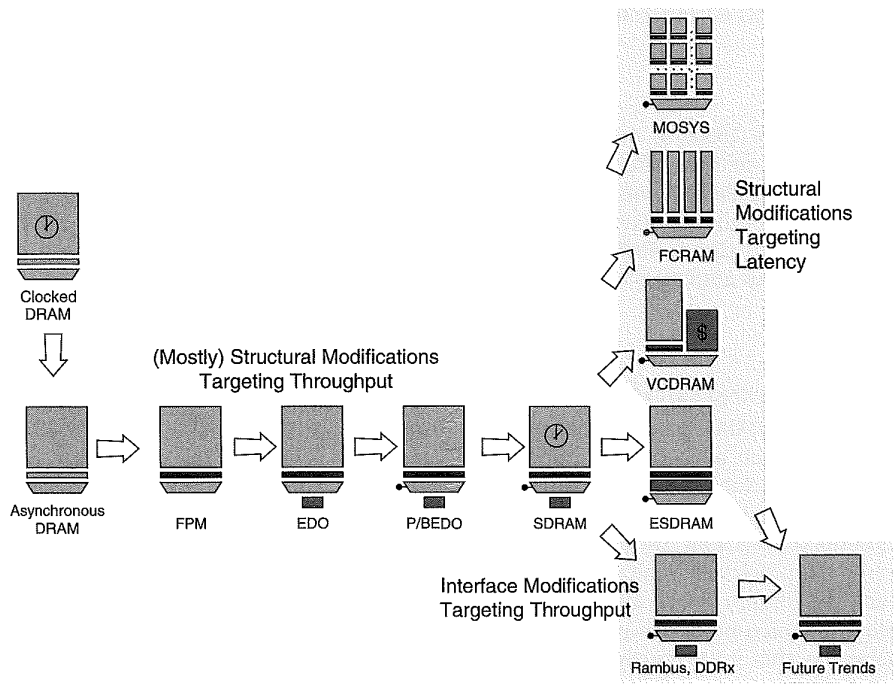


FIGURE 7.10: Evolution of the DRAM architecture. To the original DRAM design, composed of an array, a block of sense amps, and a column multiplexor, the fast page mode (FPM) design adds the ability to hold the contents of the sense amps valid over multiple column accesses. To the FPM design, extended data-out (EDO) design adds an output latch after the column multiplexor. To the EDO design, the burst EDO (BEDO) design adds a counter that optionally drives the column-select address latch. To the BEDO design, the synchronous DRAM (SDRAM) design adds a clock signal that drives both row-select and column-select circuitry (not just the column-select address latch).

bus for a longer period of time. To this, BEDO adds an internal counter that drives the address latch so that the memory controller does not need to supply a new address to the DRAM on every $\overline{CAS}$ command if the desired address is simply one off from the previous $\overline{CAS}$ command. Thus, in BEDO, the DRAM's column-select circuitry is driven from an internally generated signal, not an externally generated signal; the source of the control signal is close to the circuitry that it controls in space and therefore time, and this makes the timing of the circuit's activation more precise. Finally, SDRAM takes this perspective one step further and drives all internal circuitry (row select, column select, data read-out) by a clock, as opposed to the $\overline{RAS}$ and $\overline{CAS}$ strobes. The following paragraphs describe this evolution in more detail.

Clocked DRAM

The earliest DRAMs (1960s to mid-1970s, before de facto standardization) were often clocked [Rhoden 2002, Sussman 2002, Padgett and Newman 1974]; DRAM commands were driven by a periodic clock signal. Figure 7.10 shows a stylized DRAM in terms of the memory array, the sense amplifiers, and the column multiplexer.

The Conventional Asynchronous DRAM

In the mid-1970s, DRAMs moved to the asynchronous design with which most people are familiar. These DRAMs, like the clocked versions before them, require that every single access go through all of the steps described previously: for every access, the bitlines need to be precharged, the row needs to be activated, and the column is read out after row activation. Even if the microprocessor wants to request the same data row that it previously requested, the entire process (row activation followed by column read/write) must be repeated. Once the column is read, the row is deactivated or closed, and the bitlines are precharged. For the next request, the entire process is repeated, even if the same datum is requested twice in succession. By convention and circuit design, both $\overline{RAS}$ and $\overline{CAS}$ must rise in unison. For example, one cannot hold $\overline{RAS}$ low while toggling $\overline{CAS}$. Figure 7.11 illustrates the timing for the conventional asynchronous DRAM.

Fast Page Mode DRAM (FPM DRAM)

FPM DRAM implements page mode, an improvement on conventional DRAM in which the row address is held constant and data from multiple columns is read from the sense amplifiers. This simply lifts the restriction described in the previous paragraph: the memory controller may hold $\overline{RAS}$ low while toggling $\overline{CAS}$, thereby creating a de facto cache out of the data held active in the sense amplifiers. The data held in the sense amps form an "open page" that can be accessed relatively quickly. This speeds up successive accesses to the same row of the DRAM core, as is very common in computer systems (the term is locality of reference and indicates that oftentimes memory requests that are nearby in time are also nearby in the memory-address space and would therefore likely lie within the same DRAM row). Figure 7.12 gives the timing for FPM reads.

Extended Data-Out DRAM (EDO DRAM)

EDO DRAM, sometimes referred to as hyper-page mode DRAM, adds a few transistors to the output drivers of an FPM DRAM to create a latch between the sense amps and the output pins of the DRAM. This latch holds the output pin state and permits the $\overline{CAS}$ to rapidly deassert, allowing the memory array to begin precharging sooner. In addition, the latch in the output path also implies that the data on the outputs of the DRAM circuit remain valid longer into the next clock phase, relative to previous DRAM architectures (thus the name "extended data-out"). By permitting the memory array to begin precharging sooner, the addition of a latch allows EDO DRAM to operate faster than FPM DRAM. EDO enables the microprocessor to access memory at least 10 to 15% faster than with FPM [Kingston 2000, Cuppu et al. 1999, 2001]. Figure 7.13 gives the timing for an EDO read.

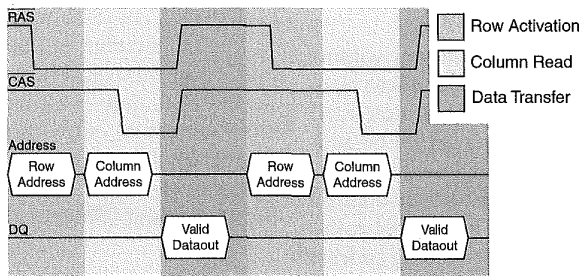


FIGURE 7.11: Read timing for the asynchronous DRAM.

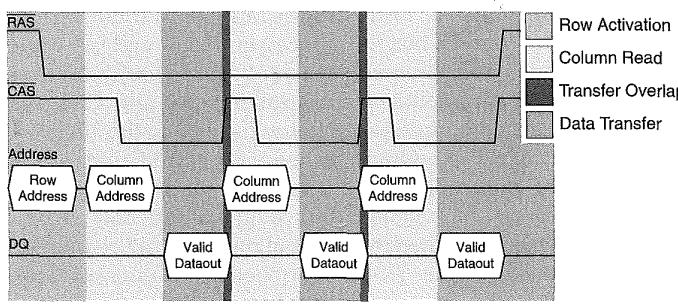


FIGURE 7.12: FPM read timing. The FPM allows the DRAM controller to hold a row constant and receive multiple columns in rapid succession.

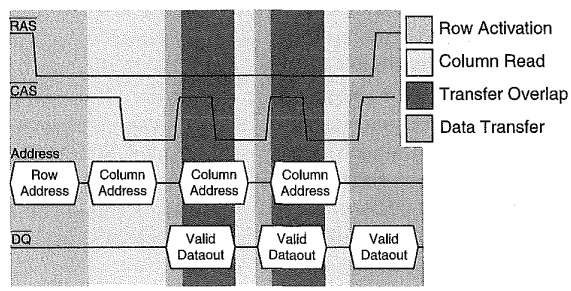


FIGURE 7.13: EDO read timing. The output latch in EDO DRAM allows more overlap between column access and data transfer than in FPM.

Burst-Mode EDO DRAM (BEDO DRAM)

Although BEDO DRAM never reached the volume of production that EDO and SDRAM did, it was positioned to be the next-generation DRAM after EDO [Micron 1995]. BEDO builds on EDO DRAM by adding the concept of "bursting" contiguous blocks of data from an activated row each time a new column address is sent to the DRAM chip. An internal counter was added that first accepts the incoming address and then increments that value on every successive toggling of $\overline{\text{CAS}}$, driving the incremented value into the column-address latch. With each toggle of the $\overline{\text{CAS}}$, the DRAM chip sends the next sequential column of data onto the bus. In previous DRAMs, the column-address latch was driven by an externally generated address signal. By eliminating the need to send successive column addresses over the bus to drive a burst of data in response to each microprocessor request, BEDO eliminates a significant amount of timing uncertainty between successive addresses, thereby increasing the rate at which data can be read from the DRAM. In practice, the minimum cycle time for driving the output bus was reduced by roughly 30% compared to EDO DRAM [Prince 2000], thereby increasing bandwidth proportionally. Figure 7.14 gives the timing for a BEDO read.

IBM's High-Speed Toggle Mode DRAM

IBM's High-Speed Toggle Mode ("toggle mode") is a high-speed DRAM interface designed and fabricated in the late 1980s and presented at the International Solid-State Circuits Conference in February 1990 [Kalter et al. 1990a]. In September 1990, IBM presented toggle mode to JEDEC as an option for the next-generation DRAM architecture (minutes of JC-42.3 meeting 55). Toggle mode transmits data to and from a DRAM on both edges of a high-speed data strobe rather than transferring data on a single edge of the strobe. The strobe was very high speed for its day; Kalter reports a 10-ns data cycle time—an effective 100 MHz data rate—in 1990 [Kalter et al. 1990b]. The term "toggle" is probably derived from its implementation: to obtain twice the normal data rate,[4] one would toggle a signal pin which would cause the DRAM to toggle back and forth between two different (interleaved) output buffers, each of which would be pumping data out at half the speed of the strobe [Kalter et al. 1990b]. As proposed to JEDEC, it offered burst lengths of 4 or 8 bits of data per memory access.

Synchronous DRAM (SDRAM)

Conventional, FPM, and EDO DRAM are controlled asynchronously by the memory controller. Therefore, in theory, the memory latency and data toggle rate can be some fractional number of microprocessor clock cycles.[5] More importantly, what makes the DRAM asynchronous is that the memory controller's RAS and $\overline{\text{CAS}}$ signals directly control latches internal to the DRAM, and those signals can arrive at the DRAM's pins at any time. An alternative is to make the DRAM interface synchronous such that requests can only arrive at regular intervals. This allows the latches internal to the DRAM to be controlled by an internal clock signal. The primary benefit is similar to that seen in BEDO: by associating all data and control transfer with a clock signal, the timing of events is made more predictable. Such a scheme, by definition, has less skew. Reduction in skew means that the system can potentially achieve faster turnaround on requests, thereby yielding higher throughput. A timing diagram for synchronous DRAM is shown in Figure 7.15. Like BEDO DRAMs, SDRAMs support the concept of a burst mode; SDRAM devices have a programmable register that holds a burst length. The DRAM uses this to determine how many columns to output over successive cycles; SDRAM may therefore return many bytes over several cycles per request. One advantage of this is the elimination of the timing signals (i.e., toggling $\overline{\text{CAS}}$) for each successive burst, which

[4]The term "normal" implies the data cycling at half the data-strobe rate.
[5]In practice, this is not the case, as the memory controller and DRAM subsystem are driven by the system clock, which typically has a period that is an integral multiple of the microprocessor's clock.

reduces the command bandwidth used. The underlying architecture of the SDRAM core is the same as in a conventional DRAM.

The evolutionary changes made to the DRAM interface up to and including BEDO have been relatively inexpensive, especially when considering the pay-off: FPM was essentially free compared to the conventional design, EDO simply added a latch, and BEDO added a counter and mux. Each of these evolutionary changes added only a small amount of logic, yet each improved upon its predecessor by as much as 30% in terms of system performance [Cuppu et al. 1999, 2001]. Though SDRAM represented a more significant cost in implementation and offered no performance improvement over BEDO at the same clock speeds,[6] the presence of a source-synchronous data strobe in its interface (in this case, the global clock signal) would allow SDRAM to scale to much

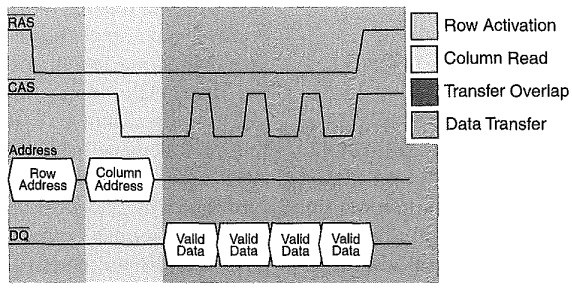


FIGURE 7.14: BEDO read timing. By driving the column-address latch from an internal counter rather than an external signal, the minimum cycle time for driving the output bus was reduced by roughly 30% over EDO.

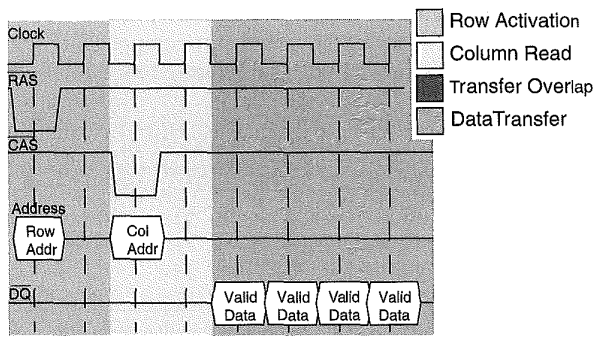


FIGURE 7.15: SDR SDRAM read operation clock diagram (CAS-2).

[6]By some estimates, BEDO actually had higher performance than SDRAM [Williams 2001].

higher switching speeds more easily than the earlier asynchronous DRAM interfaces such as FPM and EDO.[7] Note that this benefit applies to any interface with a source-synchronous data strobe signal, whether the interface is synchronous or asynchronous, and therefore, an asynchronous burst-mode DRAM with source-synchronous data strobe could have scaled to higher switching speeds just as easily—witness the February 1990 presentation of a working 100-MHz asynchronous burst-mode part from IBM, which used a dedicated pin to transfer the source-synchronous data strobe [Kalter 1990a, b].

7.2.2 Interface Modifications Targeting Throughput

Since the appearance of SDRAM in the mid-1990s, there has been a large profusion of novel DRAM architectures proposed in an apparent attempt by DRAM manufacturers to make DRAM less of a commodity [Dipert 2000]. One reason for the profusion of competing designs is that we have apparently run out of the same sort of "free" ideas that drove the earlier DRAM evolution. Since BEDO, there has been no architecture proposed that provides a 30% performance advantage at near-zero cost; all proposals have been relatively expensive. As Dipert suggests, there is no clear heads-above-the-rest winner yet because many schemes seem to lie along a linear relationship between additional cost of implementation and realized performance gain. Over time, the market will most likely decide the winner; those DRAM proposals that provide sub-linear performance gains relative to their implementation cost will be relegated to zero or near-zero market share.

Rambus DRAM (RDRAM, Concurrent RDRAM, and Direct RDRAM)

Rambus DRAM (RDRAM) is very different from traditional main memory. It uses a bus that is significantly narrower than the traditional bus, and, at least in its initial incarnation, it does not use dedicated address, control, data, and chip-select portions of the bus. Instead, the bus is fully multiplexed: the address, control, data, and chip-select information all travel over the same set of electrical wires but at different times. The bus is 1 byte wide, runs at 250 Mhz, and transfers data on both clock edges to achieve a theoretical peak bandwidth of 500 MB/s. Transactions occur on the bus using a split request/response protocol. The packet transactions resemble network request/response pairs: first an address/control packet is driven, which contains the entire address (row address and column address), and then the data is driven. Different transactions can require different numbers of cycles, depending on the transaction type, location of the data within the device, number of devices on the channel, etc. Figure 7.16 shows a typical read transaction with an arbitrary latency.

Because of the bus's design—being a single bus and not composed of separate segments dedicated to separate functions—only one transaction can use

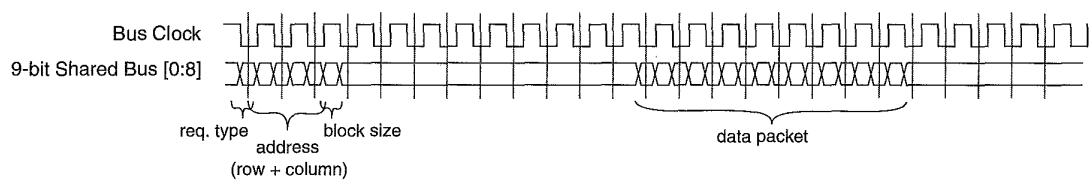


FIGURE 7.16: Rambus read clock diagram for block size 16. The original Rambus design (from the 1990 patent application) had but a single bus multiplexed between data and address/control. The request packet is six "cycles," where a cycle is one beat of a cycle, not one full period of a cycle.

[7]Making an interface synchronous is a well-known technique to simplify the interface and to ease scaling to higher switching speeds [Stone 1982].

the bus during any given cycle. This limits the bus's potential *concurrency* (its ability to do multiple things simultaneously). Due to this limitation, the original RDRAM design was not considered well suited to the PC main memory market [Przybylski 1996], and the interface was redesigned in the mid-1990s to support more concurrency.

Specifically, with the introduction of "Concurrent RDRAM," the bus was divided into separate address, command, and data segments reminiscent of a JEDEC-style DRAM organization. The data segment of the bus remained 1 byte wide, and to this was added a 1-bit address segment and a 1-bit control segment. By having three separate, dedicated segments of the bus, one could perform potentially three separate, simultaneous actions on the bus. This divided and dedicated arrangement simplified transaction scheduling and increased performance over RDRAM accordingly. Note that at this point, Rambus also moved to a four clock cycle period, referred to as an octcycle. Figure 7.17 gives a timing diagram for a read transaction.

One of the few limitations to the "Concurrent" design was that the data bus sometimes carried a brief packet of address information, because the 1-bit address bit was too narrow. This limitation has been removed in Rambus' latest DRAMs. The divided arrangement introduced in Concurrent RDRAM has been carried over into the most recent incarnation of RDRAM, called "Direct RDRAM," which increases the width of the data segment to 2 bytes, the width of the address segment to 5 bits, and the width of the control segment to 3 bits. These segments remain separate and dedicated—similar to a JEDEC-style organization—and the control and address segments are wide enough that the data segment of the bus never needs to carry anything but data, thereby increasing data throughput on the channel. Bus operating speeds have also changed over the years, and the latest designs are more than double the original speeds (500 MHz bus frequency). Each half-row buffer in Direct RDRAM is shared between adjacent banks, which implies that adjacent banks cannot be active simultaneously. This organization has the result of increasing the row buffer miss rate as compared to having one open row per bank, but it reduces the cost by reducing the die area occupied by the row buffers, compared to 16 full row buffers. Figure 7.18 gives a timing diagram for a read operation.

Double Data Rate DRAM (DDR SDRAM)

Double data rate (DDR) SDRAM is the modern equivalent of IBM's High-Speed Toggle Mode. DDR doubles the data bandwidth available from single data rate SDRAM by transferring data at both edges of the clock (i.e., both the rising edge and the falling

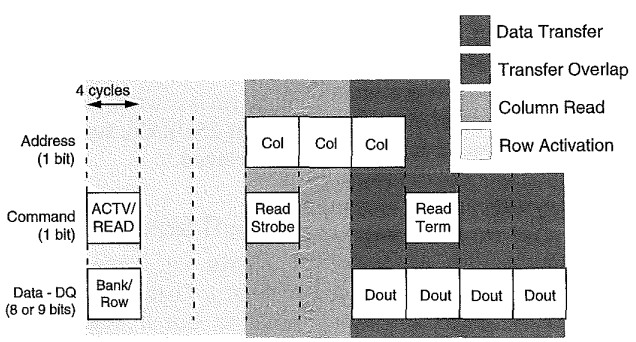


FIGURE 7.17: Concurrent RDRAM read operation. Concurrent RDRAMs transfer on both edges of a fast clock and use a 1-byte data bus multiplexed between data and addresses.

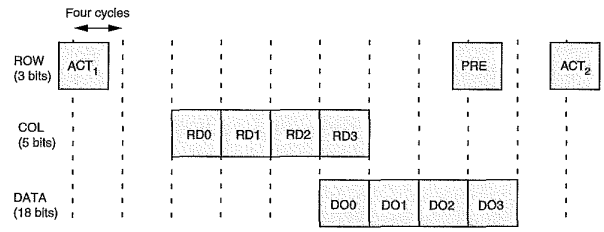


FIGURE 7.18: Direct Rambus read clock diagram. Direct RDRAMs transfer on both edges of a fast clock and use a 2-byte data bus dedicated to handling data only.

edge), much like the toggle mode's dual-edged clocking scheme. DDR SDRAM is very similar to single data rate SDRAM in all other characteristics. They use the same signaling technology, the same interface specification, and the same pin-outs on the DIMM carriers. However, DDR SDRAM's internal transfers from and to the SDRAM array, respectively, read and write twice the number of bits as SDRAM. Figure 7.19 gives a timing diagram for a CAS-2 read operation.

7.2.3 Structural Modifications Targeting Latency

The following DRAM offshoots represent attempts to lower the latency of the DRAM part, either by increasing the circuit's speed or by improving the average latency through caching.

Virtual Channel Memory (VCDRAM)

Virtual channel adds a substantial SRAM cache to the DRAM that is used to buffer large blocks of data (called segments) that might be needed in the future. The SRAM segment cache is managed explicitly by the memory controller. The design adds a new step in the DRAM-access protocol: a row activate operation moves a page of data into the sense amps; "prefetch" and "restore" operations (data read and data write, respectively) move data between the sense amps and the SRAM segment cache one segment at a time; and column read or write operations move a column of data between the segment cache and the output buffers. The extra step adds latency to read and write operations, unless all of the data required by the application fits in the SRAM segment cache.

Enhanced SDRAM (ESDRAM)

Like EDO DRAM, ESDRAM adds an SRAM latch to the DRAM core, but whereas EDO adds the latch after the column mux, ESDRAM adds it *before* the column mux. Therefore, the latch is as wide as a DRAM page. Though expensive, the scheme allows for better overlap of activity. For instance, it allows row precharge to begin immediately without having to close out the row (it is still active in the SRAM latch). In addition, the scheme allows a write-around mechanism whereby an incoming write can proceed without the need to close out the currently active row. Such a feature is useful for write-back caches, where the data being written at any given time is not likely to be in the same row as data that is currently being read from the DRAM. Therefore, handling such a write delays future reads to the same row. In ESDRAM, future reads to the same row are not delayed.

MoSys 1T-SRAM

MoSys, i.e., Monolithic System Technology, has created a "1-transistor SRAM" (which is not really possible, but it makes for a catchy name). Their design wraps an SRAM interface around an extremely fast DRAM core to create an SRAM-compatible part that approaches the storage and power consumption

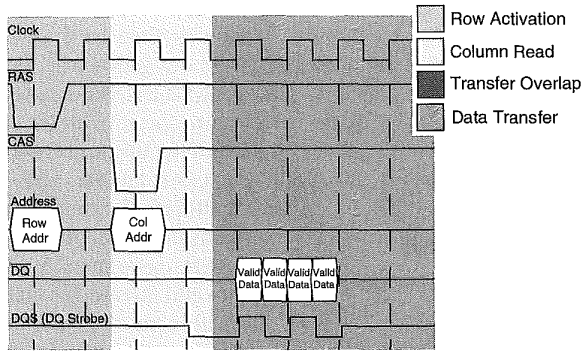


FIGURE 7.19: DDR SDRAM read timing (CAS-2). DDR SDRAMs use both a clock and a source-synchronous data strobe (DQS) to achieve high data rates. DQS is used by the DRAM to sample incoming write data; it is typically ignored by the memory controller on DRAM reads.

characteristics of a DRAM, while simultaneously approaching the access-time characteristics of an SRAM. The fast DRAM core is made up of a very large number of independent banks; decreasing the size of a bank makes its access time faster, but increasing the number of banks complicates the control circuitry (and therefore cost) and decreases the part's effective density. No other DRAM manufacturer has gone to the same extremes as MoSys to create a fast core, and thus, the MoSys DRAM is the lowest latency DRAM in existence. However, its density is low enough that OEMs have not yet used it in desktop systems in any significant volume. Its niche is high-speed embedded systems and game systems (e.g., Nintendo GameCube).

Reduced Latency DRAM (RLDRAM)

Reduced latency DRAM (RLDRAM) is a fast DRAM core that has no DIMM specification: it must be used in a direct-to-memory-controller environment (e.g., inhabiting a dedicated space on the motherboard). Its manufacturers suggest its use as an extremely large off-chip cache, probably at a lower spot in the memory hierarchy than any SRAM cache. Interfacing directly to the chip, as opposed to through a DIMM, decreases the potential for clock skew, thus the part's high-speed interface.

Fast Cycle DRAM (FCRAM)

Fujitsu's fast cycle RAM (FCRAM) achieves a low-latency data access by segmenting the data array into subarrays, only one of which is driven during a row activate. This is similar to decreasing the size of an array, thus its effect on access time. The subset of the data array is specified by adding more bits to the row address, and therefore the mechanism is essentially putting part of the column address into the row activation (e.g., moving part of the column-select function into row activation). As opposed to RLDRAM, the part does have a DIMM specification, and it has the highest DIMM bandwidth available, in the DRAM parts surveyed.

7.2.4 Rough Comparison of Recent DRAMs

Latter-day advanced DRAM designs have abounded, largely because of the opportunity to appeal to markets asking for high performance [Dipert 2000] and because engineers have evidently run out of design ideas that echo those of the early-on evolution; that is, design ideas that are relatively

inexpensive to implement and yet yield tremendous performance advantages. The DRAM industry tends to favor building the simplest design that achieves the desired benefits [Lee 2002, DuPreez 2002, Rhoden 2002]. Currently, the dominant DRAM in the high-performance design arena is DDR.

7.3 Modern-Day DRAM Standards

DRAM is a commodity; in theory, any DRAM chip or DIMM is equivalent to any other that has similar specifications (width, capacity, speed grade, interface, etc.). The standard-setting body that governs this compatibility is JEDEC, an organization formerly known as the *Joint Electron Device Engineering Council.* Following a recent marketing decision reminiscent of Kentucky Fried Chicken's move to be known as simply "KFC," the organization is now known as the "JEDEC" Solid-State Technology Association. Working within the Electronic Industries Alliance (EIA), JEDEC covers the standardization of discrete semiconductor devices and integrated circuits. The work is done through 48 committees and their constituent subcommittees; anyone can become a member of any committee, and so any individual or corporate representative can help to influence future standards. In particular, DRAM device standardization is done by the 42.3 subcommittee (JC-42.3).

7.3.1 Salient Features of JEDEC's SDRAM Technology

JEDEC SDRAMs use the traditional DRAM-system organization, described earlier and illustrated in Figure 7.6. There are four different busses, with each classified by its function—a "memory bus" in this organization is actually composed of separate (1) data, (2) address, (3) control, and (4) chip-select busses. Each of these busses is dedicated to handle only its designated function, except in a few instances, for example, when control information is sent over an otherwise unused address bus wire. (1) The data bus is relatively wide: in modern PC systems, it is 64 bits wide, and it can be much wider in high-performance systems. (2) The width of the address bus grows with the number of bits stored in an individual DRAM device; typical address busses today are about 15 bits wide. (3) A control bus is composed of the row and column strobes, output enable, clock, clock enable, and other similar signals that connect from the memory controller to every DRAM in the system. (4) Finally, there is a chip-select network that uses one unique wire per DRAM rank in the system and thus scales with the maximum amount of physical memory in the system. Chip select is used to enable ranks of DRAMs and thereby allow them to read commands off the bus and read/write data off/onto the bus.

The primary difference between SDRAMs and earlier asynchronous DRAMs is the presence in the system of a clock signal against which all actions (command and data transmissions) are timed. Whereas asynchronous DRAMs use the RAS and CAS signals as strobes—that is, the strobes directly cause the DRAM to sample addresses and/or data off the bus—SDRAMs instead use the clock as a strobe, and the RAS and CAS signals are simply commands that are themselves sampled off the bus in time with the clock strobe. The reason for timing transmissions with a regular (i.e., periodic) free-running clock instead of the less regular RAS and CAS strobes was to achieve higher dates more easily; when a regular strobe is used to time transmissions, timing uncertainties can be reduced, and therefore data rates can be increased.

Note that any regular timing signal could be used to achieve higher data rates in this way; a free-running clock is not necessary [Lee 2002, Rhoden 2002, Karabotsos 2002, Baker 2002, Macri 2002]. In DDR SDRAMs, the clock is all but ignored in the data transfer portion of write requests: the DRAM samples the incoming data with respect not to the clock, but instead to a separate, regular signal known as DQS [Lee 2002, Rhoden 2002, Macri 2002, Karabotsos 2002, Baker 2002]. The implication is that a free-running clock could be dispensed with entirely, and the result would be something very close to IBM's toggle mode.

Single Data Rate SDRAM

Single data rate SDRAM use a *single-edged clock* to synchronize all information; that is, all transmissions on the various busses (control, address, data) begin in time with one edge of the system clock (as so happens, the rising edge). Because the transmissions on the various busses are ideally valid from one clock edge to the next, those signals are very likely to be valid during the other edge of the clock (the falling edge). Consequently, that edge of the clock can be used to sample those signals.

SDRAMs have several features that were not present in earlier DRAM architectures: a burst length that is programmable and a CAS latency that is programmable.

Programmable Burst Length Like BEDO DRAMs, SDRAMs use the concept of *bursting* data to improve bandwidth. Instead of using successive toggling of the CAS signal to burst out data, however, SDRAM chips only require CAS to be signaled once and, in response, transmit or receive in time with the toggling of the clock the number of bits indicated by a value held in a programmable mode register. Once an SDRAM receives a row address and a column address, it will burst the number of columns that correspond to the burst length value stored in the register. If the mode register is programmed for a burst length of four, for example, then the DRAM will automatically burst four columns of contiguous data onto the bus. This eliminates the need to toggle the CAS to derive a burst of data in response to a microprocessor request. Consequently, the potential parallelism in the memory system increases (i.e., it improves) due to the reduced use of the command bus—the memory controller can issue other requests to other banks during those cycles that it otherwise would have been toggling CAS.

Programmable CAS Latency The mode register also stores the CAS latency of an SDRAM chip. Latency is a measure of delay. CAS latency, as the name implies, refers to the number of clock cycles it takes for the SDRAM to return the data once it receives a CAS command. The ability to set the CAS latency to a desired value allows parts of different generations fabricated in different process technologies (which would all otherwise have different performance characteristics) to behave identically. Thus, mixed-performance parts can easily be used in the same system and can even be integrated onto the same memory module.

Double Data Rate SDRAM

DDR SDRAMs have several features that were not present in single data rate SDRAM architectures: dual-edged clocking and an on-chip delay-locked loop (DLL).

Dual-Edged Clocking DDR SDRAMs, like regular SDRAMs, use a single-edged clock to synchronize control and address transmissions, but for data transmissions DDR DRAMs use a *dual-edged clock*; that is, some data bits are transmitted on the data bus in time with the rising edge of the system clock, and other bits are transmitted on the data bus in time with the falling edge of the system clock.

Figure 7.20 illustrates the difference, showing timing for two different clock arrangements. The top design is a more traditional arrangement wherein data is transferred only on the rising edge of the clock; the bottom design uses a data rate that is twice the speed of the top design, and data is transferred on both the rising and falling edges of the clock. IBM had built DRAMs using this feature in the late 1980s and presented their results in the International Solid-State Circuits Convention in February of 1990 [Kalter 1990a]. Reportedly, Digital Equipment Corp. had been experimenting with similar schemes in the late 1980s and early 1990s [Lee 2002, minutes of JC-42.3 meeting 58].

In a system that uses a single-edged clock to transfer data, there are two clock edges for every data "eye;" the data eye is framed on both ends by a clock edge, and a third clock edge is found somewhere in the middle of the data transmission (cf. Figure 7.20(a)). Thus, the clock signal can be used directly to perform two actions: to drive data onto the bus and to read data off the bus. Note that in a single-edged clocking scheme, data is transmitted once per clock cycle.

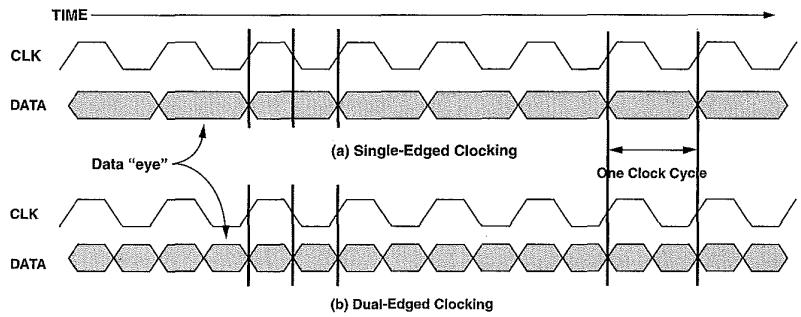


FIGURE 7.20: Running the bus clock at the data rate. The top diagram (a) illustrates a single-edged clocking scheme wherein the clock is twice the frequency of the data transmission. The bottom diagram (b) illustrates a dual-edged clocking scheme in which the data transmission rate is equal to the clock frequency.

By contrast, in a dual-edged clocking scheme, data is transmitted twice per clock cycle. This halves the number of clock edges available to drive data onto the bus and/or read data off the bus (cf. Figure 7.20(b)). The bottom diagram shows a clock running at the same rate as the data transmission. Note that there is only one clock edge for every data eye. The clock edges in a dual-edged scheme are either "edge-aligned" with the data or "center-aligned" with the data. This means that the clock can either drive the data onto the bus or read the data off the bus, but it cannot do both, as it can in a single-edged scheme. In the figure, the clock is edge-aligned with the data.

The dual-edged clocking scheme by definition has fewer clock edges per data transmission that can be used to synchronize or perform functions. This means that some other mechanism must be introduced to get accurate timing for both driving data and sampling data, i.e., to compensate for the fact that there are fewer clock edges, a dual-edged signaling scheme needs an additional mechanism beyond the clock. For example, DDR SDRAM specifies along with the system clock a center-aligned data strobe that is provided by the memory controller on DRAM writes that the DRAM uses directly to sample incoming data. On DRAM reads, the data strobe is edge-aligned with the data and system clock; the memory controller is responsible for providing its own mechanism for generating a center-aligned edge. The strobe is called DQS.

On-Chip Delay-Locked Loop In DDR SDRAM, the on-chip DLL synchronizes the DRAM's outgoing data and DQS (data strobe) signals with the memory controller's global clock [JEDEC Standard 21-C, Section 3.11.6.6]. The DLL synchronizes those signals involved in DRAM reads, not those involved in DRAM writes; in the latter case, the DQS signal accompanying data sent from the memory controller on DRAM writes is synchronized with that data by the memory controller and is used by the DRAM directly to sample the data [Lee 2002, Rhoden 2002, Karabotsos 2002, Baker 2002, Macri 2002]. The DLL circuit in a DDR DRAM thus ensures that data is transmitted by the DRAM in synchronization with the memory controller's clock signal so that the data arrives at the memory controller at expected times. The memory controller typically has two internal clocks: one in synch with the global clock, and another that is delayed 90° and used to sample data incoming from the DRAM. Because the DQS is in-phase with the data for read operations (unlike write operations), DQS cannot be used by the memory controller to sample the data directly. Instead, it is only used to ensure that the DRAM's outgoing DQS signal (and therefore data signals as well) is correctly aligned with the memory controller's clocks. The memory controller's 90°

delayed clock is used to sample the incoming data, which is possible because the DRAM's DLL guarantees minimal skew between the global clock and outgoing read data. The following paragraphs provide a bit of background to explain the presence of this circuit in DDR SDRAMs.

Because DRAMs are usually external to the microprocessor, DRAM designers must be aware of the issues involved in propagating signals between chips. In chip-to-chip communications, the main limiting factor in building high-speed interfaces is the variability in the amount of time it takes a signal to propagate to its destination (usually referred to as the *uncertainty* of the signal's timing). The total uncertainty in a system is often the sum of the uncertainty in its constituent parts, e.g., each driver, each delay line, each logic block in a critical path adds uncertainty to the total. This additive effect makes it very difficult to build high-speed interfaces for even small systems, because even very small fixed uncertainties that seem insignificant at low clock speeds become significant as the clock speed increases.

There exist numerous methods to decrease the uncertainty in a system, including sending a strobe signal along with the data (e.g., the DQS signal in DDR SDRAM or the clock signal in a source-synchronous interface), adding a phase-locked loop (PLL) or DLL to the system, or matching the path lengths of signal traces so that the signals all arrive at the destination at (about) the same time. Many of the methods are complementary; that is, their effect upon reducing uncertainty is cumulative. Building systems that communicate at high frequencies is all about engineering methods to reduce uncertainty in the system.

The function of a PLL or DLL, in general, is to synchronize two periodic signals so that a certain fixed amount of phase-shift or apparent delay exists between them. The two are similar, and the terms are often used interchangeably. A DLL uses variable delay circuitry to delay an existing periodic signal so that it is in synch with another signal; a PLL uses an oscillator to create a new periodic signal that is in synch with another signal. When a PLL/DLL is added to a communication interface, the result is a "closed-loop" system, which can, for example, measure and cancel the bulk of the uncertainty in both the transmitter and the receiver circuits and align the incoming data strobe with the incoming data (see, for example, Dally and Poulton [1998]).

Figure 7.21 shows how the DLL is used in DDR SDRAM, and it shows the effect that the DLL has upon the timing of the part. The net effect is to delay the output of the part (note that the output burst of the bottom configuration is shifted to the right, compared to the output burst of the top configuration). This delay is chosen to be sufficient to bring into alignment the output of the part with the system clock.

7.3.2 Other Technologies, Rambus in Particular

This section discusses Rambus' technology as described in their 1990 patent application number 07/510,898 (the '898 application) and describes how some of the technologies mentioned would be used in a Rambus-style memory organization as compared to a JEDEC-style memory organization.

Rambus' '898 Patent Application

The most novel aspect of the Rambus memory organization, the aspect most likely to attract the reader's attention and the aspect to which Rambus draws the most attention in the document, is the physical bus organization and operation. The bus' organization and protocol are more reminiscent of a computer network than a traditional memory bus. When the word *revolutionary* is used to describe the Rambus architecture, this is the aspect to which it applies.[8]

[8]There is also significant discussion (and illustration) in the patent application describing Rambus' unorthodox/revolutionary proposed packaging technology, in which a DRAM's pins are all on one edge of the chip and in which the chips are inserted directly into a backplane-type arrangement individually, as opposed to being integrated onto a memory module.

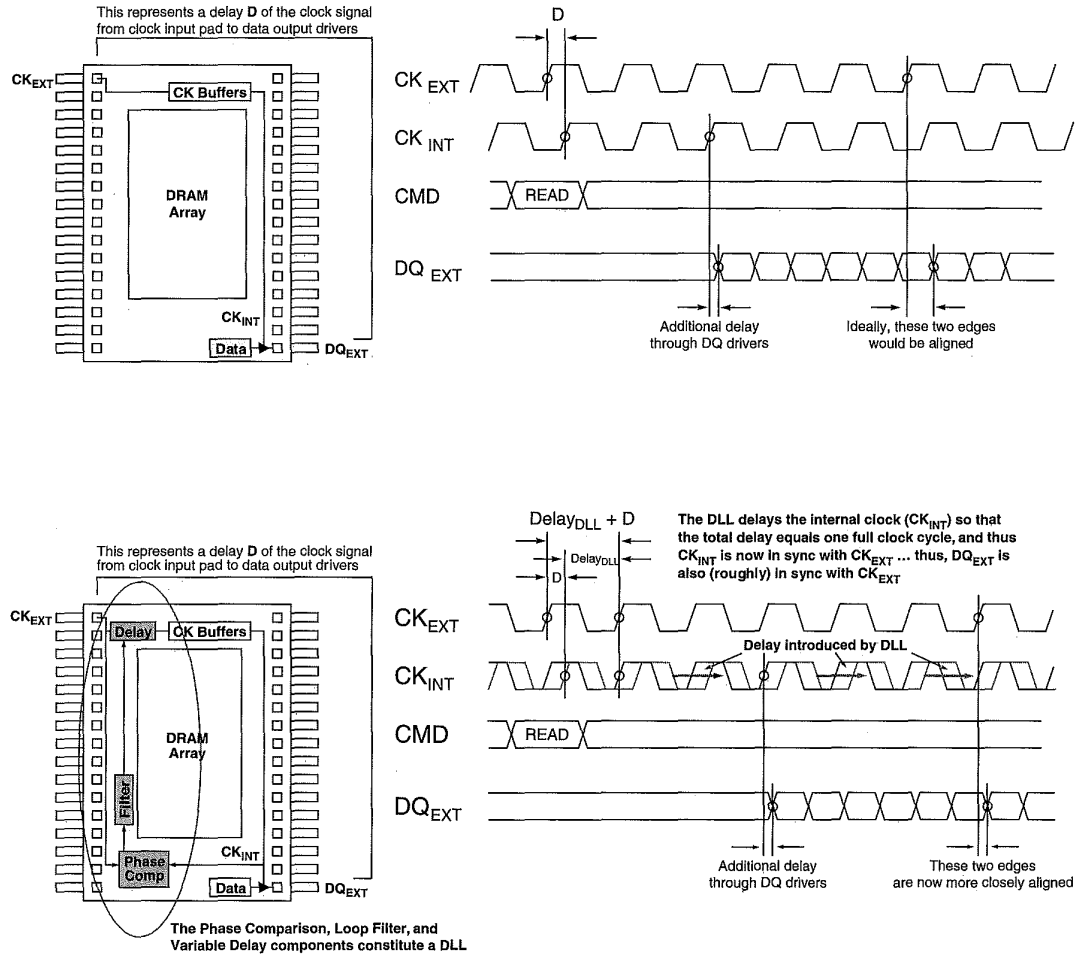
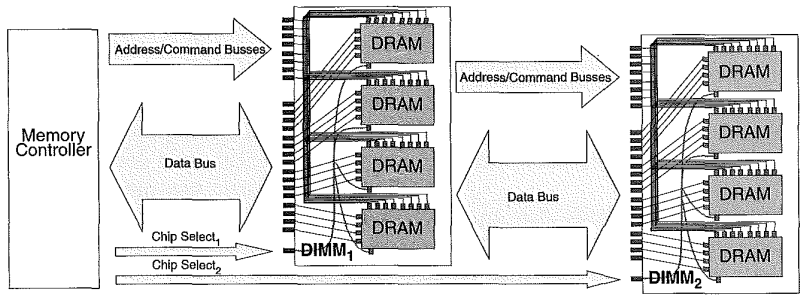
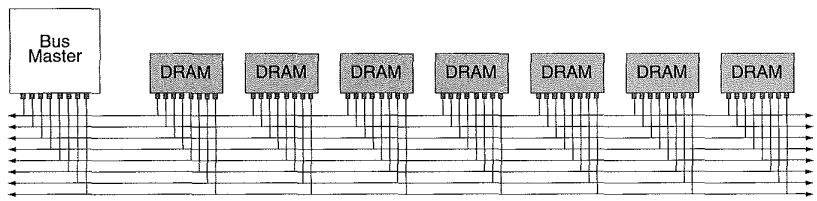


FIGURE 7.21: The use of the DLL in DDR SDRAMs. The top figure illustrates the behavior of an DDR SDRAM without a DLL. Due to the inherent delays through the clock receiver, multi-stage amplifiers, on-chip wires, output pads and bonding wires, output drivers, and other effects, the data output (as it appears from the perspective of the bus) occurs slightly delayed with respect to the system clock. The bottom figure illustrates the effect of adding the DLL. The DLL delays the incoming clock signal so that the output of the part is more closely aligned with the system clock. Note that this introduces extra latency into the behavior of the part.

Figure 7.22 illustrates the "new bus" that Rambus describes in the 898 application. It juxtaposes Rambus' bus with a more traditional DRAM bus and thereby illustrates that these bus architectures are substantially different. As described earlier, the traditional memory bus is organized into four dedicated busses: (1) the data bus, (2) the address bus, (3) the command bus, and (4) the chip-select bus. In contrast, all of the information



(a) Traditional Memory-Bus Organization: A "Wide-Bus" Architecture



(b) Rambus Memory-Bus Organization: A "Narrow-Bus" Architecture

FIGURE 7.22: Memory bus organizations. The figure compares the organizations of a traditional memory bus and a Rambus-style organization. (a) shows a system of a memory controller and two memory modules, with a 16-bit data bus and an 8-bit address and command bus. (b) shows the Rambus organization with a bus master and seven DRAM slave devices.

for the operation of the DRAM is carried over a single bus in Rambus' architecture. Moreover, there is no separate chip-select network. In the specification, Rambus describes a narrow bus architecture over which command, address, and data information travels using a proprietary packetized protocol. There are no dedicated busses in the Rambus architecture described in the '898 application. In the Rambus bus organization, all addresses, commands, data, and chip-select information are sent on the same bus lines. This is why the organization is often called "multiplexed." At different points in time, the same physical lines carry dissimilar classes of information.

Another novel aspect of the Rambus organization is its width. In Rambus' architecture, all of the information for the operation of the DRAM is carried over a very narrow bus. Whereas the bus in a traditional system can use more than 90 bus lines,[9] the Rambus organization uses "substantially fewer bus lines than the number of bits in a single address." Given that a single physical address in the early 1990s was 20–30 bits wide, this indicates a very narrow bus, indeed. In the Rambus specification, the example system uses a total of nine (9) lines to carry all necessary information, including addresses, commands, chip-select information, and data. Because the bus is narrower than a single data address, it takes many bus cycles to transmit a single command from a bus master (i.e., memory controller) to a DRAM device. The information is transmitted over an uninterrupted sequence of

[9]In a PC system, the data bus is 64 bits; the address bus can be 16 bits; and there can be 10 or more control lines, as the number of chip-select lines scales with the number of memory modules in the system.

bus cycles and must obey a specified format in terms of both time and wire assignments. This is why the Rambus protocol is called "packetized," and it stands in contrast to a JEDEC-style organization in which the command and address busses are wide enough to transmit all address and command information in a single bus cycle.[10]

As mentioned, the bus' protocol is also unusual and resembles a computer network more than a traditional memory bus. In an Internet-style computer network, for example, every packet contains the address of the recipient; a packet placed on the network is seen by every machine on the subnet, and every machine must look at the packet, decode the packet, and decide if the packet is destined for the machine itself or for some other machine. This requires every machine to have such decoding logic on board. In the Rambus memory organization, there is no chip-select network, and so there must be some other means to identify the recipient of a request packet. As in the computer network, a Rambus request packet contains (either explicitly or implicitly) the identity of the intended recipient, and every DRAM in the system has a unique identification number (each DRAM knows its own identity). As in the computer network, every DRAM must initially assume that a packet placed on the bus may be destined for it; every DRAM must receive and decode every packet placed on the bus so that each DRAM can decide whether the packet is for the DRAM itself or for some other device on the bus. Not only does each DRAM require this level of intelligence to decode packets and decide if a particular packet is intended for it, but ultimately the implication is that each DRAM is in some sense an autonomous device—the DRAM is not *controlled*; rather, *requests* are made of it.

This last point illustrates another sense in which Rambus is a revolutionary architecture. Traditional DRAMs were simply marionettes. The Rambus architecture casts the DRAM as a semi-intelligent device capable of making decisions (e.g., determining whether a requested address is within range), which represented an unorthodox way of thinking in the early 1990s.

Low-Skew Clock Using Variable Delay Circuits
The clocking scheme of the Rambus system is designed to synchronize the internal clocks of the DRAM devices with a non-existent, ideal clock source, achieving low skew. Figure 7.23 illustrates the scheme. The memory controller sends out a global clock signal that is either turned around or reflected back, and each DRAM as well as the memory controller has two clock inputs,

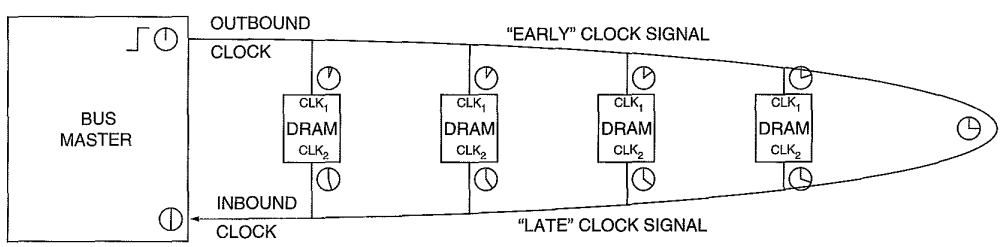


FIGURE 7.23: Rambus clock synchronization. The clock design in Rambus' 1990 patent application routes two clock signals into each DRAM device, and the path lengths of the clock signals are matched so that the delay average of the two signals represents the time as seen by the midpoint or clock turnaround point.

[10] A "bus cycle" is the period during which a bus transaction takes place, which may correspond to one clock cycle, more than one clock cycle, or less than one clock cycle.

CLK_1 and CLK_2, with the first being the "early" clock signal and the second being the "late" clock signal.

Because the signal paths between each DRAM have non-zero length, the global clock signal arrives at a slightly different time to each DRAM. Figure 7.23 shows this with small clock figures at each point representing the absolute time (as would be measured by the memory controller) that the clock pulse arrives at each point. This is one component contributing to clock skew, and clock skew traditionally causes problems for high-speed interfaces. However, if the clock path is symmetric, i.e., if each side of the clock's trace is path-length matched so that the distance to the turnaround point is equal from both CLK_1 and CLK_2 inputs, then the combination of the two clocks (CLK_1 and CLK_2) can be used to synthesize, at each device, a local clock edge that is in synch with an imaginary clock at the turnaround point. In Figure 7.23, the memory controller sends a clock edge at 12:00 noon. That edge arrives at the first DRAM at 12:03, the next DRAM at 12:06, the next at 12:09, and so on. It arrives at the turnaround point at 12:15 and begins to work its way back to the DRAM devices' CLK_2 inputs, finally arriving at the memory controller at 12:30. If at each point the device is able to find the average of the two clock arrival times (e.g., at the DRAM closest to the memory controller, find the average between 12:03 and 12:27), then each device is able to synthesize a clock that is synchronized with an ideal clock at the turnaround point; each device, including the memory controller, can synthesize a clock edge at 12:15, and so all devices can be "in synch" with an ideal clock generating an edge at 12:15. Note that even though Figure 7.23 shows the DRAMs evenly spaced with respect to one another, this is not necessary. All that is required is for the path length from CLK_1 to the turnaround point to be equal to the path length from CLK_2 to the turnaround point for each device.

Rambus' specification includes on-chip variable delay circuits (very similar to traditional DLLs) to perform this clock signal averaging. In other words, Rambus' on-chip "DLL" takes an early version and a late version of the same clock signal and finds the midpoint of the two signals. Provided that the wires making up the U-shaped clock on the motherboard (or wherever the wires are placed) are symmetric, this allows every DRAM in the system to have a clock signal that is synchronized with those of all other DRAMs.

Variable Request Latency Rambus' 1990 patent application defines a mechanism that allows the memory controller to specify how long the DRAM should wait before handling a request. There are two parts to the description found in the specification. First, on each DRAM there is a set of registers, called *access-time registers*, that hold delay values. The DRAM uses these delay values to wait the indicated number of cycles before placing the requested data onto (or reading the associated data from) the system bus. The second part of the description is that the DRAM request packet specifies which of these registers to use for a delay value in responding to that particular request.

The patent application does not delve into the uses of this feature at all; the mechanism is presented simply, without justification or application. Our (educated) guess is that the mechanism is absolutely essential to the successful operation of the design—having a variable request latency is not an afterthought or flight of whimsy. Without variable request latencies, the support of variable burst lengths, and indeed the support of any burst length other than one equal to the length of a request packet or one smaller than the request latency, cannot function even tolerably well. This is illustrated in Figure 7.24. The figure shows what happens when a multiplexed, split transaction bus has back-to-back requests: if the request shapes are symmetric (e.g., the request and response packets are the same length, and the latency is slightly longer) or if the latency is long relative to the request and response packet lengths, then it is possible to pipeline requests and achieve good throughput, though neither scenario is optimal in bus efficiency. If the request packet and transaction latency are both short relative to the data transfer (a more optimal arrangement for a given request), later requests must be delayed until earlier requests finish, negating the value of having a split transaction bus in the first place (cf. Figure 7.24(a)). This is the most likely arrangement. Long data bursts are desirable, particularly if the target application is video processing or something similar. The potential problem is that these long data bursts necessarily generate asymmetric

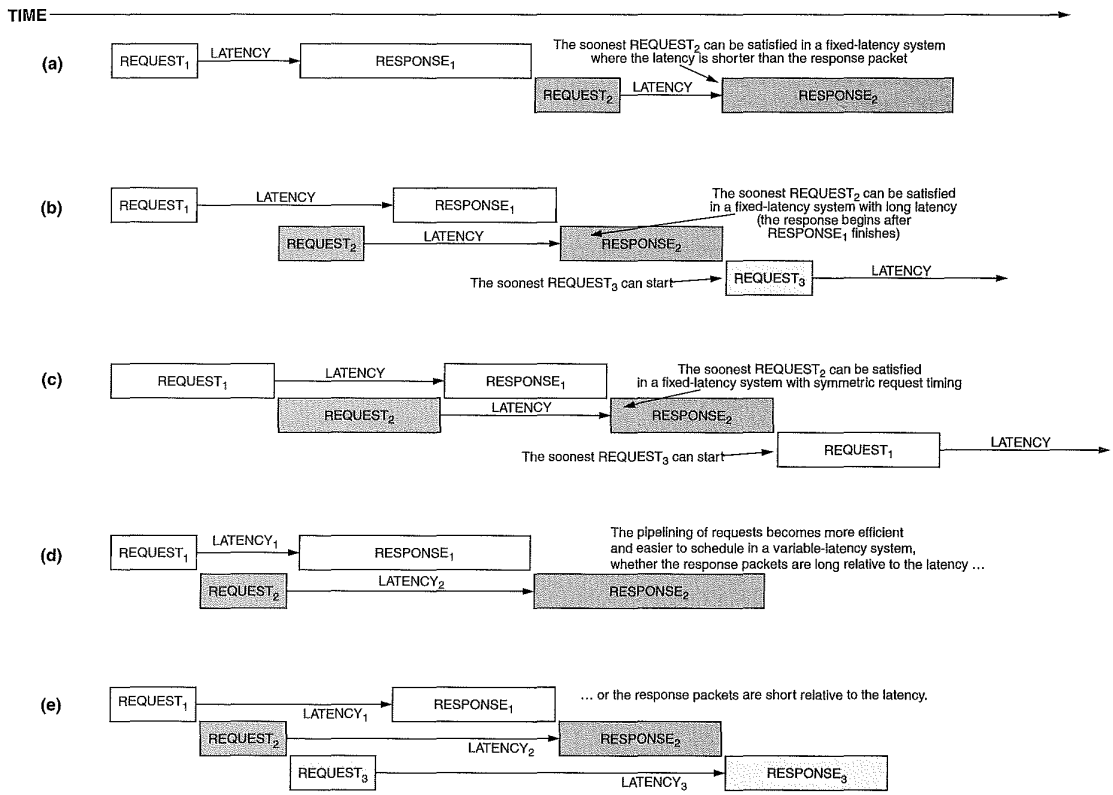


FIGURE 7.24: The importance of variable request latencies in the Rambus organization. Scheduling odd-sized request-response pairs on a multiplexed bus can be difficult to the point of yielding unacceptable performance.

request-response shapes because the request packet should be as short as possible, dictated by the information in a request. If the DRAM supports variable request latencies, then the memory controller can pipeline requests, even those that have asymmetric shapes, and thus achieve good throughput despite the shape of the request.

Variable Block Size Rambus' 1990 patent application defines a mechanism that allows the memory controller to specify how much data should be transferred for a read or write request. The request packet that the memory controller sends to the DRAM device specifies the data length in the request packet's *BlockSize* field. Possible data length values range from 0 bytes to 1024 bytes (1 KB). The amount of data that the DRAM sends out on the bus, therefore, is programmed at every transaction.

Like variable request latency, the variable block size feature is necessitated by the design of the bus. To dynamically change the transaction length from request to request would likely have seemed novel to an engineer in the early 1990s. The unique features of Rambus' bus—its narrow width, multiplexed nature, and packet request protocol—place unique scheduling demands on the bus. Variable block size is used to

cope with the unique scheduling demands; it enables the use of Rambus' memory in many different engineering settings, and it helps to ensure that Rambus' bus is fully utilized.

Running the Clock at the Data Rate The design specifies a clock rate that can be half what one would normally expect in a simple, non-interleaved memory system. Figure 7.25 illustrates this, showing timing for two different clock arrangements. The top design is a more traditional arrangement; the bottom design uses a clock that is half the speed of the top design. Rambus' 1990 application describes the clock design and its benefits as follows:

> *Clock distribution problems can be further reduced by using a bus clock and device clock rate equal to the bus cycle data rate divided by two, that is, the bus clock period is twice the bus cycle period. Thus a 500 MHz bus preferably uses a 250 MHz clock rate. This reduction in frequency provides two benefits. First it makes all signals on the bus have the same worst case data rates—data on a 500 MHz bus can only change every 2 ns. Second, clocking at half the bus cycle data rate makes the labeling of the odd and even bus cycles trivial, for example, by defining even cycles to be those when the internal device clock is 0 and odd cycles when the internal device clock is 1.*

As the inventors claim, the primary reason for doing so is to reduce the number of clock transitions per second to be equal to the maximum number of data transitions per second. This becomes important as clock speeds increase, which is presumably why IBM's toggle mode uses the same technique. The second reason given is to simplify the decision of which edge to use to activate which receiver or driver (the labelling of "even/odd" cycles). Figure 10 in the specification illustrates the interleaved physical arrangement required to implement the clock-halving scheme: the two edges of the clock activate different input receivers at different times and cause different output data to be multiplexed to the output drivers at different times.

Note that halving the clock complicates receiving data at the DRAM end, because no clock edge exists during the eye of the data (this is noted in the figure), and therefore, the DRAM does not know when to sample the incoming data—that is, assuming no additional help. This additional help would most likely be in the form of a PLL or DLL circuit to accurately delay the clock edge so that it would be 90° out of phase with the data and thus could be used to sample the data. Such a circuit would add complexity to the DRAM and would consequently add cost in terms of manufacturing, testing, and power dissipation.

7.3.3 Comparison of Technologies in Rambus and JEDEC DRAM

The following paragraphs compare briefly, for each of four technologies, how the technology is used in a JEDEC-style DRAM system and how it is used in a Rambus-style memory system as described in the '898 application.

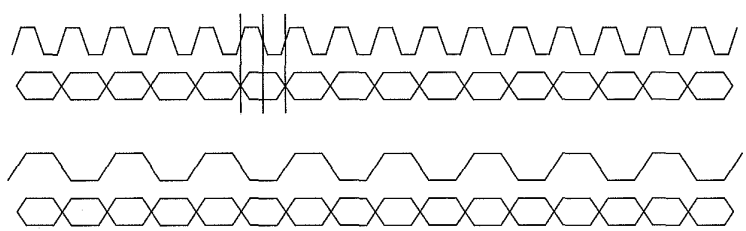


FIGURE 7.25: Running the bus clock at the data rate. Slowing down the clock so that it runs at the same speed as the data makes the clock easier to handle, but it reduces the number of clock edges available to do work. Clock edges exist to drive the output circuitry, but no clock edge exists during the eye of the data to sample the data.

Programmable CAS Latency

JEDEC's *programmable CAS latency* is used to allow each system vendor to optimize the performance of its systems. It is programmed at system initialization, and, according to industry designers, it is never set again while the machine is running [Lee 2002, Baker 2002, Kellogg 2002, Macri 2002, Ryan 2002, Rhoden 2002, Sussman 2002]. By contrast, with Rambus' *variable request latency*, the latency is programmed every time the microprocessor sends a new request to the DRAM, but the specification also leaves open the possibility that each access register could store two or more values held for each transaction type. Rambus' system has the potential (and we would argue the *need*) to change the latency at a request granularity, i.e., each request could specify a different latency than the previous request, and the specification has room for many different latency values to be programmed. Whereas the feature is a convenience in the JEDEC organization, it is a necessity in a Rambus organization.

Programmable Burst Length

JEDEC's *programmable burst length* is used to allow each system vendor to optimize the performance of its systems. It is programmed at system initialization, and, according to industry designers, it is never set again while the machine is running [Lee 2002, Baker 2002, Kellogg 2002, Rhoden 2002, Sussman 2002]. By contrast, with Rambus' *variable block size*, the block size is programmed every time the microprocessor sends a new request to the DRAM. Whereas a JEDEC-style memory system can function efficiently if every column of data that is read out of or written into the DRAM is accompanied by a CAS signal, a Rambus-style memory system could not (as the command would occupy the same bus as the data, limiting data to less than half of the available bus cycles). Whereas the feature is a convenience in the JEDEC organization, it is a necessity in a Rambus organization.

Dual-Edged Clocking

Both JEDEC- and Rambus-style DRAMs use a clocking scheme that goes back at least to IBM's high-speed toggle-mode DRAMs, in which on-chip interleaving allows one to toggle back and forth between two buffers (e.g., on the rising and falling edges of a strobe signal) to achieve a data rate that is twice that possible by a single buffer alone. Both JEDEC- and Rambus-style DRAMs transfer data on both edges of a timing signal. In JEDEC-style DRAMs, the timing signal is an internal clock generated from the system clock and the DQS data strobe—a source-synchronous strobe that accompanies the data sent and is quiescent when there is no data on the bus. The data is edge-aligned with the system clock. In Rambus-style DRAMs, the timing signal is a synthesized internal clock signal in synch with no other clock source in the system and generated from two different phases of the U-shaped global clock (described previously). The U-shaped clock is not source synchronous and remains free running whether there is data on the bus or not. As opposed to the DDR clocking scheme, in the Rambus scheme the data is *not* aligned with either phase of the system clock at all; it is neither edge-aligned nor center-aligned. Furthermore, a different phase relationship exists between each Rambus DRAM's output data and the global clock's early and late signals, whereas DDR DRAMs strive to maintain the same phase relationship between each DRAM's output data and the system clock.

On-Chip PLL/DLL

JEDEC uses an *on-chip DLL* in their DDR SDRAMs to ensure that data being driven onto the data bus is aligned with the global clock signal. The DLL does this by delaying the DRAM's response to a read request just long enough that the data is driven at the same time the DRAM sees the next clock edge. Rambus uses an *on-chip variable delay* circuit to ensure that every DRAM in the system as well as the memory controller has a synchronized clock (i.e., they all believe that it is precisely 12:00 at the same time). The delay circuit does this by finding the midpoint in time between an early version and a late version (i.e., two different phases) of the same clock signal and thereby creates a synthesized internal clock signal in synch with no other physical clock in the system. This is a process that is significantly more complicated than a simple DLL, and thus Rambus' variable delay circuit is more complex than a simple DLL.

7.3.4 Alternative Technologies

Numerous alternative technologies could achieve the same result as the technologies described. Many of these alternatives are simply applications of long-understood techniques to solving particular problems. This section discusses a sample of those mechanisms; it is not intended to be exhaustive, but rather to be illustrative. This should give insight into issues that DRAM designers consider.

Programmable CAS Latency

The primary benefit of programmable CAS latency is flexibility. Its presence in a DRAM allows a fast part to emulate the behavior of a slow part, which would enable an OEM or end consumer to intermingle parts from different generations (with different speeds) in the same system or intermingle same-generation parts from different manufacturers (which might have slightly different performance capabilities due to slight variations in process technologies) in the same DIMM. To illustrate, if a DRAM manufacturer has a part with a minimum 20-ns CAS read-out, the part could conceivably work as a CAS-2, 100-MHz part (20 ns requires 2 cycles at 100 MHz); a CAS-3, 150-MHz part (20 ns requires 3 cycles at 150 MHz); or a CAS-4, 200-MHz part (20 ns requires 4 cycles at 200 MHz). Note that the part would never be able to make CAS-2 latencies at the higher bus speeds or CAS-3 at the highest bus speed (CAS-2 requires 13.33 ns at 150 MHz and 10 ns at 200 MHz; CAS-3 requires 15 ns at 200 MHz). Alternative technologies include the following:

- Use fixed CAS latency parts.
- Explicitly identify the CAS latency in the read or write command.
- Program CAS latency by blowing fuses on the DRAM.
- Scale CAS latency with clock frequency.

Use Fixed CAS Latency Fixing the CAS latency would simply mean that each individual SDRAM would operate with a single, predetermined latency. Although using fixed latency would reduce flexibility, it would simplify testing considerably [Lee 2002], as only one mode of operation would need to be tested instead of two or more, and it would simplify design slightly. In a JEDEC-style bus organization, where the command, address, and data busses are all separate, the scheduling of requests is simplified if all requests have the same latency. Thus, memory controllers typically set the CAS latency at initialization and never change it again while the machine is powered on; i.e., current systems use a *de facto* fixed CAS latency while the machine is running [Lee 2002, Baker 2002, Kellogg 2002, Macri 2002, Ryan 2002, Rhoden 2002, Sussman 2002].

Explicitly Identify CAS Latency in the Command One option would be to design the DRAM command set so that each *column read* command explicitly encodes the desired CAS latency. Instead of initializing the DRAM to use a latency of 2 or 3 cycles, and then presenting generic CAS commands, each of which would implicitly use a latency of 2 or 3 cycles, respectively, a memory controller could present commands to the SDRAM that would explicitly specify a desired latency. For example, the memory controller would use "CAS-2" read commands if it desired a latency of two and "CAS-3" read commands if it desired a latency of three.

Program CAS Latency Using Fuses The CAS latency value could also be determined by the use of on-chip fuses. Modern DRAMs have numerous fuses on them that, when blown, enable redundant circuits, so at test time a DRAM manufacturer can, for example, disable memory arrays that contain fabrication errors and in their place enable redundant arrays that are error free. A typical DRAM has thousands of such fuses. One could add one or more similar fuses that set different CAS latencies. This would enable a DRAM manufacturer to sell what are essentially fixed-latency parts, but because they could be programmed at the last possible moment, the DRAM manufacturer would not need separate stocks for parts with different latencies; one part would satisfy for multiple latencies, as the DRAM manufacturer could set the fuses appropriately right before shipping the parts. The manufacturer could also sell the parts with fuses intact to OEMs and embedded-systems manufacturers, who would likely have the technical savvy necessary to set the fuses properly, and who might want the

ability to program the latency according to their own needs. This level of flexibility would be almost on par with the current level of flexibility, since in most systems the CAS latency is set once at system initialization and never set again.

Scale CAS Latency with Clock Frequency One option is for the DRAM to know the bus speed and deliver the fastest CAS latency available for that bus speed. The programming of the DRAM could be either explicit, in which case the memory controller tells the DRAM the bus speed using a command at system initialization, or implicit, in which case the DRAM senses the timing of the clock on the bus and "learns" the bus speed on its own. To return to the earlier example, if a DRAM part had an internal CAS read-out latency of 20 ns, it could deliver CAS-2 in a 100-MHz/10-ns bus environment, CAS-3 in a 133-MHz/7.5-ns bus environment, and CAS-4 in a 200-MHz/5-ns bus environment. This would satisfy most DRAM consumers, because OEMs and end-users usually want the fastest latency possible. However, without an additional mechanism in either the DRAM or the memory controller, the scheme would not allow parts with different internal CAS read-out latencies to be mixed within the same system. Disallowing mixed-latency systems would not cause undue confusion in the marketplace, because asynchronous parts in the 1980s and 1990s were sold as "70ns DRAMs" or "60ns DRAMs," etc.; the speed rating was explicit, so both OEMs and end-users were already accustomed to matching speed ratings in their systems. However, if support for mixed-latency systems was desired, one could always design the memory controller so that it determines at initialization the latency of each DIMM in its system[11] and accounts for any latency differences between DIMMs in its scheduling decisions.

Programmable Burst Length

In JEDEC-style DRAM systems, the ability to set the burst length to different values is a convenience for system designers that can also system performance. Numerous subtle interactions between the chosen parameters of a DRAM system (such as request latency, burst length, bus organization, bus speed, and bus width) can cause the resulting performance of that system to vary significantly, even for small changes in those parameters [Cuppu & Jacob 2001]. The ability to fine-tune the burst length of the DRAMs enables a designer to find better combinations of parameters for his system, given the expected workload. In most systems, the burst length is set once at system initialization and never set again [Lee 2002, Baker 2002, Kellogg 2002, Rhoden 2002, Sussman 2002]. Alternative technologies include the following:

- Use a short, fixed burst length.
- Explicitly identify the burst length in the read or write command.
- Program the burst length by blowing fuses on the DRAM.
- Use a long, fixed burst length coupled with the *burst-terminate* command.
- Use a BEDO-style protocol where each CAS pulse toggles out a single column of data.

The first three are analogous to the alternative solutions for programmable CAS latency, described above.

Use Burst-Terminate Command One can use a fixed burst length, where the length is some number large enough to satisfy developers of systems that prefer large chunks of data, and use the *burst-terminate* command, also called *burst stop*, to halt data read-out of the DRAM or to signal the end of data written to the DRAM. The advantage of the scheme is that the *burst stop* command is already part of the specification, and bursts must also be interruptible by additional commands such as reads and writes. Therefore, an efficient pipeline of multiple column reads is already possible by simply sending multiple read commands spaced as closely as the desired burst length dictates (e.g., every four cycles to achieve a burst length of

[11] One would probably not want to create a DIMM out of DRAM parts with different fixed CAS latencies.

four, even if the nominal burst length is longer). Each successive request would implicitly terminate the request immediately preceding it, and only the last burst in the pipeline would need to be terminated explicitly. The disadvantage of the scheme is that it would increase pressure slightly on the command bus: during a *burst-terminate* command the memory controller would not be able to control any other bank on the same command bus. If the increase was determined to be significant, it could be alleviated by redesigning the memory controller to support multiple control busses (which is already supported by memory controllers).

Toggle Data-Out Using CAS JEDEC could have adopted the BEDO style of bursting data, with each successive column read-out driven by toggling the CAS pin or holding it at a low voltage until the desired number of columns has been read out (e.g., holding it low for four cycles to read out four columns of data). This would require some modification to the specification, enabling the DRAM to distinguish between a CAS accompanied by a new column address and a CAS signifying a serial read-out based on the most recent column address received. There are numerous possibilities for solving this, including the following:

- Redefine the command set to have two different CAS commands: one for a new address, and one signifying a sequential read-out based on the most recent column address received.
- Add a pin to the command bus (which is usually thought of as comprising signals such as RAS, CAS, WE, and CKE) that is dedicated for the sequential read-out version of CAS. This would be similar to IBM's implementation of the high-speed toggle mode, where the data strobe signal uses a different pin than the CAS signal [Kalter 1990a, b].

The advantage of the second scheme, as compared to the first, is that it allows the memory controller to control other DRAM banks simultaneously while the first bank is involved in data transfer.

Dual-Edged Clocking

The advantage of using dual-edged clocking in a JEDEC-style DRAM bus organization is that it increases DRAM bandwidth without having to drive the clock any faster and without a commensurate increase in the clock signal's energy consumption. Alternative technologies include the following:

- Use two or more interleaved memory banks on-chip and assign a different clock signal to each bank (e.g., use two or more out-of-phase clocks).
- Keep each DRAM single data rate, and interleave banks on the module (DIMM).
- Increase the number of pins per DRAM.
- Increase the number of pins per module.
- Double the clock frequency.
- Use simultaneous bidirectional I/O drivers.

Interleave On-Chip Banks As mentioned earlier, interleaving multiple memory banks has been a popular method used to achieve high-bandwidth memory busses using low-bandwidth devices. The technique goes back at least to the mid-1960s, where it was used in the IBM System/360 Model 91 [Anderson et al. 1967] and Seymour Cray's Control Data 6600 [Thornton 1970]. In an interleaved memory system, the data bus uses a frequency that is faster than any one DRAM bank can support; the control circuitry toggles back and forth between multiple banks to achieve this data rate. An alternative to using dual-edged clocking (e.g., to double or triple or even quadruple the memory bandwidth of SDRAM without using both edges of the clock to send/receive data) is to specify two, three, or four independent banks per DRAM (respectively) and assign each bank its own clock signal. The memory controller would send one request to the DRAM, and that request would be handed off to each bank in synch with the clock assigned to that bank. Thus, each bank would receive its request slightly advanced or delayed with respect to the other banks. There are two ways to create the different clock signals:

- The different clock signals driving each of the different banks could be generated by

the memory controller (or any entity outside the DRAM). Thus, if there were two interleaved banks, the memory controller would send two clock signals; if there were four interleaved banks, the memory controller would send four clock signals; and so on.

- The DRAM could receive one clock signal and delay and distribute it internally to the various banks on its own. Thus, if there were two interleaved banks, the DRAM would split the incoming clock signal two ways, delaying the second a half-phase; if there were four interleaved banks, the DRAM would split the incoming clock signal four ways, delaying the second clock one-quarter of a phase, delaying the third clock one-half of a phase, and delaying the last clock three-quarters of a phase; and so on.

The first implementation would require extra DRAM pins (one CK pin for each bank); the latter would require on-DRAM clock generation and synchronization logic.

Interleave Banks on the Module Instead of increasing the bandwidth of individual DRAMs, an argument could be made that the only place it matters is at the DIMM level. Therefore, one could take SDRAM parts and create a DDR DIMM specification where on-module circuitry takes a single incoming clock and interleaves two or more banks of DRAM (transparently, as far as the memory controller is concerned). Kentron [2002] shows that this is certainly within our limits, even at very high data rates; they currently take DDR parts and interleave them transparently at the module level to achieve quadruple data rate DIMMs.

Increase DRAM Data Width To achieve higher bandwidth per DRAM, the trend in recent years has been not only to increase DRAM speed, but also to increase the number of data-out pins; x32 parts are now common. Every increase from x4 to x8 to x16 and so on doubles the DRAM's data rate. Doubling the data rate by doubling the data width of existing parts requires very little engineering know-how. It only requires the addition of more pins and doubling the width of the data bus to each individual DRAM. Note that the number of data pins has increased from x1 parts in the late 1980s to x32 parts today [Przybylski 1996], while over the same time period the data rate has increased from 16 MHz to the 400-MHz clocks found in DDR SDRAM today. JEDEC, therefore, could have simply put their weight more firmly behind this trend and increased bandwidth by increasing pin-out. The advantage of this approach is that it combines very well with the previous alternative—interleaving multiple banks at the module level—because doubling the data width of a DRAM decreases the number of DRAM parts needed to achieve the DIMM data width, effectively doubling the number of independent banks one can put on the DIMM. If the DRAM pin-out increases, then fewer DRAMs would be needed per DIMM to create the 64-bit data bus standard in PC-compatible systems. This would leave extra room on the DIMM for more DRAMs that could make up extra banks to implement an interleaved system.

Increase Module Data Width As mentioned earlier, one could argue that the DIMM bandwidth is more important than the bandwidth of an individual DRAM. Therefore, one could simply increase the data width of the memory bus (the number of wires between the DIMMs and the memory controller) to increase bandwidth, without having to increase the clock speed of the individual DRAMs at all. The disadvantage is the increased cost of the de-skewing circuitry.

Double the Clock Frequency An alternative to using a dual-edged clock is to use a single-edged clock and simply speed up the clock. There are several advantages of a single-edged clocking scheme over a dual-edged clocking scheme (cf. Figure 7.20). One advantage illustrated in Figure 7.20 is the existence of more clock edges to be used to drive data onto the bus and sample data off the bus. Another advantage is that the clock does not need to be as symmetric as it is in a dual-edged clocking scheme. A single-edged clock's duty cycle does not need to be 50%, as it needs to be for a dual-edged clock, and the rise and fall times (i.e., slew rates) do not need to match, as they need to for a dual-edged clock. Consequently, one

can achieve the same speed clock in a single-edged clocking scheme with much less effort than in a dual-edged clocking scheme. The disadvantage of this alternative is that it requires more engineering effort than simply widening the memory bus or increasing the number of data pins on a DRAM.

Use Simultaneous Bidirectional I/O Running the data at the same rate as the clock doubles a DRAM's bandwidth over previous, single-edged clock designs without an increase in the number of data pins. An alternative is to use simultaneous bidirectional I/O, in which it is possible to conduct a read from the DRAM and a write to the DRAM at exactly the same time; that is, both data values (the read data and the write data) are on the bus simultaneously, which is an effective doubling of the available bandwidth. Using the simultaneous bidirectional I/O does not require additional data pins. Such a scheme would require structural changes at the DRAM side to accommodate reading and writing simultaneously, and those changes would most likely resemble the ESDRAM design by Enhanced Memory Systems [ESDRAM 1998], which was proposed for DDR2 [Davis et al. 2000b]. ESDRAM places a buffer after the sense amps that holds an entire DRAM row for reading and allows concurrent writes to the DRAM array; once the read data is in the buffer, the sense amplifiers are no longer needed.

On-Chip PLL/DLL

DDR SDRAMs use an on-chip DLL circuit to ensure that the DRAM transmits its data and DQS signal to the memory controller as closely as possible to the next appropriate edge of the system clock. Its use is specified because the clock rates in DDR are high enough to warrant relatively strong methods for reducing the effects of dynamic changes in timing skew. Alternative technologies include the following:

- Achieve high bandwidth using more DRAM pins or module pins, not clock frequency.
- Use a Vernier method to measure and account for dynamic changes in skew.
- Put the DLL on the memory controller.
- Use off-chip (on-module) DLLs.
- Use asynchronous DRAM, for example, toggle mode or BEDO.

Go Wider, Not Faster As the previous section indicates, there are numerous ways to achieve higher bandwidth at the DRAM, DIMM, or memory-system level, and many of the methods that achieve higher bandwidth are easier to implement than increasing clock speed. The use of a DLL is dictated only by the desired increase in clock speed. Therefore, one can forgo the use of an on-chip DLL by increasing DRAM or DIMM data width.

Vernier Mechanism There are two main components to the uncertainty of signal propagation time. The first is a static component that is due to differences in process technologies, process variations, electrical loading of an unknown number of memory modules, etc. The second is a dynamic component that is typically due to temperature or voltage fluctuations. Some fluctuations change too fast to be handled effectively, but most of the fluctuations in voltage and temperature change over a relatively long time period. Most voltage fluctuations are associated with the 60-Hz AC power cycle, which is a relatively long time-span, and most temperature fluctuations occur on the order of milliseconds or longer [Lee 2002, Macri 2002]. In many DDR systems, the static component is corrected by using a Vernier mechanism in the memory controller at system initialization. It is essentially a trial-and-error method in which the upper and lower limits of timing failure are found and the midpoint is used for signal transmission (analogous to setting the tracking on one's VCR by going forward and backward to the points of noise and then leaving the tracking set somewhere in between) [Lee 2002]. The dynamic component is then corrected by the on-chip DLL.

An alternative to the on-chip DLL is to perform recalibration of the Vernier correction often enough to account for the slower changes in voltage and temperature, as opposed to performing the timing correction only once at system initialization. Assuming that the most significant voltage fluctuations are associated with the 60-Hz AC power supply and

that most of the temperature fluctuations occur on the order of milliseconds or longer [Lee 2002, Macri 2002], recalibration would be needed roughly once or twice every millisecond.

Move the DLL onto the Memory Controller Because the DLL is only used on the DRAM to synchronize outgoing data and DQS signals with the global clock for the benefit of the memory controller [Lee 2002, Rhoden 2002, Karabotsos 2002, Baker 2002, Macri 2002], it is possible to move that functionality onto the memory controller itself. The memory controller could maintain two clocks: the first, for example, could be synchronized with the global clock, and the second could be delayed 90°. The incoming DQS and data signals could be given a variable delay, with the amount of delay controlled by a DLL/PLL on the memory controller so that the DQS signal would be in-phase with the delayed clock. This arrangement could align the incoming data so that the eye would be centered on the global clock signal, which could be used to sample the data. The weakness of this scheme is that, as clock speeds increase to very high rates, the timing differences between different DIMMs would become significant. Therefore, each DIMM in the system would require a different phase shift between the memory controller's two clocks, which would imply that the memory controller would need to maintain a separate timing structure for each DIMM.[12]

Move the DLL onto the DIMM One alternative is to place the DLL on the DDR module instead of the DDR device itself. Sun has used this alternative for many years [Becker 2002, O'Donnell 2002, Prein 2002, Walker 2002]. Moreover, it is similar in nature to the off-chip driver mechanism used by IBM in their high-speed toggle-mode DRAM [Kalter 1990a, b]. As mentioned previously, the only function of the DLL on the DDR SDRAM is to synchronize the outgoing DQS and data signals with the global clock. This can be accomplished just as easily on the module, especially if the module is *buffered* or *registered* (which simply means that the module has local storage to hold the commands and addresses that are destined for the module's DRAMs). Note that it is presently common practice for engineers to disable DDR's on-chip DLL to achieve higher performance—the on-chip DLL is a convenience, not a necessity [Rhoden 2002, Kellogg 2002, Macri 2002]. A module-level DLL is perfectly workable; even if the data exiting the DRAM is completely out of synch with the global clock, the module can use its DLL to delay the clock/s, commands, and addresses so that the output of the DRAMs is in synch with the global clock. However, this might come at the expense of an extra CAS latency cycle.

7.4 Fully Buffered DIMM: A Compromise of Sorts

An interesting development in the DRAM industry is the recent introduction of the fully buffered DIMM (FB-DIMM). This is a system-level DRAM organization developed by Intel that, at first glance, looks a bit like a compromise between a traditional JEDEC wide-bus design and a Rambus-like narrow-bus design. Chapter 14 explores the FB-DIMM in more detail. Here, we will simply introduce it and present a motivation for its development.

An important trend facing system designers ultimately argues for FB-DIMM or something like it. As DRAM technology has advanced, the channel speed has improved at the expense of channel capacity. SDR SDRAMs could populate a channel with eight DIMMs. DDR SDRAMs operate at a higher data rate and limit the number of DIMMs to four per channel. DDR2 SDRAMs operate at an even higher data rate and limit the number of DIMMs to two per channel. DDR3 is expected to limit the number of DIMMs per channel to one. This is a significant limitation;

[12]Most DLLs/PLLs require many cycles to lock onto a signal and create the correct phase shift. Forcing the memory controller to wait many cycles before switching from reading from one DIMM to reading from another DIMM would be an unsatisfactory situation.

for a server designer, it is critical, as servers typically depend on memory capacity for their performance.

The trend is clear: to improve data rates, one must reduce the number of drops on the bus. The designers of graphics subsystems have known this for a long time—for every DRAM generation, graphics cards use the same DRAM technology as is found in commodity DIMMs, but the graphics cards operate their DRAMs at significantly higher data rates because they use point-to-point connections. Note that this is possible in a marketing sense because graphics cards are *de facto* embedded systems. Users do not expect to upgrade their graphics cards by adding more memory to them; they instead replace the entire card.

So there is an obvious dilemma: future designers would like both increased channel capacity and increased data rates. How can one provide both?

The relationship between a modular organization and the graphics-card organization is shown in Figure 7.26, which depicts a multi-drop DRAM bus next to a DRAM bus organization typical of graphics cards. The graphics-card organization uses the same DRAM technology as in the multi-drop organization, but it also uses point-to-point soldered connections between the DRAM and memory controller and can therefore reach higher rates of speed. How can one exploit this to increase data rates without sacrificing channel capacity? One solution is to redefine

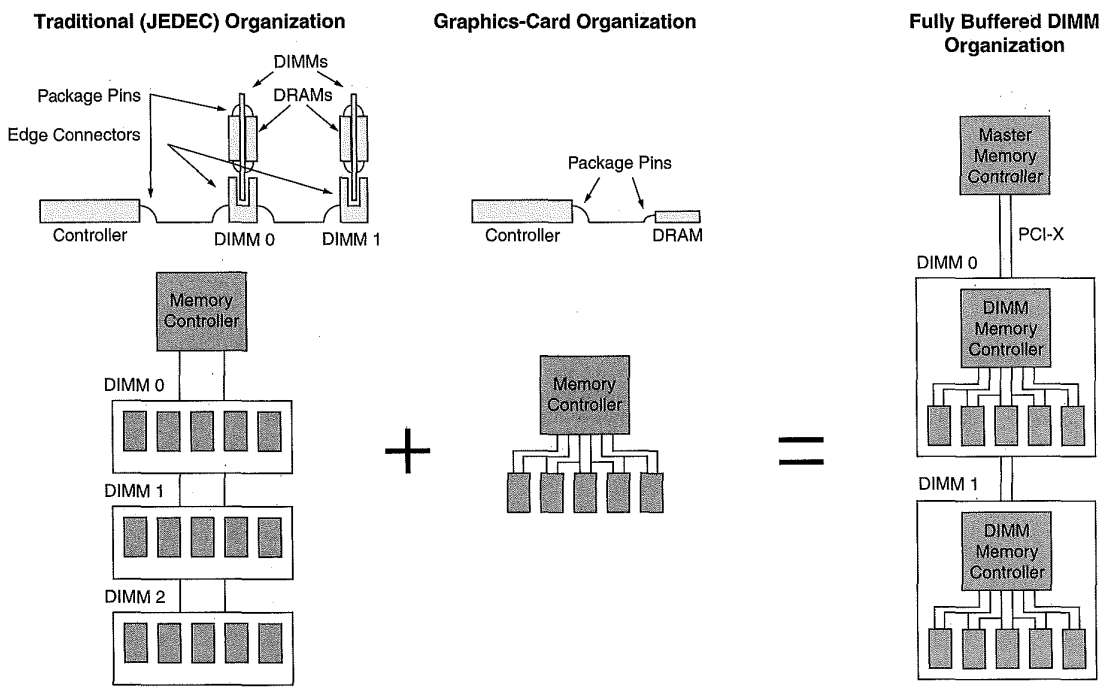


FIGURE 7.26: Motivation for the design of FB-DIMM. The first two organizations compare a modular wide-bus and a high-performance embedded system such as a graphics card. Above each design is its side profile, indicating potential impedance mismatches (sources of relections). The organization on the far right shows how FB-DIMM takes the graphics-card organization as its *de facto* DIMM. In the FB-DIMM organization, there are no multi-drop busses; DIMM-to-DIMM connections are point-to-point.

one's terms—to call the graphic-card arrangement a "DIMM" in one's future system. This is shown on the right-hand side of the figure and is precisely what is happening in FB-DIMM: the memory controller has been moved onto the DIMM, and all connections in the system are point-to-point. The channel connecting the master memory controller to the DIMM-level memory controllers (called *Advanced Memory Buffers*, or *AMBs*) is very narrow and very fast, based on the PCI-X standard. Moreover, each DIMM-to-DIMM connection is a point-to-point connection, making an entire channel a *de facto* multi-hop store and forward network. The FB-DIMM architecture limits the channel length to eight DIMMs, and the width of the inter-module bus is narrow enough to require roughly one-third the number of pins as a traditional organization. The result is that an FB-DIMM organization can handle roughly 24 times the storage capacity as a traditional DDR3-based system (assuming DDR3 systems will indeed be limited to one DIMM per channel), without sacrificing any bandwidth and even leaving room for increased intra-module bandwidth (note that the FB-DIMM module should be capable of reaching the same performance as a graphics-card organization).

7.5 Issues in DRAM Systems, Briefly

The previous section already touched on an important issue facing the system designers of tomorrow: bandwidth increases are not free and require some form of trade-off. The traditional approach has been to reduce the number of drops on the shared bus, but this approach cleary fails to scale. The *Overview* section described other issues, namely that protocol-level details interact with system-level organization parameters in non-intuitive ways that can reduce performance by factors of two or more. There are many other issues facing future DRAM systems, and these will be treated in more detail in the upcoming chapters. The following is a brief introduction to several of the issues.

7.5.1 Architecture and Scaling

The bandwidth improvements in each DRAM generation have come at a cost in channel capacity, but they have also come at a cost in the granularity of access. Through a trick of trading off space for time, DRAM designers have increased the data rate at the DRAM's I/O pins without having also to increase the speed of the DRAM's core. DDR improves speed relative to single data rate (SDR) by moving to a *2n prefetch* architecture—one in which twice the number of bits are fetched from the DRAM core. The bandwidth increase on one side of the interface is matched by a bandwidth increase on the other side; the I/O side is twice as fast, and the core side is twice as wide (on the I/O side of the DRAM, bits are transmitted twice as fast as before; on the core side, twice as many bits are fetched in a single cycle). DDR2 improves speed again by moving to *4n prefetch*; DDR3 is expected to implement *8n prefetch*. In each instance, the minimum number of bits that the DRAM can read or write increases by a factor of two. The implication is that processors need to double the amount of data that they read and write in each operation, but the question remains as to how far a cache block will scale before SRAM designers say *enough*: 128 B per block? 256 B? 1 KB?

7.5.2 Topology and Timing

As previously discussed, there is a direct connection between signal integrity and system organization; changes to the system organization can significantly improve or degrade signal integrity. The same is true of bus topologies. For example, designers of unidirectional busses do not need to concern themselves with turnaround time, a limitation on throughput when the masters of bi-directional buses change. A related issue is that of the timing convention used. When the topology changes, the timing convention may need to be changed to follow suit. Unidirectional busses would tend to work better with source-synchronous clocking, for example, but source-synchronous clocking is not as simple as a global clocking scheme.

7.5.3 Pin and Protocol Efficiency

Pin cost is becoming a major concern; the cost of a transistor or a capacitor on-die is reducing at a tremendous rate, and packaging costs are not decreasing as rapidly. The obvious conclusion is that in the future, DRAM design will be far more concerned about pin count than transistor count. The only way to reduce significantly the number of pins needed in a part, without also significantly reducing the part's data bandwidth (e.g., by simply reducing the number of data pins on the package), is to rethink the protocol and allow a trade-off between pins used for data and control (address, command, etc.).

7.5.4 Power and Heat Dissipation

DRAMs have traditionally not accounted for much of a design's power or heat dissipation. For instance, DRAMs typically do not come with heat sinks, nor do DIMMs. However, with modern inter-chip signaling rates exceeding 1 Gbps per pin (as in present-day FB-DIMMs), both power dissipation and heat dissipation become serious issues. Previous-era DIMMs exhibited power dissipation on the order of 1 W. Current FB-DIMMs dissipate nearly ten times that. Couple this with the ability to put 25 times the number of FB-DIMMs into a system (three times as many channels, eight times as many DIMMs per channel), and you now have a significant heating problem. Modern blade servers are a popular design choice and work because only a handful of components in the system dissipate significant heat, and DRAM was never one of these. Now, the DRAM subsystem threatens to complicate blade-server design, among other things.

7.5.5 Future Directions

Currently, much of the focus in future DRAM design has been placed on increasing bandwidth via increased pin rates. This only scales well for a limited design window. In particular, it places too much emphasis on optimizing the device and not enough on optimizing the system. This is precisely the type of narrow-scope thinking that has led to the systemic problems described in the *Overview* chapter: designers thinking only at the device level can create system-level problems, just as designers thinking only at the system level can create device-level problems.

The solution is non-trivial, because it requires facing economic arguments. DRAM is a device that tries to suit many different needs. A design solution must work just as well in a DIMM organization as being soldered directly to a motherboard, talking one-on-one with the microprocessor. In other words, in many cases the device *is* the system. How to marry these two perspectives will be an ongoing concern.

CHAPTER 8

DRAM Device Organization: Basic Circuits and Architecture

In this chapter, basic circuits and architecture of DRAM devices are described. Modern DRAM devices exist as the result of more than three decades of devolutionary development, and it is impossible to provide a complete overview as well as an in-depth coverage of circuits and architecture of various DRAM devices in a single chapter. The limited goal in this chapter is to provide a broad overview of circuits and functional blocks commonly found in modern DRAM devices, and then proceed in subsequent chapters to describe larger memory systems consisting of DRAM devices composed of the commonly found circuits and functional blocks described herein.

This chapter proceeds through an examination of the basic building blocks of modern DRAM devices by first providing a superficial overview of a *Fast Page Mode* (FPM) DRAM device. Basic building blocks such as DRAM storage cells, DRAM array structure, voltage sense amplifiers, decoders, control logic blocks, data I/O structures, and packaging considerations are examined in this chapter. Specific DRAM devices, such as SDRAM, DDR SDRAM and D-RDRAM devices, and the evolution of DRAM devices, in general, are examined in Chapter 12.

8.1 DRAM Device Organization

Figure 8.1 illustrates the organization and structure of an FPM DRAM device that was widely used in the 1980s and early 1990s. Internally, the DRAM storage cells in the FPM DRAM device in Figure 8.1 are organized as 4096 rows, 1024 columns per row, and 16 bits of data per column. In this device, each time a row access occurs, a 12-bit address is placed on the address bus and the *row-address strobe* (RAS)[1] is asserted by an external memory controller. Inside the DRAM device, the address on the address bus is buffered by the row address buffer and then sent to the row decoder. The row address decoder then accepts the 12-bit address and selects 1 of 4096 rows of storage cells. The data values contained in the selected row of storage cells are then sensed and kept active by the array of sense amplifiers. Each row of

[1]RAS is known as both row-address strobe (more common) or as row-access strobe. The author prefers "access" because of the way the DRAM access protocol has morphed from more of a signal-based interface to a command-based interface. Both "address" and "access" are commonly accepted usage for "A" in both RAS and CAS.

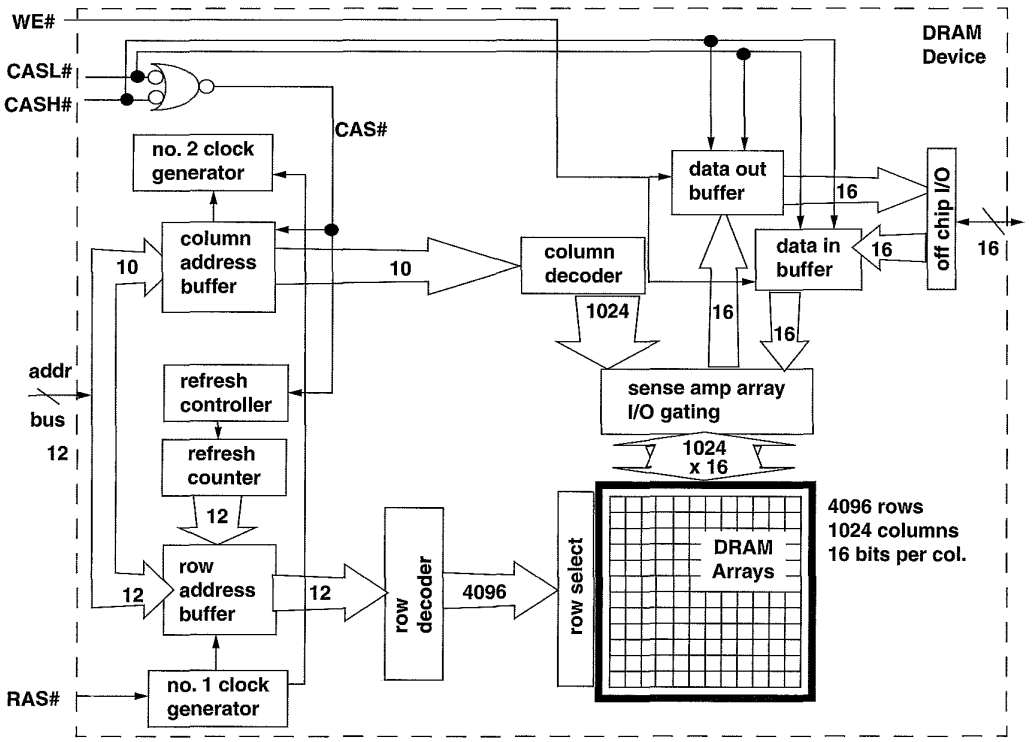


FIGURE 8.1: A 64-Mbit Fast Page Mode DRAM device (4096 x 1024 x 16).

DRAM cells in the DRAM device illustrated in Figure 8.1 consists of 1024 columns, and each column is 16 bits wide. That is, the 16-bit-wide column is the basic addressable unit of memory in this device, and each column access that follows the row access would read or write 16 bits of data from the same row of DRAM.[2]

In the FPM DRAM device illustrated in Figure 8.1, column access commands are handled in a similar manner as the row access commands. For a column access command, the memory controller places a 10-bit address on the address bus and then asserts the appropriate *column-access strobe* (CAS#) signals. Internally, the DRAM chip takes the 10-bit column

[2]The FPM DRAM device illustrated in Figure 8.1 does allow each 8-bit half of the 16-bit column to be accessed independently through the use of separate *column-access strobe high* (CASH) and *column-access strobe low* (CASL) signals. However, since the data bus is 16 bits wide, a column of data in this device is 16 bits rather than 8 bits. Equivalently, modern DRAM devices make use of data mask signals to enable partial data write operations within a single column. Some other DRAM devices such as XDR devices make use of sophisticated command encoding to control sub-column read and write operations.

address, decodes it, and selects 1 column out of 1024 columns. The data for that column is then placed onto the data bus by the DRAM device in the case of an ordinary column read command or is overwritten with data from the memory controller depending on the *write enable* (WE) signal.

All DRAM devices, from the venerable FPM DRAM device to modern DDRx SDRAM devices, to high data rate XDR DRAM devices to low-latency RLDRAM devices, to low-power MobileRAM devices, share some basic circuits and functional blocks. In many cases, different types of DRAM devices from the same DRAM device manufacturer share the exact same cells and same array structures. For example, the DRAM cells in all DRAM devices are organized into one or more arrays, and each array is arranged into a number of rows and columns. All DRAM devices also have some logic circuits that control the timing and sequence of how the device operates. The FPM DRAM device shown in Figure 8.1 has internal clock generators as well as a built-in refresh controller. The FPM DRAM device keeps the address of the next row that needs to be refreshed so that when a refresh command is issued, the row address to be refreshed can be loaded from the internal refresh counter instead of having to be loaded from the off-chip address bus. The inclusion of the fresh counter in the DRAM device frees the memory controller from having to keep track of the row addresses in the refresh cycles.

Advanced DRAM devices such as ESDRAM, Direct RDRAM, and RLDRAM have evolved to include more logic circuitry and functionality on-chip than the basic DRAM device examined in this chapter. For example, instead of a single DRAM array in an FPM DRAM device, modern DRAM devices have multiple banks of DRAM arrays, and some DRAM devices have additional row caches or write buffers that allow for read-around-write functionality. The discussion in this chapter is limited to basic circuitry and architecture, and advanced performance-enhancing logic circuits not typically found on standard DRAM devices are described separately in discussions of specific DRAM devices and memory systems.

8.2 DRAM Storage Cells

Figure 8.2 shows the circuit diagram of a basic one-transistor, one-capacitor (1T1C) cell structure used in modern DRAM devices to store a single bit of data. In this structure, when the access transistor is turned on by applying a voltage on the gate of the access transistor, a voltage representing the data value is placed onto the bitline and charges the storage capacitor. The storage capacitor then retains the stored charge after the access transistor is turned off and the voltage on the wordline is removed. However, the electrical charge stored in the storage capacitor will gradually leak away with the passage of time. To ensure data integrity, the stored data value in the DRAM cell must be periodically read out and written back by the DRAM device in a process known as *refresh*. In the following section, the relationships between cell capacitance, leakage, and the need for refresh operations are briefly examined.

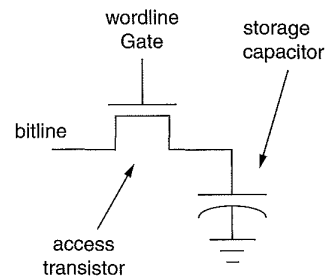


FIGURE 8.2: Basic 1T1C DRAM cell structure.

Different cell structures, such as a three-transistor, one-capacitor (3T1C) cell structure in Figure 8.3 with separate read access, write access, and storage transistors, were used in early DRAM designs.[3] The 3T1C cell structure has an interesting characteristic in that reading data from the storage cell does not require the content of the cell to be discharged onto a shared bitline. That is, data reads to DRAM cells are not destructive in 3T1C cells, and a simple read cycle does not require data to be restored into the storage cell as they are in 1T1C cells. Consequently, random read cycles are faster for 3T1C cells than 1T1C cells. However, the size advantage of the 1T1C cell has ensured

[3]The first commercial DRAM device, Intel's 1103 DRAM device, utilized a 3T1C cell structure.

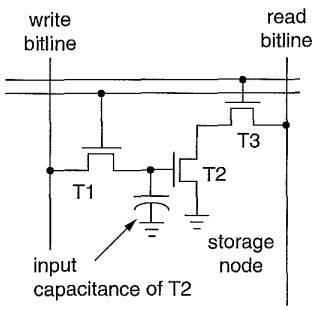


FIGURE 8.3: 3T1C DRAM cell.

that this basic cell structure is used in all modern DRAM devices.

Aside from the basic 1T1C cell structure, research is ongoing to utilize alternative cell structures such as the use of a single transistor on a Silicon-on-Insulator (SOI) process as the basic cell. In one proposed structure, the isolated substrate is used as the charge storage element, and a separate storage capacitor is not needed. Similar to data read-out of the 3T1C cell, data read-out is not destructive, and data retrieval is done via current sensing rather than charge sensing. However, despite the existence of alternative cell structures, the 1T1C cell structure is used as the basic charge storage cell structure in all modern DRAM devices, and the focus in this chapter is devoted to this dominant 1T1C DRAM cell structure.

8.2.1 Cell Capacitance, Leakage, and Refresh

In a 90-nm DRAM-optimized process technology, the capacitance of a DRAM storage cell is on the order of 30 fF, and the leakage current of the DRAM access transistor is on the order of 1 fA. With a cell capacitance of 30 fF and a leakage current of 1 fA, a typical DRAM cell can retain sufficient electrical charge that will continue to resolve to the proper digital value for an extended period of time—from hundreds of milliseconds to tens of seconds. However, transistor leakage characteristics are temperature-dependent, and DRAM cell data retention times can vary dramatically not only from cell to cell at the same time and temperature, but also at different times for the same DRAM cell.[4] However, memory systems must be designed so that not a single bit of data is lost due to charge leakage. Consequently, every single DRAM cell in a given device must be refreshed at least once before any single bit in the entire device loses its stored charge due to leakage. In most modern DRAM devices, the DRAM cells are typically refreshed once every 32 or 64 ms. In cases where DRAM cells have storage capacitors with low capacitance values or high leakage currents, the time period between refresh intervals is further reduced to ensure reliable data retention for all cells in the DRAM device.

8.2.2 Conflicting Requirements Drive Cell Structure

Since the invention of the 1T1C DRAM cell, the physical structure of the basic DRAM cell has undergone continuous evolution. DRAM cell structure evolution occurred as a response to the conflicting requirements of smaller cell sizes, lower voltages, and noise tolerances needed in each new process generation. Figure 8.4 shows an abstract implementation of the 1T1C DRAM cell structure. A storage capacitor is formed from a *stacked* (or folded plate) capacitor structure that sits in between the polysilicon layers above active silicon. Alternatively, some DRAM device manufacturers instead use cells with *trench* capacitors that dive deeply into the active silicon area. Modern DRAM devices typically utilize one of these two different forms of the capacitor structure as the basic charge storage element.

In recent years, two competing camps have been formed between manufacturers that use a trench capacitor and manufacturers that use a stacked capacitor as the basic charge storage element. Debates are ongoing as to the relative costs and long-term scalability of each design. For manufacturers that seek to integrate DRAM cells with logic circuits on the same process technology, the trench capacitor structure allows for better integration of embedded DRAM cells with logic-optimized semiconductor process technologies. However, manufacturers that focused on stand-alone DRAM devices appear to

[4]The variable leakage problem is a well-known and troublesome phenomenon for DRAM manufacturers that leads to variable retention times (VRTs) in DRAM cells.

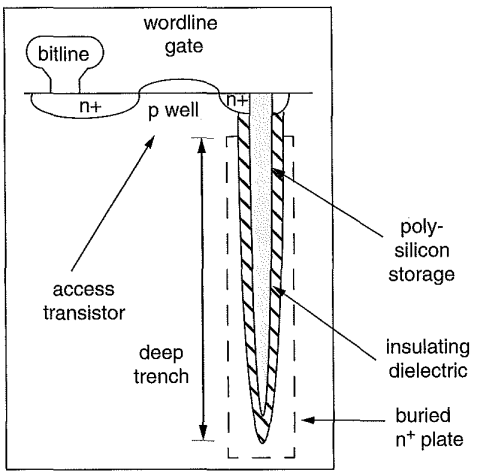


FIGURE 8.4: Cross-section view of a 1T1C DRAM cell with a trench capacitor. The storage capacitor is formed from a **trench** capacitor structure that dives deeply into the active silicon area. Alternatively, some DRAM device manufacturers instead use cells with a **stacked** capacitor structure that sits in between the polysilicon layers above the active silicon.

favor stacked capacitor cell structures as opposed to the trench capacitor structures. Currently, DRAM device manufacturers such as Micron, Samsung, Elpida, Hynix, and the majority of the DRAM manufacturing industry use the stacked capacitor structure, while Qimonda, Nanya, and several other smaller DRAM manufacturers use the trench capacitor structure.

8.2.3 Trench Capacitor Structure

Currently, the overriding consideration for DRAM devices, in general, and commodity DRAM devices, in particular, is that of cost-minimization; This over-riding consideration leads directly to the pressure to reduce the cell size—either to increase selling price by putting more DRAM cells onto the same piece of silicon real estate or to reduce cost for the same number of storage cells. The pressure to minimize cell area, in turn, means that the storage cell either has to grow into a three-dimensional stacked capacitor above the surface of the silicon or has to grow deeper into and trench below the surface of the active silicon. Figure 8.4 shows a diagram of the 1T1C DRAM cell with a deep trench capacitor as the storage element. The abstract illustration in Figure 8.4 shows the top cross section of the trench capacitor.[5] The depth of the trench capacitor allows a DRAM cell to decrease the use of the silicon surface area without decreasing storage cell capacitance. Trench capacitor structures and stacked capacitor structures have respective advantages and disadvantages. One advantage of the trench capacitor design is that the three-dimensional capacitor structure is under the interconnection layers so that the higher level metallic layers can be more easily made planar. The planar characteristic of the metal layers means that the process could be more easily integrated into a logic-optimized process technology, where there are more metal layers above the active silicon. The buried structure also means that the trench capacitor could be constructed before logic transistors are constructed. The importance of this subtle distinction means that processing steps to create the capacitive layer could be activated before logic transistors are fabricated, and the performance characteristics of logic transistors would not be degraded by formation of the (high-temperature) capacitive layer.[6]

8.2.4 Stacked Capacitor Structure

The stacked capacitor structure uses multiple layers of metal or conductive polysilicon above the surface of the silicon substrate to form the plates of the capacitor to form the plates of the capacitor that holds the stored electrical charge. Figure 8.5 shows an abstract

[5]In some modern DRAM devices that use trench capacitors, the depth-to-width aspect ratio of the trench cell exceeds 50:1.
[6]The integration of DRAM cells with the logic circuit on the same process technology is non-trivial. Please see the continuing discussion in Section 8.10.

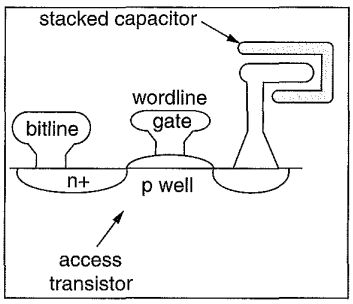


FIGURE 8.5: Abstract view of a 1T1C DRAM cell with stacked capacitor.

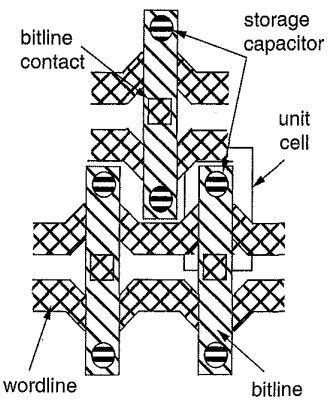


FIGURE 8.6: Top-down view of a DRAM array.

illustration of the stacked capacitor structures. The capacitor structure in Figure 8.5 is formed between two layers of polysilicon, and the capacitor lies underneath the bitline. It is referred to as the *Capacitor-under-Bitline* (CUB) structure. The stacked capacitive storage cell can also be formed above the bitline in the *Capacitor-over-Bitline* (COB) structure. Regardless of the location of the storage cell relative to the bitline, both the CUB and COB structures are variants of the stacked capacitor structure, and the capacitor resides in the polysilicon layers above the active silicon. The relentless pressure to reduce DRAM cell size while retaining cell capacitance has forced the capacitor structure to grow in the vertical dimension, and the evolution of the stacked capacitor structure is a natural migration from two-dimensional plate capacitor structures to three-dimensional capacitor structures.

8.3 RAM Array Structures

Figure 8.6 illustrates an abstract DRAM array in a top-down view. The figure shows a total of six cells, with every two cells sharing the same bitline contact. The figure also abstractly illustrates the size of a cell in the array. The size of a unit cell is 8 F^2. In the current context, "F" is a process-independent metric that denotes the smallest feature size in a given process technology. In a 90-nm process, F is literally 90 nm, and an area of 8 F^2 translates to 64,800 nm^2 in the 90-nm process. The cross-sectional area of a DRAM storage cell is expected to scale linearly with respect to the process generation and maintains the cell size of 6~8 F^2 in each successive generation.

In DRAM devices, the process of data read out from a cell begins with the activation of the access transistor to that cell. Once the access transistor is turned on, the small charge in the storage capacitor is placed on the bitline to be resolved into a digital value.[7] Figure 8.7 illustrates a single bank of DRAM storage cells where the row address is sent to a row decoder, and the row decoder selects one row of cells. A row of cells is formed from one or more wordlines that are driven concurrently to activate one cell on each one of thousands of bitlines. There may be hundreds of cells connected to the same bitline, but only one cell per bitline will share its stored charge with the bitline at any given instance in time. The charge sharing process by the storage capacitor minutely changes the voltage level on the bitline, and the resulting voltage on the bitline is then resolved into a digital value by a differential sense amplifier. The differential sense amplifier is examined in the following section.

In modern DRAM devices, the capacitance of a storage capacitor is far smaller than the capacitance of the bitline. Typically, the capacitance of a storage

[7]Bitlines are also known as digitlines.

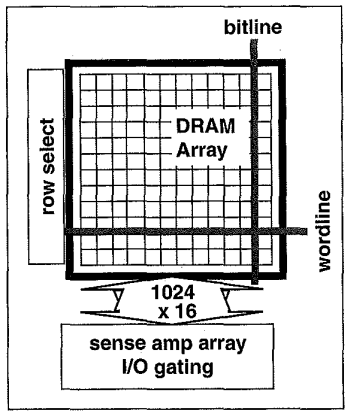


FIGURE 8.7: One DRAM bank illustrated: consisting of one DRAM array, one array of sense amplifiers, and one row decoder—note that the row decoder may be shared across multiple banks.

capacitor is one-tenth of the capacitance of the long bitline that is connected to hundreds or thousands of other cells. The relative capacitance values create the scenario that when the small charge contained in a storage capacitor is placed on the bitline, the resulting voltage on the bitline is small and difficult to measure in an absolute sense. In DRAM devices, the voltage sensing problem is resolved through the use of a differential sense amplifier that compares the voltage of the bitline to a reference voltage.

The use of the differential sense amplifier, in turn, places certain requirements on the array structure of the DRAM device. In particular, the use of a differential sense amplifier means that instead of a single bitline, a pair of bitlines is used to sense the voltage value contained in any DRAM cell. Furthermore, in order to ensure that bitlines are closely matched in terms of voltage and capacitance values, they must be closely matched in terms of path lengths and the number of cells attached. These requirements lead to two distinctly different array structures: open bitline structures and folded bitline structures. The structural difference between an array with an open bitline structure and an array with a folded bitline structure is that in the open bitline structure, bitline pairs used for each sense amplifier come from separate array segments, while bitline pairs in a folded bitline structure come from the same array segment. The different structure types have different advantages and disadvantages in terms of cell size and noise tolerance. Some of the important advantages and disadvantages are selected for discussion in the following sections.

8.3.1 Open Bitline Array Structure

Figure 8.8 shows an abstract layout of an *open bitline* DRAM array structure. In the open bitline structure, bitline pairs used for each sense amplifier come from separate array segments. Figure 8.8 shows that the open bitline structure leads to a high degree of regularity in the array structure, and the result is that cells in an open bitline structure can be packed closely together. Typically, DRAM cells in an open bitline structure can occupy an area as small as 6 F^2. In contrast, DRAM cells in a *folded bitline* structure typically occupy a minimum area of 8 F^2.[8] The larger area used by cells in a folded bitline structure is due to the fact that two bitlines are routed through the array for each DRAM cell in the folded bitline structure, while only one bitline is routed through the array for each cell in an open bitline structure.

Open bitline array structures were used in 64-Kbit and earlier DRAM generations. Some 256-Kbit DRAM devices also used open bitline structures. However, despite the advantage of smaller cell sizes, open bitline structures also have some disadvantages. One disadvantage of the open bitline structure is that it requires the use of dummy array segments at the edges of the DRAM array in order to ensure that the lengths and capacitance characteristics of the bitline pairs are closely matched. Another disadvantage of the classic

[8]Currently, most manufacturers utilize DRAM cells that occupy an area of 8 F^2, and the International Technology Roadmap for Semiconductors (ITRS) predicts that DRAM manufacturers will begin transition to new DRAM cell structures that are only 6 F^2 in 2008. However, Micron announced in 2004 that it had succeeded in the development of a Metal-Insulator-Metal (MIM) capacitor that occupies an area of 6 F^2 and began shipping products based on this structure in 2004 and 2005.

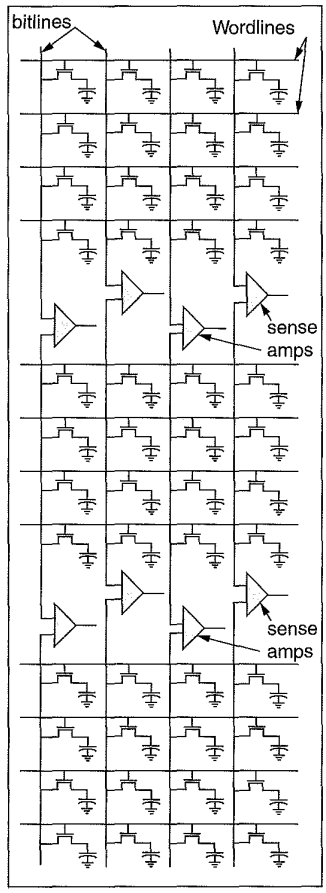


FIGURE 8.8: Open bitline DRAM array structure.

open bitline structure is that bitline pairs in the open bitline array structure come from different array segments, and each bitline would be more susceptible to electronic noises as compared to bitlines in the folded bitline structure.

The larger area for routing and dummy array segments in open bitline structure minutely dilutes the cell size advantage of the open bitline structures. The various trade-offs, in particular, the noise tolerance issue, have led to the predominance of folded bitline structures in modern DRAM devices. Consequently, open bitline array structures are not currently used in modern DRAM devices. However, as process technology advances, open bitline structures promise potentially better scalability for the DRAM cell size in the long run. Research in the area of basic DRAM array structure is thus ongoing. Open bitline array structures or more advanced twisting or folding of the bitline structure with the cell size advantage of the open bitline architecture can well make a comeback as the mainstream DRAM array structure in the future.

8.3.2 Folded Bitline Array Structure

Figure 8.9 shows a DRAM array with a folded bitline structure. In the folded bitline configuration, bitlines are routed in pairs through the DRAM array structure, fulfilling several critical requirements of the sensitive differential sense amplifier. The close proximity of the bitline pairs in the folded bitline structure means that the differential sense amplifier circuit, when paired with this array structure, exhibits superior common-mode noise rejection characteristics. That is, in the case of a charge spike induced by a single event upset (SEU) neutron or alpha particle striking the DRAM device, the voltage spike would have a good chance of appearing as common-mode noise at the input of the differential sense amplifier. In the case of the open bitline array structure, the charge spike induced by the SEU would likely appear as noise on only one bitline of a bitline pair that connects to a sense amplifier.

Figure 8.9 shows a logical layout of the folded bitline structure where alternate pairs of DRAM cells are removed from an open bitline array. The array of DRAM cells is then compressed, resulting in the folded bitline structure illustrated in Figure 8.9. The folded bitline structure shown in Figure 8.9 is a simple twisting scheme where the bitline pairs cross over each other for every two transistors on the bitline. More advanced bitline folding schemes are being study to reduce the area impact while retaining the noise immunity aspect of the folded bitline structure.

8.4 Differential Sense Amplifier

In DRAM devices, the functionality of resolving small electrical charges stored in storage capacitors into digital values is performed by a differential

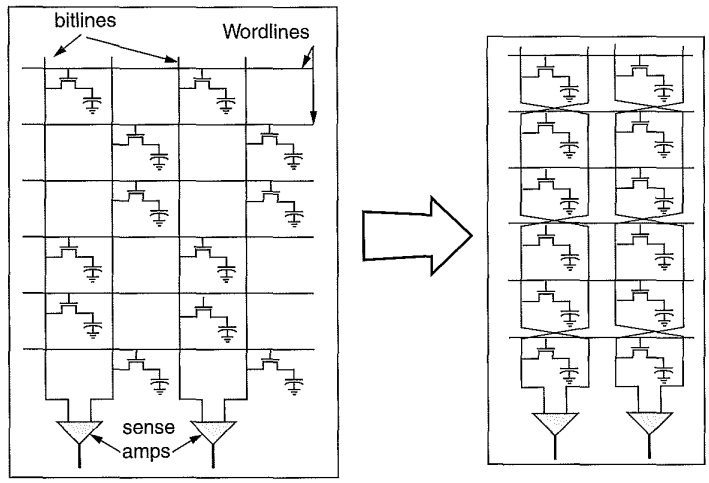


FIGURE 8.9: One type of folded bitline array structure.

sense amplifier. In essence, the differential sense amplifier takes the voltages from a pair of bitlines as input, senses the difference in voltage levels between the bitline pairs, and amplifies the difference to one extreme or the other.

8.4.1 Functionality of Sense Amplifiers in DRAM Devices

Sense amplifiers in modern DRAM devices perform three generalized functions. The first function is to sense the minute change in voltage that occurs when an access transistor is turned on and a storage capacitor places its charge on the bitline. The sense amplifier compares the voltage on that bitline against a reference voltage provided on a separate bitline and amplifies the voltage differential to the extreme so that the storage value can be resolved as a digital 1 or 0. This is the sense amplifier's primary role in DRAM devices, as it senses minute voltage differentials and amplifies them to represent digital values.

The second function is that it also restores the value of a cell after the voltage on the bitline is sensed and amplified. The act of turning on the access transistor allows a storage capacitor to share its stored charge with the bitline. However, the process of sharing the electrical charge from a storage cell discharges that storage cell. After the process of charge sharing occurs, the voltage within the storage cell is roughly equal to the voltage on the bitline, and this voltage level cannot be used for another read operation. Consequently, after the sensing and amplification operations, the sense amplifier must also restore the amplified voltage value to the storage cell.

The third function is that the sense amplifiers also act as a temporary data storage element. That is, after data values contained in storage cells are sensed and amplified, the sense amplifiers will continue to drive the sensed data values until the DRAM array is precharged and readied for another access. In this manner, data in the same row of cells can be accessed from the sense amplifier without repeated row accesses to the cells themselves. In this role, the array of sense amplifiers effectively acts as a row buffer that caches an entire row of data. As a result, an array of sense amplifiers is also referred to as a row buffer, and management policies are devised to control operations of the sense amplifiers. Different row buffer management policies dictate whether an array of sense amplifiers will retain the data for an indefinite period of time (until the next refresh), or will discharge it immediately after data has been restored

to the storage cells. Active sense amplifiers consume additional current above quiescent power levels, and effective management of the sense amplifier operation is an important task for systems seeking optimal trade-off points between performance and power consumption.

8.4.2 Circuit Diagram of a Basic Sense Amplifier

Figure 8.10 shows the circuit diagram of a basic sense amplifier. More complex sense amplifiers in modern DRAM devices contain the basic elements shown in Figure 8.10, as well as additional circuit elements for array isolation, careful balance of the sense amplifier structure, and faster sensing capability. In the basic sense amplifier circuit diagram shown in Figure 8.10, the equalization (*EQ*) signal line controls the voltage equalization circuit. The functionality of this circuit is to ensure that the voltages on the bitline pairs are as closely matched as possible. Since the differential sense amplifier is designed to amplify the voltage differential between the bitline pairs, any voltage imbalance that exists on the bitline pairs prior to the activation of the access transistors would degrade the effectiveness of the sense amplifier.

The heart of the sense amplifier is the set of four cross-connected transistors, labelled as the sensing circuit in Figure 8.10. The sensing circuit is essentially a bi-stable circuit designed to drive the bitline pairs to complementary voltage extremes, depending on the respective voltages on the bitlines at the time the *SAN* (Sense-Amplifier N-Fet Control) and *SAP* (Sense-Amplifier P-Fet Control) sensing signals are activated. The *SAN* signal controls activation of the NFets in the sensing circuit, and the *SAP* signal controls the activation of the PFets in the sensing circuit. After the assertion of the *SAN* and *SAP*, the bitlines are driven to the full voltage levels. The column-select line (*CSL*) then turns on the output transistors and allows the fully driven voltage to reach the output and be read out of the DRAM device. At the same time, the access transistor for the accessed cell remains open, and the fully driven voltage on the bitline now recharges the storage capacitor. Finally, in case of a write operation, the column-select line and the write enable (WE) signals collectively allow the input write drivers to provide a large current to overdrive the sense amplifier and the bitline voltage. Once the sense amplifier is overdriven to the new data value, it will then hold that value and drive it into the DRAM cell through the still open access transistor.

8.4.3 Basic Sense Amplifier Operation

The maximum voltage that can be placed across the access transistor is $V_{gs} - V_t$. (V_t is the threshold

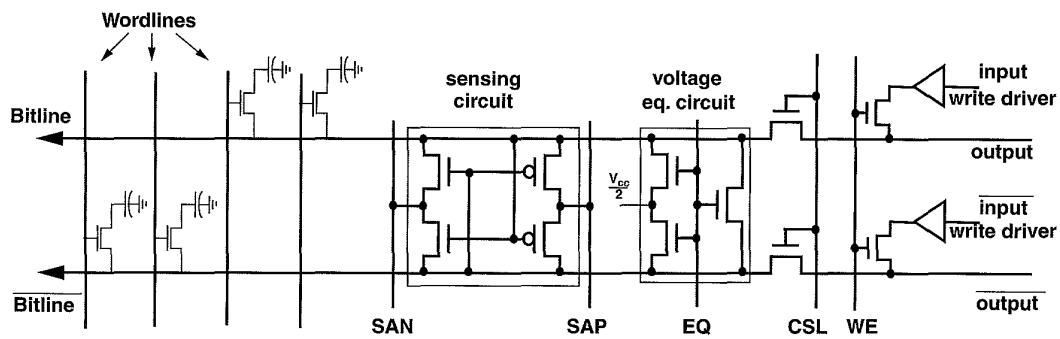


FIGURE 8.10: Basic sense amplifier circuit diagram.

voltage of the access transistor, and V_{gs} is the gate-source voltage on the access transistor.) By overdriving the wordline voltage to $V_{cc} + V_t$, the storage capacitor can be charged to full voltage (maximum of V_{cc}) by the sense amplifier in the restore phase of the sensing operation. In modern DRAM devices, the higher-than-V_{cc} wordline voltage is generated by additional level-shifting voltage pumping circuitry not examined in this text.

Figure 8.11 shows four different phases in the sensing operations of a differential sense amplifier. The precharge, access, sense, and restore operations of a sense amplifier are labelled as phases zero, one, two, and three, respectively. The reason that the precharge phase is labelled as phase zero is because the precharge phase is typically considered as a separate operation from the phases of a row-access operation. That is, while the *Precharge* phase is a prerequisite for a row-access operation, it is typically performed separately from the row-access operation itself. In contrast, *Access*, *Sense*, and *Restore* are three different phases that are performed in sequence for the row-access operation.

Phase zero in Figure 8.11 is labelled as *Precharge*, and it illustrates that before the process of reading data from a DRAM array can begin, the bitlines in a DRAM array are precharged to a reference voltage, V_{ref}. In many modern DRAM devices, $V_{cc}/2$, the voltage halfway between the power supply voltage and ground, is used as the reference voltage. In Figure 8.11, the equalization circuit is activated to place the reference voltage for the bitlines, and the bitlines are precharged to V_{ref}.

Phase one in Figure 8.11 is labelled as (cell) *Access*, and it illustrates that as a voltage is applied to a wordline, that wordline is overdriven to a voltage that is at least V_t above V_{cc}. The voltage on the wordline activates the access transistors, and the selected storage cells discharge their contents onto the respective bitlines. In this case, since the voltage in the storage cell represents a digital value of "1," the charge sharing process minutely increases the voltage on the bitline from V_{ref} to V_{ref}^+. Then, as the voltage on the bitline changes, the voltage on the bitline begins to affect operations of the cross-connected sensing circuit. In the case illustrated in Figure 8.11, the slightly higher voltage on the bitline begins to drive the lower NFet to be more conductive than the upper NFet. Conversely, the minute voltage difference also drives the lower PFet to be less conductive than the upper PFet. The bitline voltage thus biases the sensing circuit for the sensing phase.

Phase two in Figure 8.11 is labelled *Sense*, and it illustrates that as the minute voltage differential drives a bias into the cross-connected sensing circuit, SAN, the DRAM device's NFet sense amplifier control signal, turns on and drives the voltage on the lower bitline down.[9] The figure shows that as SAN turns on, the more conductive lower NFet allows SAN to drive the lower bitline down in voltage from V_{ref} to ground. Similarly, SAP, the PFet sense amplifier control signal, drives the bitline to a fully restored voltage value that represents the digital value of "1." The SAN and SAP control signals thus collectively force the bi-stable sense amplifier circuit to be driven to the respective maximum or minimum voltage rails.

Finally, phase three of Figure 8.11 is labelled as *Restore*, and it illustrates that after the bitlines are driven to the respective maximum or minimum voltage values, the overdriven wordline remains active, and the fully driven bitline voltage now restores the charge in the storage capacitor through the access transistor. At the same time, the voltage value on the bitline can be driven out of the sense amplifier circuit to provide the requested data. In this manner, the contents of a DRAM row can be accessed concurrently with the row restoration process.

8.4.4 Voltage Waveform of Basic Sense Amplifier Operation

Figure 8.12 shows the voltage waveforms for the bitline and selected control signals illustrated in Figure 8.11. The four phases labelled in Figure 8.12 correspond

[9] In modern DRAM devices, the timing and shape of the SAN and SAP control signals are of great importance in defining the accuracy and latency of the sensing operation. However, for the sake of simplicity, the timing and shape of these important signals are assumed to be optimally generated by the control logic herein.

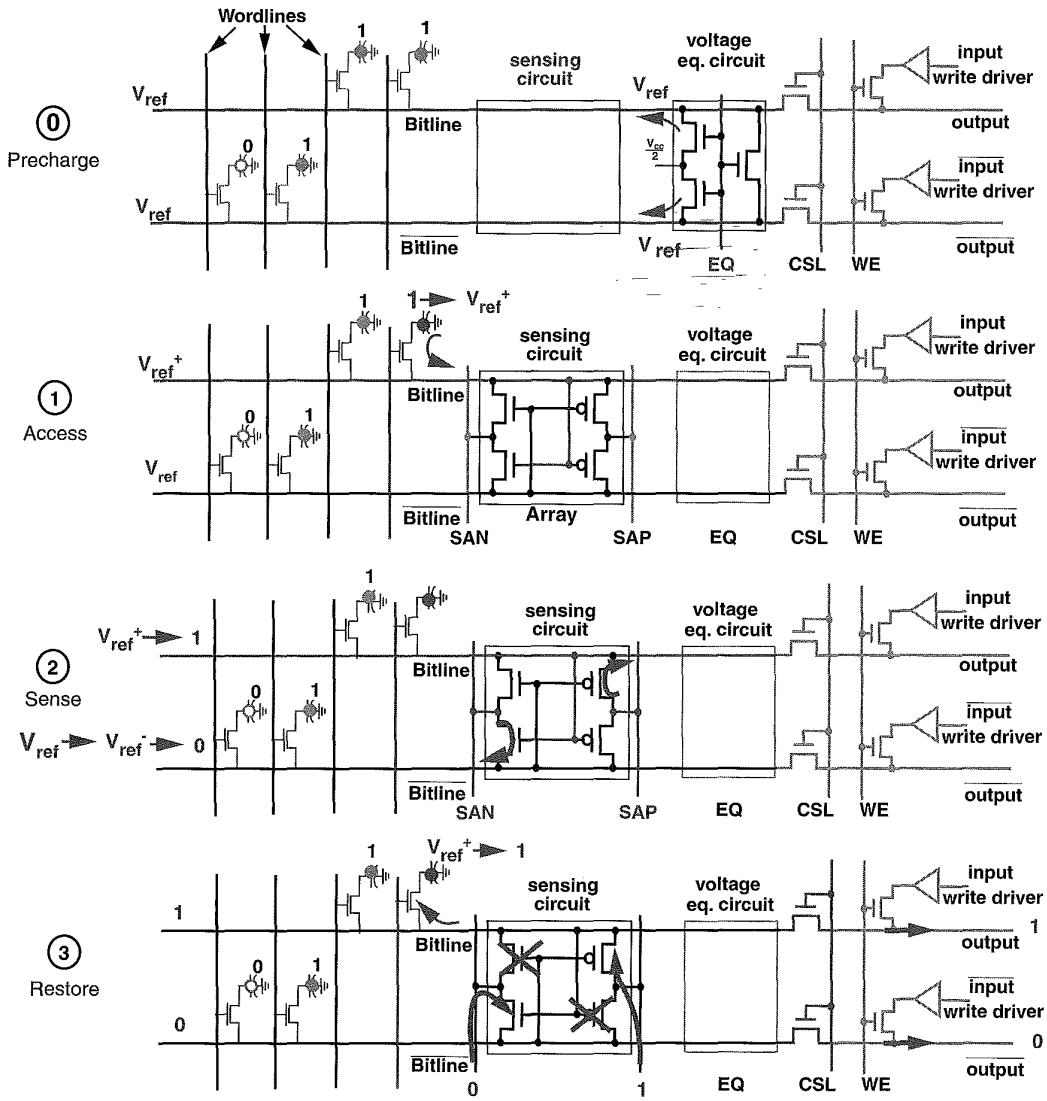


FIGURE 8.11: Illustrated diagrams of the sense amplifier operation. Read(1) example.

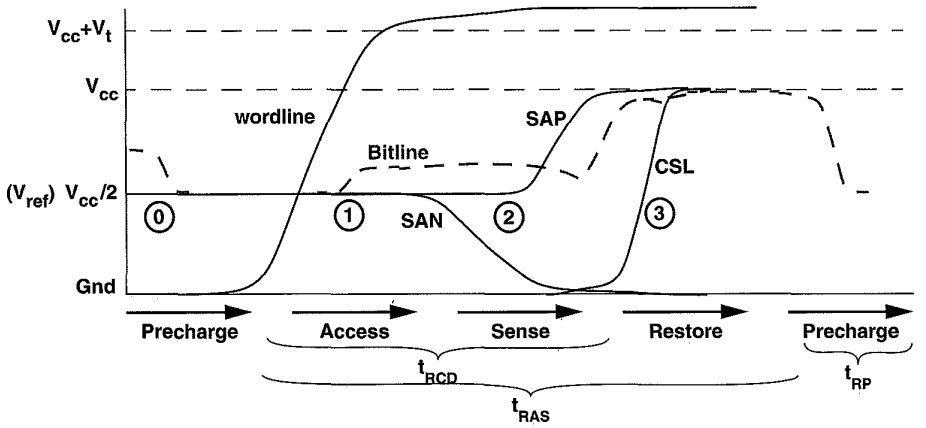


FIGURE 8.12: Simplified sense amplifier voltage waveform. Read(1) example.

to the four phases illustrated in Figure 8.11. Figure 8.12 shows that before a row-access operation, the bitline is precharged, and the voltage on the bitline is set to the reference voltage, V_{ref}. In phase one, the wordline voltage is overdriven to at least V_t above V_{cc}, and the DRAM cell discharges the content of the cell onto the bitline and raises the voltage from V_{ref} to V_{ref}^+. In phase two, the sense control signals SAN and SAP are activated in quick succession and drive the voltage on the bitline to the full voltage. The voltage on the bitline then restores the charge in the DRAM cells in phase three.

Figure 8.12 illustrates the relationship between two important timing parameters: t_{RCD} and t_{RAS}. Although the relative durations of t_{RCD} and t_{RAS} are not drawn to scale, Figure 8.12 shows that after time t_{RCD}, the sensing operation is complete, and the data can be read out through the DRAM device's data I/O. However, after a time period of t_{RCD} from the beginning of the activation process, data is yet to be restored to the DRAM cells. Figure 8.12 shows that the data restore operation is completed after a time period of t_{RAS} from the beginning of the activation process, and the DRAM device is then ready to accept a precharge command that will complete the entire row cycle process after a time period of t_{RP}.

8.4.5 Writing into DRAM Array

Figure 8.13 shows a simplified timing characteristic for the case of a write command. As part of the row activation command, data is automatically restored from the sense amplifiers to DRAM cells. However, in the case of a write command in commodity DRAM devices, data written by the memory controller is buffered by the I/O buffer of the DRAM device and used to overwrite the sense amplifiers and DRAM cells.[10] Consequently, in the case of a write command that follows a row activation command, the restore phase may be extended by the write recovery phase.[11]

[10]Some DRAM devices, such as Direct RDRAM devices, have write buffers. Data is not driven directly into the DRAM array by the data I/O circuitry in that case, but the write mechanism into the DRAM array remains the same when the write buffer commits the data into the DRAM array prior to a precharge operation that closes the page.

[11]The row cycle times of most DRAM devices are write-cycle limited. That is, the row cycle times of these DRAM devices are defined so that a single, minimal burst length column write command can be issued to a given row, between an activation command and a precharge command to the same row.

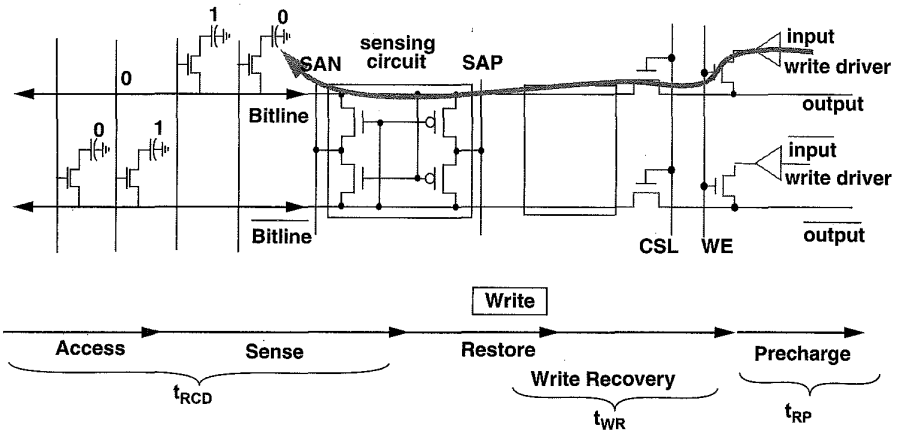


FIGURE 8.13: Row activation followed by column write into a DRAM array.

Figure 8.13 shows that the timing of a column write command means that a precharge command cannot be issued until after the correct data values have been restored to the DRAM cells. The time period required for write data to overdrive the sense amplifiers and written through into the DRAM cells is referred to as the write recovery time, denoted as t_{WR} in Figure 8.13.

8.5 Decoders and Redundancy

Modern DRAM devices rely on complex semiconductor processes for manufacturing. Defects on the silicon wafer or subtle process variations lead directly to defective cells, defective wordlines, or defective bitlines. The technique adopted by DRAM designers to tolerate some amount of defects and increase yield is through the use of redundant rows and columns. Figure 8.14 shows an array with redundant wordlines and redundant bitlines. The figure shows a DRAM array with 2^n rows and m redundant rows. The row decoder must select one out of $2^n + m$ rows with an n-bit-wide row address. The constraint placed on the decoder is that the spare replacement mechanism should not introduce unacceptable area overhead or additional delays into the address decode path.

In modern DRAM devices, each row of a DRAM array is connected to a decoder that can be selectively disconnected via a laser (or fuse) programmable link. In cases where a cell or an entire wordline is found to be defective, the laser or electrical programmable link for the standard decoder for that row disconnects the wordline attached to that standard decoder, and a spare row is engaged by connecting the address lines to match the address of the disconnected row. In this manner, the spare decoder can seamlessly engage the spare row when the address of the faulty row is asserted. The programmable links in the decoders may be laser programmable fuses or electrically programmable fuses, depending on the process technology and the mechanism selected by the DRAM design engineers as optimal for the specific manufacturing process technology.

Figure 8.15 shows a set of standard and spare decoder designs that are used to drive rows of DRAM cells in some DRAM devices. In such a DRAM device, a standard decoder is attached to each of the 2^n row of cells, and a spare decoder is attached to each of the spare rows. The figure shows that the standard decoder illustrated is functionally equivalent to an n-input NOR gate. In the standard decoder, each input of the functionally equivalent n-input NOR gate is

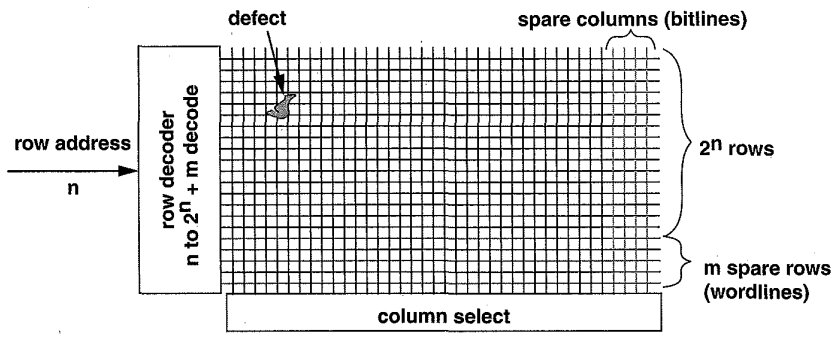


FIGURE 8.14: Redundant rows and columns in a DRAM array.

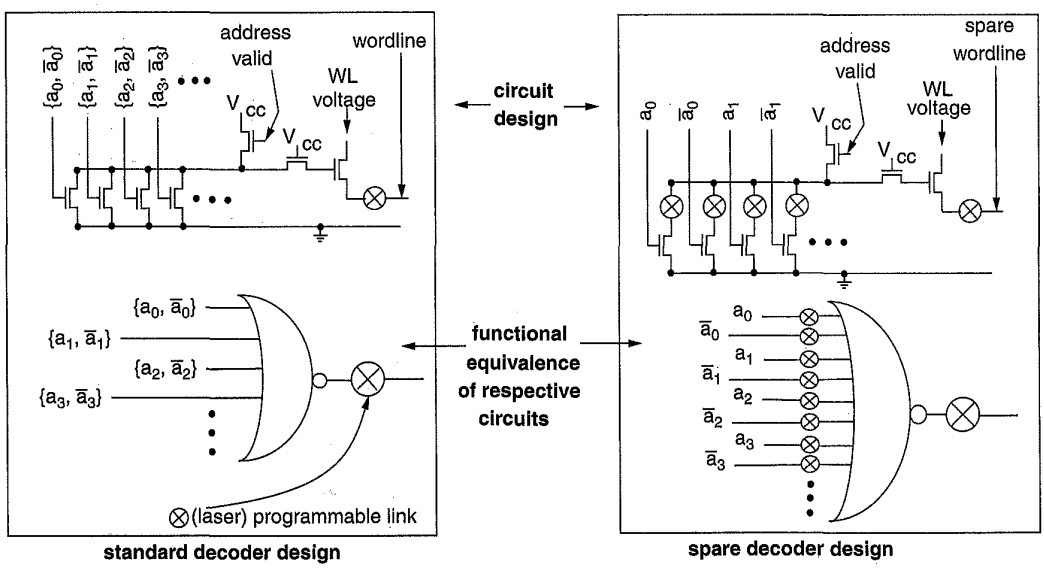


FIGURE 8.15: Standard and spare decoder design.

connected to one bit in the n-bit address—either the inverted or the non-inverted signal line. Figure 8.15 shows that the spare decoder illustrated is functionally equivalent to a $2n$-input NOR gate, and each bit in the n-bit address as well as the complement of each bit of the n-bit address is connected to the $2n$ inputs. In cases where a spare decoder is used, the input of the NOR gate is selectively disabled so that the remaining address signals match the address of the disabled standard decoder.

8.5.1 Row Decoder Replacement Example

Figure 8.16 illustrates the replacement of a standard decoder circuit with a spare decoder for a DRAM array with 16 standard rows and 2 spare rows. In Figure 8.16, the topmost decoder becomes active with the address of 0b1111, and each of the 16 standard decoders is connected to one of 16 standard rows. In the example illustrated in Figure 8.16, row 0b1010 is discovered to be defective, and the standard decoder for row 0b1010 is disconnected. Then, the inputs of a spare decoder are selectively disconnected, so the remaining inputs match the address of 0b1010. In this manner, the spare decoder and the row associated with it take over the storage responsibility for row 0b1010.

A second capability of the decoders shown in Figure 8.15, but not specifically illustrated in Figure 8.16, is the ability to replace a spare decoder with another spare decoder. In cases where the spare row connected to the spare decoder selected to replace row 0b1010 is itself defective, the DRAM device can still be salvaged by disconnecting the spare decoder at its output and programming yet another spare decoder for row 0b1010.

8.6 DRAM Device Control Logic

All DRAM devices contain some basic logic control circuitry to direct the movement of data onto, within, and off of the DRAM device. Essentially, some control logic must exist on DRAM devices that accepts externally asserted signal and control and then orchestrates appropriately timed sequences of internal control signals to direct the movement of data. As an example, the previous discussion on sense amplifier operations hinted to the complexity of the intricate timing sequence in the assertion of the wordline voltage followed by assertion of the SAN and SAP sense amplifier control signals followed yet again by the column-select signal. The sequence of timed control signals is generated by the control logic on DRAM devices.

Figure 8.17 shows the control logic that generates and controls the timing and sequence of signals for the sensing and movement of data on the FPM DRAM device illustrated in Figure 8.1. The control logic on the FPM DRAM device asynchronously accepts external signal control and generates the sequence of internal control signals for the FPM DRAM device.

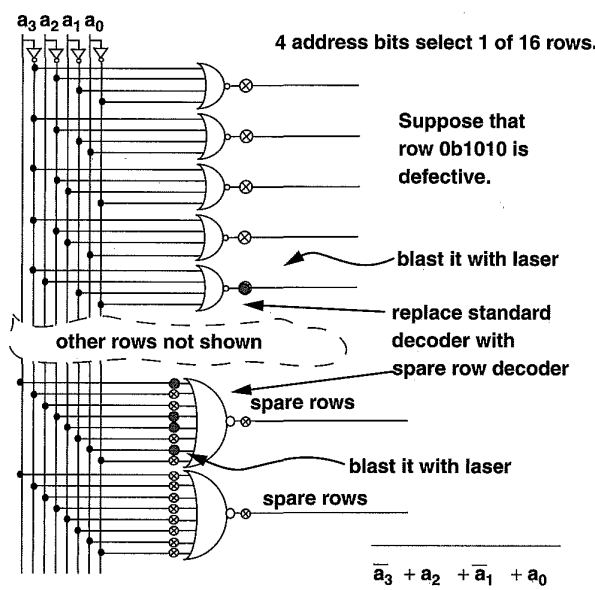


FIGURE 8.16: Standard and spare decoder design.

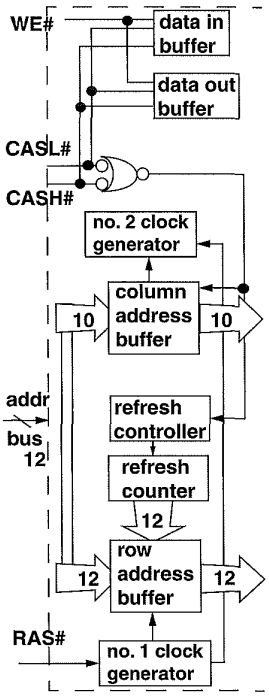


FIGURE 8.17: Control logic for a 32-Mbit, x16-wide FPM DRAM device.

The external interface to the control logic on the FPM DRAM device is simple and straightforward, consisting of essentially three signals: the *row-address strobe* (RAS), the *column-address strobe* (CAS), and the *write enable* (WE). The FPM DRAM device described in Figure 8.1 is a device with a 16-bit-wide data bus, and the use of separate CASL and CASH signals allows the DRAM device to control each half of the 16-bit-wide data bus separately.

In the FPM DRAM device, the control logic and external memory controller directly control the movement of data. Moreover, the controller to the FPM DRAM device interface is an asynchronous interface. In early generations of DRAM devices such as the FPM DRAM described here, the direct control of the internal circuitry of the DRAM device by the external memory controller meant that the DRAM device could not be well pipelined and new commands to the DRAM device could not be initiated until the previous command completed the movement of data. The movement of data was measured and reported by DRAM manufacturers in terms of nanoseconds. The asynchronous nature of the interface meant that system design engineers could implement a different memory controller that operated at different frequencies, and designers of the memory controller were solely responsible to ensure that the controller could correctly control different DRAM devices with subtle variations in timings from different DRAM manufacturers.

8.6.1 Synchronous vs. Non-Synchronous

Modern DRAM devices such as *synchronous DRAM* (SDRAM), *Direct Rambus DRAM* (D-RDRAM), and *dual data rate synchronous DRAM* (DDR SDRAM) contain control logic that is more complex than the control logic contained in an FPM DRAM device. The inclusion of the clock signal into the device interface enables the design of programmable synchronous state machines as the control logic in modern DRAM devices. Figure 8.18 shows the control logic for an SDRAM device.

DRAM circuits are fundamentally analog circuits whose timing is asynchronous in nature. The steps that DRAM circuits take to store and retrieve data in capacitors through the sense amplifier have relatively long latency, and these latencies are naturally specified in terms of nanoseconds rather than numbers of cycles. Moreover, different DRAM designs and process variations from different DRAM manufacturers lead to different sets of timing parameters for each type and design of DRAM devices. The asynchronous nature and the variations of DRAM devices introduce

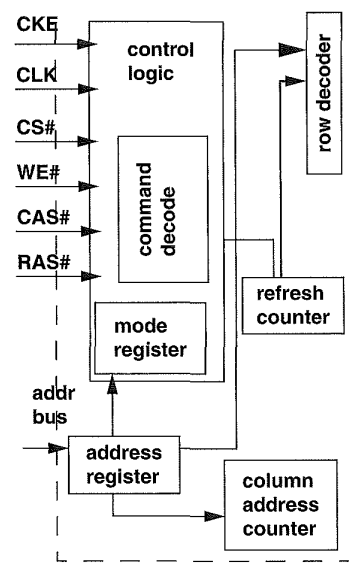


FIGURE 8.18: Control logic for a synchronous DRAM device.

design complexity to computing platforms that use DRAM devices as temporary memory storage. The solution deployed by the DRAM industry as a whole was the migration of DRAM devices to the synchronous interface.

The control logic for synchronous DRAM devices such as SDRAM and D-RDRAM differs from non-synchronous interface DRAM devices such as FPM and EDO in some significant ways. Aside from the trivial inclusion of the clock signal, one difference between control logic for synchronous and previous non-synchronous DRAM devices is that the synchronous DRAM devices can exhibit slight variations in behavior to a given command. The programmable variability for synchronous DRAM devices can be controlled by mode registers embedded as part of the control logic. For example, an SDRAM device can be programmed to return different lengths of data bursts and different data ordering for the column read command. A second difference between the control logic for synchronous DRAM devices and non-synchronous DRAM devices is that the synchronous control logic circuits have been designed to support pipelining naturally, and the ability to support pipelining greatly increases the sustainable bandwidth of the DRAM memory system. Non-synchronous DRAM devices such as EDO and BEDO DRAM can also support pipelining to some degree, but built-in assumptions that enable the limited degree of pipelining in non-synchronous DRAM devices, in turn, limit the frequency scalability of these devices.

8.6.2 Mode Register-Based Programmability

Modern DRAM devices are controlled by state machines whose behavior depends on the input values of the command signals as well as the values contained in the programmable mode register in the control logic. Figure 8.19 shows that in an SDRAM device, the mode register contains three fields: CAS latency, burst type, and burst length. Depending on the value of the CAS latency field in the mode register, the DRAM device returns data two or three cycles after the assertion of the column read command. The value of the burst type determines the ordering of how the SDRAM device returns data, and the burst length field determines the number of columns that an SDRAM device will return to the memory controller with a single column read command. SDRAM devices can be programmed to return 1, 2, 4, or 8 columns or an entire row. D-RDRAM devices and DDRx SDRAM devices contain more mode registers that control an ever larger set of programmable operations, including, but not limited to, different operating modes for power conservation, electrical termination calibration modes, self-test modes, and write recovery duration.

8.7 DRAM Device Configuration

DRAM devices are classified by the number of data bits in each device, and that number typically quadruples from generation to generation. For example, 64-Kbit DRAM devices were followed by 256-Kbit DRAM devices, and 256-Kbit devices were, in turn, followed by 1-Mbit DRAM devices. Recently, half- generation devices that simply double the number of data bits of previous-generation devices have been used to facilitate smoother transitions between different generations. As a result, 512-Mbit devices now exist alongside 256-Mbit and 1 Gbit devices.

In a given generation, a DRAM device may be configured with different data bus widths for use in different applications. Table 8.1 shows three different configurations of a 256-Mbit device. The table shows that a 256-Mbit SDRAM device may be configured with a 4-bit-wide data bus, an 8-bit-wide data bus, or a 16-bit-wide data bus. In the configuration with a 4-bit-wide data bus, an address provided to the SDRAM device to fetch a single column of data will receive 4 bits of data, and there are 64 million separately addressable locations in the device with the 4-bit data bus. The 256-Mbit SDRAM device with the 4-bit-wide data bus is thus referred to as the 64 Meg x4 device. Internally, the 64 Meg x4 device consists of 4 bits of data per column, 2048 columns of data per row, and 8192 rows per bank, and there are 4 banks in the device. Alternatively, a 256-Mbit SDRAM device with a 16-bit-wide data bus will have 16 bits of data per column, 512 columns per row, and

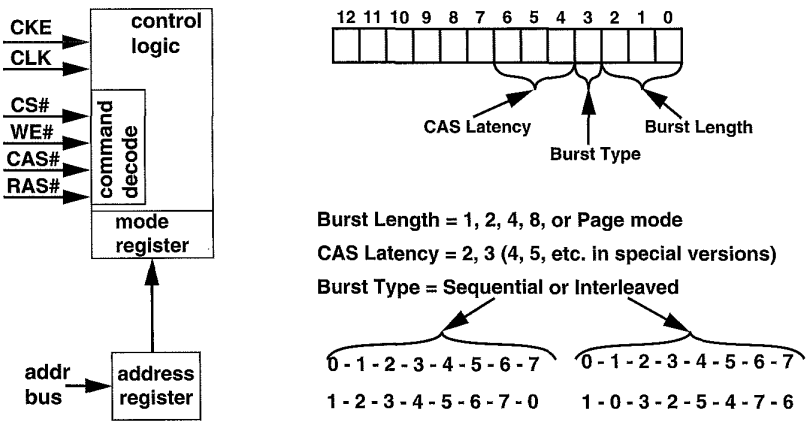


FIGURE 8.19: Programmable mode register in an SDRAM device.

TABLE 8.1 256-Mbit SDRAM device configurations

Device Configuration	64 Meg x 4	32 Meg x 8	16 Meg x 16
Number of banks	4	4	4
Number of rows	8192	8192	8192
Number of columns	2048	1024	512
Data bus width	4	8	16

8192 rows per bank; there are 4 banks in the 16 Mbit, x16 device.

In a typical application, 4 16 Mbit, x16 devices can be connected in parallel to form a single rank of memory with a 64-bit-wide data bus and 128 MB of storage. Alternatively, 16 64 Mbit, x4 devices can be connected in parallel to form a single rank of memory with a 64-bit-wide data bus and 512 MB of storage.

8.7.1 Device Configuration Trade-offs

In the 256-Mbit SDRAM device, the size of the row does not change in different configurations, and the number of columns per row simply decreases with wider data busses specifying a larger number of bits per column. However, the constant row size between different configurations of DRAM devices within the same DRAM device generation is not a generalized trend that can be extended to different device generations. For example, Table 8.2 shows different configurations of a 1-Gbit DDR2 SDRAM device, where the number of bits per row differs between the x8 configuration and the x16 configuration.

In 1-Gbit DDR2 SDRAM devices, there are eight banks of DRAM arrays per device. In the x4 and x8 configuration of the 1-Gbit DDR2 SDRAM device, there are 16,384 rows per bank, and each row consists of 8192 bits. In the x16 configuration, there are 8192 rows, and each row consists of 16,384 bits. These different configurations lead to different numbers of bits per bitline, different numbers of bits per row activation, and different number of bits per column access. In turn, differences in the number of bits moved per command lead to different power consumption

TABLE 8.2 1-Gbit DDR2 SDRAM device configurations

Device Configuration	256 Meg x 4	128 Meg x 8	64 Meg x 16
Number of banks	8	8	8
Number of rows	16,384	16,384	8192
Number of columns	2048	1024	1024
Data bus width	4	8	16

and performance characteristics for different configurations of the same device generation. For example, the 1-Gbit, x16 DDR2 SDRAM device is configured with 16,384 bits per row, and each time a row is activated, 16,384 DRAM cells are simultaneously discharged onto respective bitlines, sensed, amplified, and then restored. The larger row size means that a 1-Gbit, x16 DDR2 SDRAM device with 16,384 bits per row consumes significantly more current per row activation than the x4 and x8 configurations for the 1-Gbit DDR2 SDRAM device with 8192 bits per row. The differences in current consumption characteristics, in turn, lead to different values for t_{RRD} and t_{FAW}, timing parameters designed to limit peak power dissipation characteristics of DRAM devices.

8.8 Data I/O

8.8.1 Burst Lengths and Burst Ordering

In SDRAM and DDRx SDRAM devices, a column read command moves a variable number of columns. As illustrated in Section 8.6.2 on the programmable mode register, an SDRAM device can be programmed to return 1, 2, 4, or 8 columns of data as a single burst that takes 1, 2, 4, or 8 cycles to complete. In contrast, a D-RDRAM device returns a single column of data with an 8 beat[12] burst. Figure 8.20 shows an 8 beat, 8 column read data burst from an SDRAM device and an 8 beat, single column read data burst from a D-RDRAM device. The distinction between the 8 column burst of an SDRAM device and the single column data burst of the D-RDRAM device is that each column of the SDRAM device is individually addressable, and given a column address in the middle of an 8 column burst, the SDRAM device will reorder the burst to provide the data of the requested address first. This capability is known as critical-word forwarding. For example, in an SDRAM device programmed to provide a burst of 8 columns, a column read command with a column address of 17 will result in the data burst of 8 columns of data with the address sequence of 17-18-19-20-21-22-23-16 or 17-16-19-18-21-20-23-22, depending on the burst type as defined in the programmable register. In contrast, each column of a D-RDRAM device consists of 128 bits of data, and each column access command moves 128 bits of data in a burst of 8 contiguous beats in strict burst ordering. An D-RDRAM device supports neither programmable burst lengths nor different burst ordering.

8.8.2 N-Bit Prefetch

In SDRAM devices, each time a column read command is issued, the control logic determines the duration and ordering of the data burst, and each column is moved separately from the sense amplifiers through the I/O latches to the external data bus. However, the separate control of each column limits the operating data rate of the DRAM device. As a result, in DDRx SDRAM devices, successively larger numbers of bits are moved in parallel from the sense amplifiers to the read latch, and the data is then pipelined through a multiplexor to the external data bus.

[12]In DDRx and D-RDRAM devices, 2 beats of data are transferred per clock cycle.

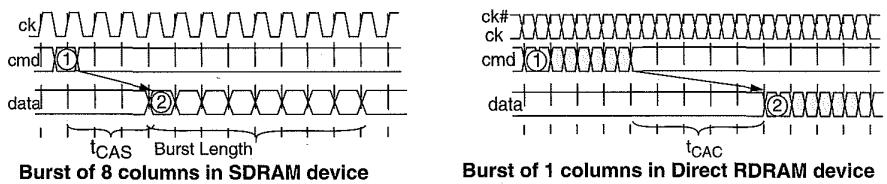


FIGURE 8.20: Burst lengths in DRAM devices.

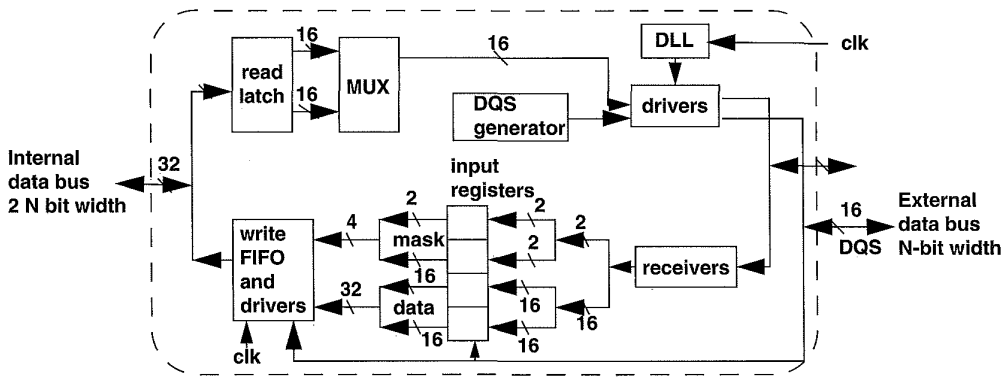


FIGURE 8.21: Data I/O in a DDR SDRAM device illustrating 2-bit prefetch.

Figure 8.21 illustrates the data I/O structure of a DDR SDRAM device. The figure shows that given the width of the external data bus as N, 2N bits are moved from the sense amplifiers to the read latch, and the 2N bits are then pipelined through the multiplexors to the external data bus. In DDR2 SDRAM devices, the number of bits prefetched by the internal data bus is 4N. The N-bit prefetch strategy in DDRx SDRAM devices means that internal DRAM circuits can remain essentially unchanged between transitions from SDRAM to DDRx SDRAM, but the operating data rate of DDRx SDRAM devices can be increased to levels not possible with SDRAM devices. However, the downside of the N-bit prefetch architecture means that short column bursts are no longer possible. In DDR2 SDRAM devices, a minimum burst length of 4 columns of data is accessed per column read command. This trend is likely to continue in DDR3 and DDR4 SDRAM devices, dictating longer data bursts for each successive generations of higher data rate DRAM devices.

8.9 DRAM Device Packaging

One difference between DRAM and logic devices is that most DRAM devices are commodity items, whereas logic devices such as processors and *application-specific integrated circuits* (ASICs) are typically specialized devices that are not commodity items. The result of the commodity status is that, even more so than logic devices, DRAM devices are extraordinarily sensitive to cost. One area that

TABLE 8.3 Package cost and pin count of high-performance logic chips and DRAM chips (ITRS 2002)

	2004	2007	2010	2013	2016
Semi generation (nm)	90	65	45	32	22
High perf. device pin count	2263	3012	4009	5335	7100
High perf. device cost (cents/pin)	1.88	1.61	1.68	1.44	1.22
Memory device pin count	**48–160**	**48–160**	**62–208**	**81–270**	**105–351**
DRAM device pin cost (cents/pin)	0.34–1.39	0.27–0.84	0.22–0.34	0.19–0.39	0.19–0.33

reflects the cost sensitivity is the packaging technology utilized by DRAM devices. Table 8.3 shows the expected pin count and relative costs from the 2002 *International Technology Roadmap for Semiconductors* (ITRS) for high-performance logic devices as compared to memory devices. The table shows the trend that memory chips such as DRAM will continue to be manufactured with relatively lower cost packaging with lower pin count and lower cost per pin.

Figure 8.22 shows four different packages used in DRAM devices. DRAM devices were typically packaged in low pin count and low cost *Dual In-line Packages* (DIP) well into the late 1980s. Increases in DRAM device density and wider data paths have required the use of the larger and higher pin count *Small Outline J-lead* (SOJ) packages. DRAM devices then moved to the *Thin, Small Outline Package* (TSOP) in the late 1990s. As DRAM device data rates increase to multiple hundreds of megabits per second, *Ball Grid Array* (BGA) packages are needed to better control signal interconnects at the package level.

8.10 DRAM Process Technology and Process Scaling Considerations

The 1T1C cell structure places specialized demands on the access transistor and the storage capacitor. Specifically, the area occupied by the 1T1C DRAM cell structure must be small, leakage through the access transistor must be low, and the capacitance of the storages capacitor must be large. The data retention time and data integrity requirements provide the bounds for the design of a DRAM cell. Different DRAM devices can be designed to meet the demand of different markets. DRAM devices can be designed for high performance or low cost. DRAM-optimized process technologies can also be used to fabricate logic circuits, and logic-optimized process technologies can also be used to fabricate DRAM circuits. However, DRAM-optimized process technologies have diverged substantially from logic-optimized process technologies in recent years. Consequently, it has become less economically feasible to fabricate DRAM circuits in logic-optimized process technology, and logic circuits fabricated in DRAM-optimized

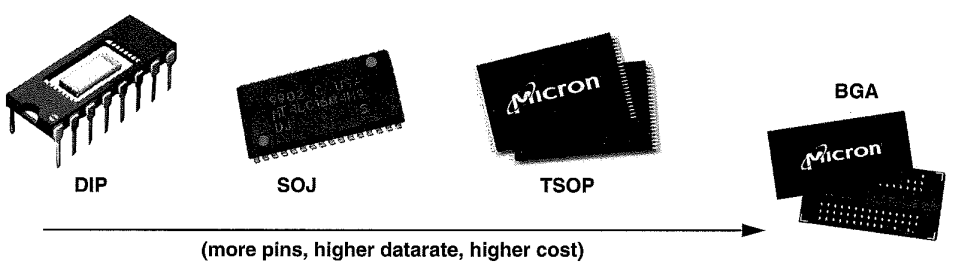


FIGURE 8.22: DRAM package evolution.

process technology are much slower than similar circuits in a logic-optimized process technology. These trends have conspired to keep logic and DRAM circuits in separate devices manufactured in process technologies optimized for the respective device.

8.10.1 Cost Considerations

Historically, manufacturing cost considerations have dominated the design of standard, commodity DRAM devices. In the spring of 2003, a single 256-Mbit DRAM device, using roughly 45 mm^2 of silicon die area on a 0.11-μm DRAM process, had a selling price of approximately $4 per chip. In contrast, a desktop Pentium 4 processor from Intel, using roughly 130 mm^2 of die area on a 0.13-μm logic process, had a selling price that ranged from $80 to $600 in the comparable time-frame. Although the respective selling prices were due to the limited sources, the non-commodity nature of processors, and the pure commodity economics of DRAM devices, the disparity does illustrate the level of price competition in the commodity DRAM market. The result is that DRAM manufacturers are singularly focused on the low-cost aspect of DRAM devices. Any proposal to add additional functionalities must then be weighed against the increase in die cost and possible increases in the selling price.

8.10.2 DRAM- vs. Logic-Optimized Process Technology

One trend in semiconductor manufacturing is the inevitable march toward integration. As the semiconductor manufacturing industry dutifully fulfills Moore's Law, each doubling of transistors allows design engineers to pack more logic circuitry or more DRAM storage cells onto a single piece of silicon. However, the semiconductor industry, in general, has thus far resisted the integration of DRAM and logic onto the same silicon device for various technical and economic reasons.

Figure 8.23 illustrates some technical issues that have prevented large-scale integration of logic circuitry

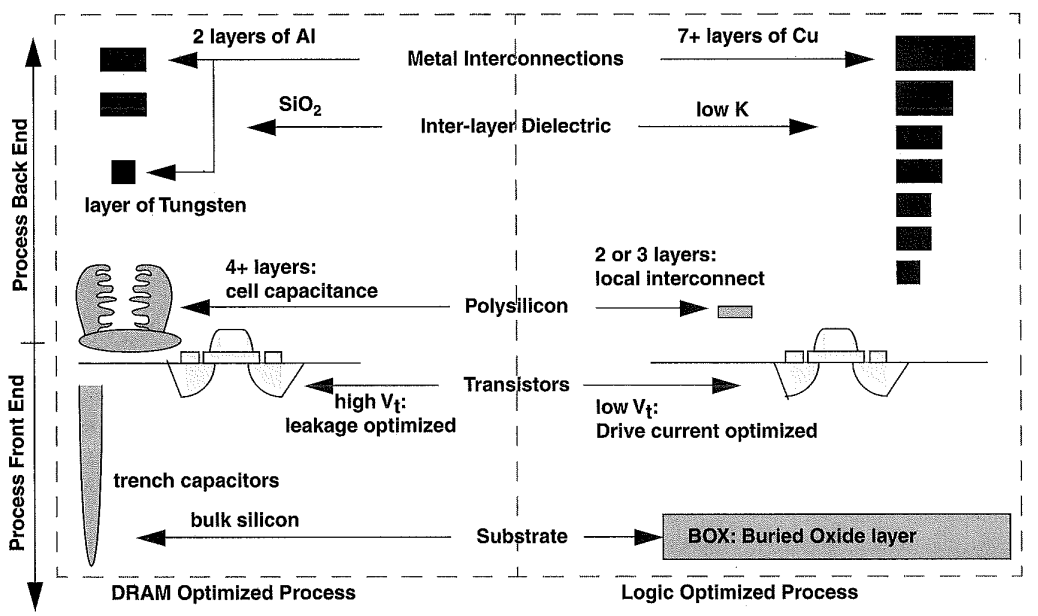


FIGURE 8.23: Comparison of a DRAM-optimized process versus a logic-optimized process.

with DRAM storage cells. Basically, logic-optimized process technologies have been designed for transistor performance, while DRAM-optimized process technologies have been designed for low cost, error tolerance, and leakage resistance. Figure 8.23 shows a typical logic-based process with seven or more layers of copper interconnects, while a typical DRAM-optimized process has only two layers of aluminum interconnects along with perhaps an additional layer of tungsten for local interconnects. Moreover, a logic-optimized process typically uses low K material for the inter-layer dielectric, while the DRAM-optimized process uses the venerable SiO_2. Figure 8.23 also shows that a DRAM-optimized process would use four or more layers of polysilicon to form the structures of a stacked capacitor (for those DRAM devices that use the stacked capacitor structure), while the logic-optimized process merely uses two or three layers of polysilicon for local interconnects. Also, transistors in a logic-optimized process are typically tuned for high performance, while transistors in a DRAM-optimized process are tuned singularly for low-leakage characteristics. Finally, even the substrates of the respectively optimized process technologies are diverging as logic-optimized process technologies move to depleted substrates and DRAM-optimized process technologies largely stays with bulk silicon.

The respective specializations of the differently optimized process technologies have largely succeeded in preventing widespread integration of logic circuitry with DRAM storage cells. The use of a DRAM-optimized process as the basis of integrating logic circuits and DRAM storage cells leads to slow transistors with low-drive currents connected to a few layers of metal interconnects and a relatively high K SiO_2 inter-layer dielectric. That is, logic circuits implemented on a DRAM-optimized process would be substantially larger as well as slower than comparable circuits on a similar generation logic-optimized process. Conversely, the use of a higher cost logic-optimized process as the basis of integrating logic circuits and DRAM storage cells leads to high-performance but leaky transistors coupled with DRAM cells with relatively lower capacitance, necessitating large DRAM cell structures and high refresh rates.

In recent years, new hybrid process technologies have emerged to solve various technical issues involving the integration of logic circuits and DRAM storage cells. Typically, the hybrid process starts with the foundation of a logic-optimized process and then additional layers are added to the process to create high-capacitance DRAM storage cells. Also, different types of transistors are made available for use as low-leakage access transistors as well as high-drive current high-performance logic transistors. However, hybrid process technology then becomes more complex than a logic-optimized process. As a result, hybrid process technologies that enable seamless integration of logic and DRAM devices are typically more expensive, and their use has thus far been limited to specialty niches that require high-performance processors and high-performance and yet small DRAM memory systems that are limited by the die size of a single logic device. Typically, the application has been limited to high-performance System-on-Chip (SOC) devices.

CHAPTER 9

DRAM System Signaling and Timing

In any electronic system, multiple devices are connected together, and signals are sent from one point in the system to another point in the system for the devices to communicate with each other. The signals adhere to predefined signaling and timing protocols to ensure correctness in the transmission of commands and data. In the grand scale of things, the topics of signaling and timing require volumes of dedicated texts for proper coverage. This chapter cannot hope to, nor is it designed to, provide a comprehensive coverage on these important topics. Rather, the purpose of this chapter is to provide basic terminologies and understanding of the fundamentals of signaling and timing—subjects of utmost importance that drive design decisions in modern DRAM memory systems. This chapter provides the basic understanding of signaling and timing in modern electronic systems and acts as a primer for further understanding of the topology, electrical signaling, and protocols of modern DRAM memory systems in subsequent chapters. The text in this chapter is written for those interested in the DRAM memory system but do not have a background as an electrical engineer, and it is designed to provide a basic survey of the topic sufficient only to understand the system-level issues that impact the design and implementation of DRAM memory systems, without having to pick up another text to reference the basic concepts.

9.1 Signaling System

In the decades since the emergence of electronic computers, the demand for ever-increasing memory capacity has constantly risen. This insatiable demand for memory capacity means that the number of DRAM devices attached to the memory system for a given class of computers has remained relatively constant despite the increase in per-device capacity made possible with advancements in semiconductor technology. The need to connect multiple DRAM devices together to form a larger memory system for a wide variety of computing platforms has remained unchanged for many years. In the cases where multiple, discrete DRAM devices are connected together to form larger memory systems, complex signaling systems are needed to transmit information to and from the DRAM devices in the memory system.

Figure 9.1 illustrates the timing diagram for two consecutive column read commands to different DDR SDRAM devices. The timing diagram shows idealized timing waveforms, where data is moved from the DRAM devices in response to commands sent by the DRAM memory controller. However, as Figure 9.1 illustrates, signals in real-world systems are far from ideal, and signal integrity issues such as ringing, attenuation, and non-monotonic signals can and do negatively impact the setup and hold time requirements of signal timing constraints. Specifically, Figure 9.1 illustrates that a given signal may be considered high-quality if it transitions rapidly and settles rapidly from one signal level to another. Figure 9.1 further illustrates that a poorly designed signaling system can result in a poor quality signal that overshoots, undershoots, and does not settle rapidly into its new signal value, possibly resulting in the violation of the setup time or the hold time requirements in a high-speed system.

Figure 9.2 illustrates the fundamental problem of frequency-dependent signal transmission in a

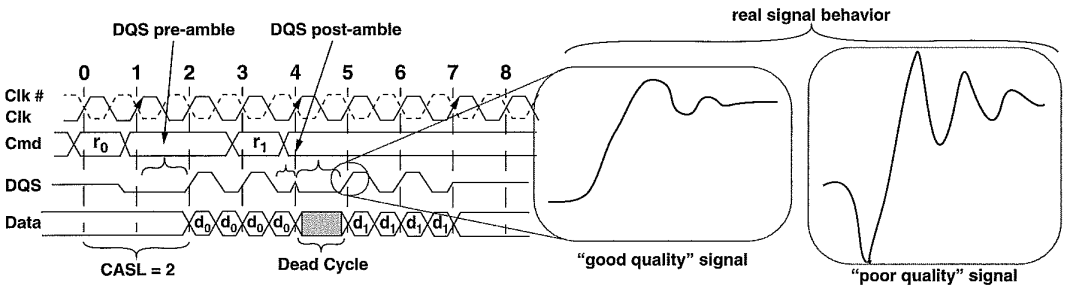


FIGURE 9.1: Real-world behavior of electrical signals.

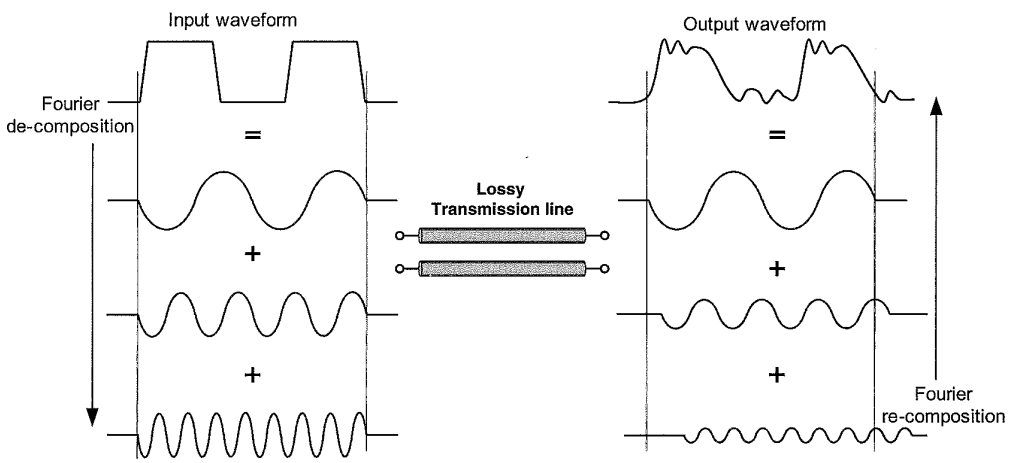


FIGURE 9.2: Frequency-dependent signal transmission in lossy, real-world transmission lines.

lossy, real-world transmission line. That is, an input waveform, even an idealized, perfect square wave signal, can be decomposed into a Fourier series—a sum of sinusoidal and cosinusoidal oscillations of various amplitudes and frequencies. The real-world, lossy transmission line can then be modelled as a non-linear low-pass filter where the low-frequency components of the Fourier decomposition of the input waveform pass through the transmission line without substantial impact to their respective amplitudes or phases. In contrast, the non-linear, low-pass transmission line will significantly attenuate and phase shift the high-frequency components of the input waveform. Then, recomposition of the various frequency components of the input waveform at the output of the lossy transmission line will result in an output waveform that is significantly different from that of the input waveform.

Collectively, constraints of the signaling system will limit the signaling rate and the delivery of symbols between discrete semiconductor devices such as DRAM devices and the memory controller. Consequently, the

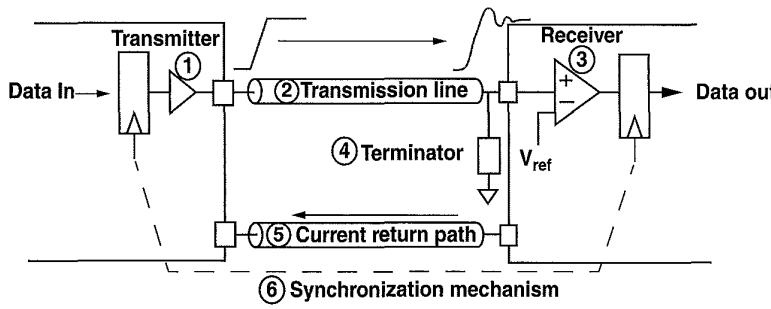


FIGURE 9.3: A basic signaling system.

construction and assumptions inherent in the signaling system can and do directly impact the access protocol of a given memory system, which in turn determines the bandwidth, latency, and efficiency characteristics of the memory system. As a result, a basic comprehension of issues relating to signaling and timing is needed as a foundation to understand the architectural and engineering design trade-offs of modern, multi-device memory systems.

Figure 9.3 shows a basic signaling system where a symbol, encoded as a signal, is sent by a transmitter along a transmission line and delivered to a receiver. The receiver must then resolve the value of the signal transmitted within valid timing windows determined by the synchronization mechanism. The signal should then be removed from the transmission line by a resistive element, labelled as the terminator in Figure 9.3, so that it does not interfere with the transmission and reception of subsequent signals. The termination scheme should be carefully designed to improve signal integrity depending on the specific interconnect scheme. Typically, serial termination is used at the receiver and parallel termination is used at the transmitter in modern high-speed memory systems. As a general summary, serial termination reduces signal ringing at the cost of reduced signal swing at the receiver, and parallel termination improves signal quality, but consumes additional active power to remove the signal from the transmission line.

Figure 9.3 illustrates a basic signaling system where signals are delivered unidirectionally from a transmitter to a single receiver. In contemporary DRAM memory systems, signals are often delivered to multiple DRAM devices connected on the same transmission line. Specifically, in SDRAM and SDRAM-like DRAM memory systems, multiple DRAM devices are often connected to a given address and command bus, and multiple DRAM devices are often connected to the same data bus where the same transmission line is used to move data from the DRAM memory controller to the DRAM devices, as well as from the DRAM devices back to the DRAM memory controller.

The examination of the signaling system in this chapter begins with an examination of basic transmission line theory with the treatment of wires as ideal transmission lines, and it proceeds to an examination of the termination mechanism utilized in the DRAM memory system. However, due to their relative complexity and the limited coverage envisioned in this chapter, specific circuits utilized by DRAM devices for signal transmission and reception are not examined herein.

9.2 Transmission Lines on PCBs

Modern DRAM memory systems are typically formed from multiple devices mounted on *printed circuit boards* (PCBs). The interconnects on PCBs are mostly wires and vias that allow an electrical signal to deliver a symbol from one point in the system to another point in the system. The symbol may be

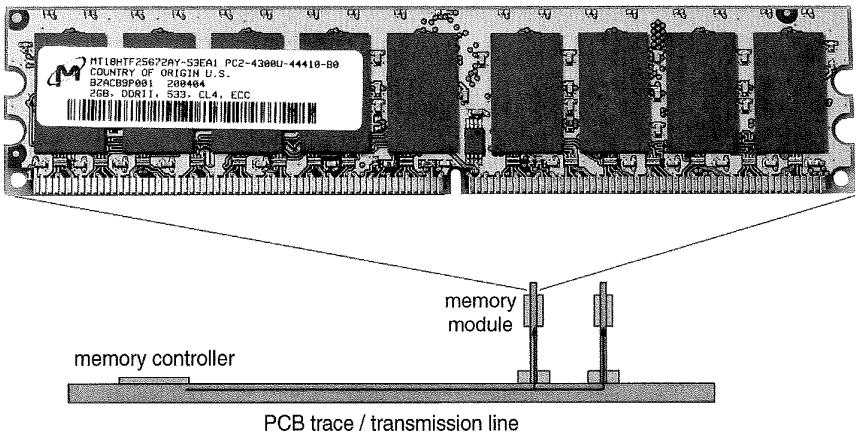


FIGURE 9.4: Signal traces in a system board and a DRAM memory module.

binary (1 or 0) as in all modern DRAM memory systems or may, in fact, be multi-valued, where each symbol can represent two or more bits of information. The limitation on the speed and reliability of the data transport mechanism in moving a symbol from one point in the system to another point in the system depends on the quality and characteristics of the traces used in the system board. Figure 9.4 illustrates a commodity DRAM memory module, where multiple DRAM devices are connected to a PCB, and multiple memory modules are connected to the memory controller through more PCB traces on the system board. In contemporary DRAM memory systems, multiple memory modules are then typically connected to a system board where electrical signals that represent different symbols are delivered to the devices in the system through traces on the different PCB segments.

In this section, the electrical properties of signal traces are examined by first characterizing the electrical characteristics of idealized transmission lines. Once the characteristics of idealized transmission lines have been established, the discussion then proceeds to examine the non-idealities of signal transmission on a system board such as *attenuation, reflection, skin effect, crosstalk, inter-symbol interference* (ISI), and *simultaneous switching outputs* (SSO). The coverage of these basic topics will then enable the reader to proceed to understand system-level design issues in modern, high-speed DRAM memory systems.

9.2.1 Brief Tutorial on the Telegrapher's Equations

To begin the examination of the electrical characteristics of signal interconnects, an understanding of transmission line characteristics is a basic requirement. This section provides the derivation of the telegrapher's equations that will be used as the basis of understanding the signal interconnect in this chapter.

Figure 9.5 illustrates that an infinitesimally small piece of transmission line can be modelled as a resistive element R that is in series with an inductive element L, and these elements are parallel to a capacitive element C and a conductive element G. To understand the electrical characteristics of the basic transmission line, Kirchhoff's voltage law can be applied to the transmission line segment in

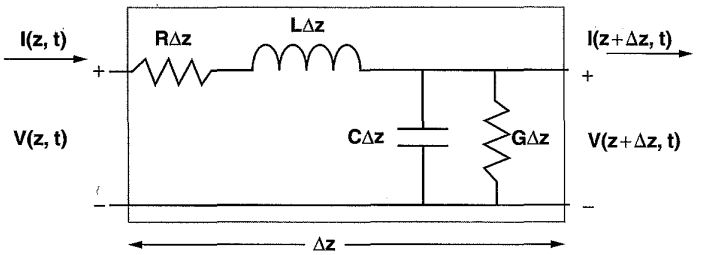


FIGURE 9.5: Mathematical model of a basic transmission line. Top wire (+) is signal; bottom wire (−) is signal return.

Figure 9.5 to obtain Equation 9.1,

$$v(z, t) - (R \cdot \Delta z \cdot i(z, t)) - \left(L \cdot \Delta z \cdot \frac{\partial i(z, t)}{\partial t}\right) - v(z + \Delta z, t) = 0 \quad \textbf{(EQ 9.1)}$$

and Kirchhoff's current law can be applied to the same transmission line segment in Figure 9.5 to obtain Equation 9.2.

$$i(z, t) - (G \cdot \Delta z \cdot v(z, + \Delta z, t)) - \left(C \cdot \Delta z \cdot \frac{\partial v(z + \Delta z, t)}{\partial t}\right) - i(z + \Delta z, t) = 0 \quad \textbf{(EQ 9.2)}$$

Then, dividing Equations 9.1 and 9.2 through by Δz, and taking the limit as Δz approaches zero, Equation 9.3 can be derived from Equation 9.1, and Equation 9.4 can be derived from Equation 9.2.

$$\frac{\partial v(z, t)}{\partial t} = -Ri(z, t) - L\frac{\delta i(z, t)}{\delta t} \quad \textbf{(EQ 9.3)}$$

$$\frac{\partial i(z, t)}{\partial t} = -Gv(z, t) - C\frac{\delta v(z, t)}{\delta t} \quad \textbf{(EQ 9.4)}$$

Equations 9.3 and 9.4 are time-domain equations that describe the electrical characteristics of the transmission line. Equations 9.3 and 9.4 are also known as the Telegrapher's Equations.[1]

Furthermore, Equations 9.3 and 9.4 can be solved simultaneously to obtain steady-state sinusoidal wave equations. Equations 9.5 and 9.6 are derived from Equations 9.3 and 9.4, respectively,

$$\frac{\partial^2 V(z)}{\partial z^2} - \gamma^2 V(z) = 0 \quad \textbf{(EQ 9.5)}$$

$$\frac{\partial^2 I(z)}{\partial z^2} - \gamma^2 V(z) = 0 \quad \textbf{(EQ 9.6)}$$

where γ is represented by Equation 9.7.

$$\gamma = \sqrt{(R + j\omega L)(G + j\omega C)} \quad \textbf{(EQ 9.7)}$$

Furthermore, solving for voltage and current equations, Equations 9.8 and 9.9 are derived

$$V(z) = V_0^+ e^{-\gamma z} + V_0^- e^{-\gamma z} \quad \textbf{(EQ 9.8)}$$

$$I(z) = I_0^+ e^{-\gamma z} + I_0^- e^{-\gamma z} \quad \textbf{(EQ 9.9)}$$

where V_0^+ and V_0^- are the respective voltages, and I_0^+ and I_0^- are the respective currents that exist at locations Z^+ and Z^-, locations infinitesimally close to reference location Z. That is, Equations 9.8 and 9.9 are the standing wave equations the describe transmission line characteristics.

Finally, rearranging Equations 9.8 and 9.9, Equations 9.10 and 9.11 can be derived

$$I(z) = \frac{\gamma}{R + j\omega L} \cdot (V_0^+ e^{-\gamma z} + V_0^- e^{-\gamma z}) \quad \textbf{(EQ 9.10)}$$

$$Z_0 = \frac{R + j\omega L}{\gamma} = \sqrt{\frac{R + j\omega L}{G + j\omega C}} \quad \textbf{(EQ 9.11)}$$

where Z_0 is the characteristic impedance, and α and β are the attenuation constant and the phase constant of

[1]The Telegrapher's Equations were first derived by William Thomson in the 1850s in his efforts to analyze the electrical characteristics of the underwater telegraph cable. Their final form, shown here, was later derived by Oliver Heaviside.

the transmission line, respectively. Although the characteristic impedance of the transmission line has the unit of ohms, it is conceptually different from simple resistance. Rather, the characteristic impedance is the resistance seen by propagating waveforms at a specific point of the transmission line.

9.2.2 RC and LC Transmission Line Models

Equations 9.1–9.11 illustrate a mathematical derivation of basic transmission line characteristics. However, system designer engineers are often interested in transmission line behavior within relatively narrow frequency bands rather than the full frequency spectrum. Consequently, simpler high-frequency LC (Inductor-Capacitor) or low-frequency RC (Resistor-Capacitor) models are often used in place of the generalized model. Figure 9.6 shows the same model for an infinitesimally small piece of transmission line as in Figure 9.5, but a closer examination of the characteristic impedance equation reveals that the equation can be simplified if the magnitude of the resistive component R is much smaller than or much greater than the magnitude of the frequency-dependent inductive component $j\omega L$. That is, in the case where the magnitude of R greatly exceeds $j\omega L$, the characteristic impedance of the transmission line can be simplified as $Z_0 = \text{SQRT}(R/j\omega C)$,[2] hereafter referred to as the RC model. Conversely, in the case where the magnitude of the frequency-dependent inductive component $j\omega L$ greatly exceeds the magnitude of the resistive component R, the characteristic impedance of the transmission line can be simplified as $Z_0 = \text{SQRT}(L/C)$, hereafter referred to as the LC model. Given that the general transmission line model can be simplified into the LC model or the RC model, the key to choosing the correct model is to compute the characteristic frequency f_0 for the transmission line where the resistance R equals $j\omega L$. The characteristic frequency f_0 can be computed with the equation $f_0 = R / 2\pi L$. The simple rule of thumb that can be used is that for operating frequencies much above f_0, system designer engineers can assume a simplified LC transmission line model, and for operating frequencies much below f_0, system designer engineers can assume a simplified RC transmission line model. Due to the fact that signal paths on silicon are highly resistive, the characteristic frequency f_0 is much lower for silicon interconnects than system interconnects. As a result, the RC model is typically used for silicon interconnects on silicon, and the LC model is typically used for package-level and system-level interconnects.

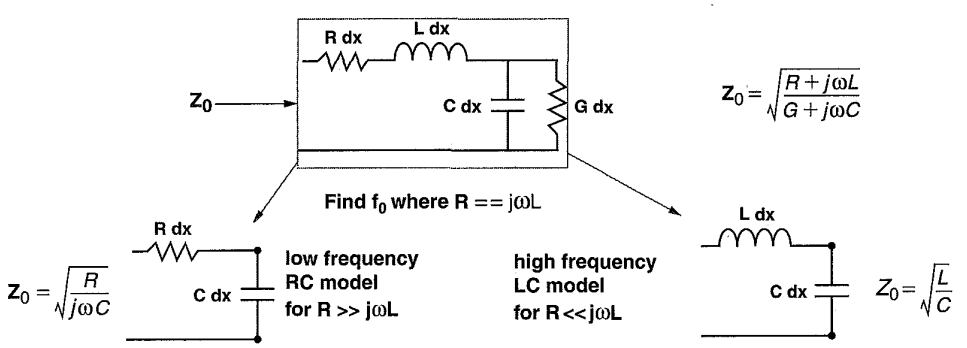


FIGURE 9.6: Simplified transmission line models.

[2]The conductive element G is assumed to be much smaller than jωC.

9.2.3 LC Transmission Line Model for PCB Traces

Figure 9.7 shows the cross section of a six-layer PCB. The six-layer PCB consists of signaling layers on the top and bottom layers with two more signaling layers sandwiched between two metal planes devoted to power and ground. Typically, inexpensive PC systems use only four-layer PCBs due to cost considerations, while more expensive server systems and memory modules typically use PCBs with six, eight, or more layers (on some systems, upwards of twenty-plus layers) for better signal shielding, signal routing, and power supplies.

Figure 9.7 also shows a close-up section of a trace on the uppermost signaling layer and the respective electrical characteristics of that signal trace. Given the resistive component of the signal trace as 0.003 Ω/mm and the inductance component of the signal trace as 0.25 nH/mm, the characteristic frequency of the signal trace can be computed as 1.9 MHz. That is, the signal traces on the illustrated PCB can be modelled effectively—to the first order—by relying only on the LC characteristics of the transmission line, since the edge transition frequencies of signals in contemporary DRAM memory systems are considerably higher than 1.9 MHz.[3]

9.2.4 Signal Velocity on the LC Transmission Line

The electrical characteristics of the transmission line derived in Equations 9.1–9.11 and discussed in the previous section assert that typical PCB traces in modern DRAM memory systems can be typically modelled as LC transmission lines. In this and the following sections, properties of typical PCB traces are further examined to qualify signal traces found on contemporary PCBs from idealized LC transmission lines.

Figure 9.8 illustrates two important characteristics of the ideal LC transmission line: the wave velocity and the superposition property of signals that travel on the transmission line. In an ideal LC transmission line, the resistive element is assumed to be negligible, and signals can theoretically propagate down an ideal LC transmission line without attenuation. Figure 9.8 also shows that the signal propagation speed on an ideal lossless LC transmission line is a function of the

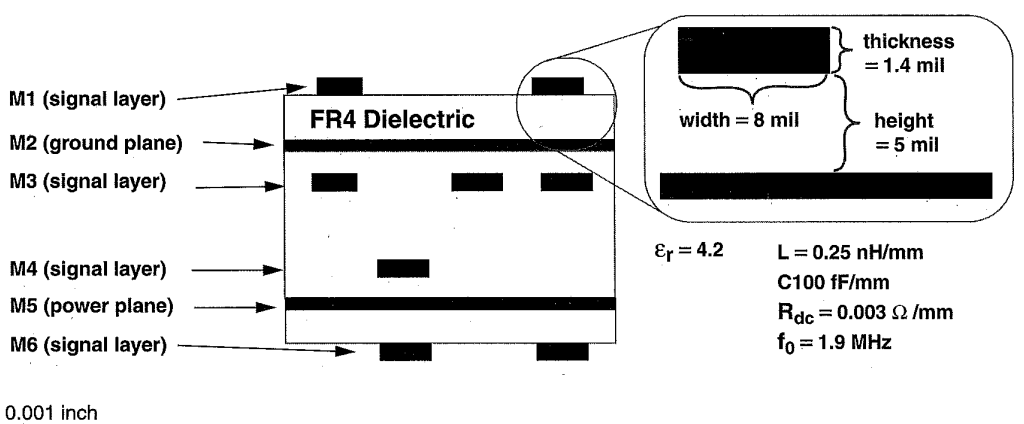


FIGURE 9.7: Derivation of characteristic frequency for PCB traces.

[3]The frequency of interest here is not the operating frequency of the signals transmitted in DRAM memory systems, but the high-frequency components of the signals as they transition from one state to another.

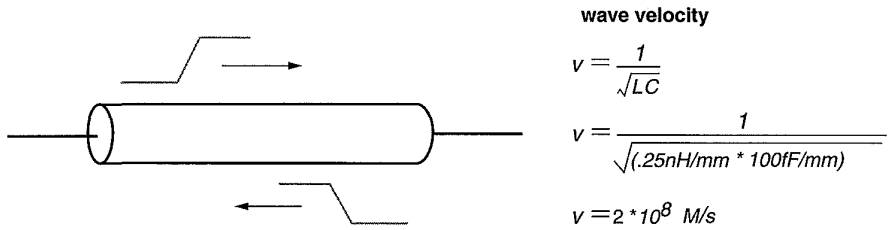


FIGURE 9.8: Idealized LC transmission line.

impedance characteristics of the LC transmission line, and signal velocity can be computed with the equation 1/SQRT(LC). Substituting in the capacitance and inductance values from Figure 9.7, the wave velocity of an electrical signal as it propagates on the specific transmission line is computed to be 200,000,000 M/s. That is, on the transmission line with the specific impedance characteristics, signal wave-fronts propagate at two-thirds the speed of light in vacuum, and signal wave-fronts travel a distance of 20 cm/ns.

Finally, in the lossless LC transmission line, signals can propagate down the transmission line without interference from signals propagating in the opposite direction. In this manner, the transmission line can support bidirectional signaling, and the voltage at a given point on the transmission line can be computed by the instantaneous superposition of the signals propagating on the transmission line.

9.2.5 Skin Effect of Conductors

The skin depth effect is a critical component of signal attenuation. That is, the resistance of a conductor is typically a function of the cross-sectional area of the conductor, and the resistance of a signal trace is typically held as a constant in the computation of transmission line characteristics. However, one interesting characteristic of conductors is that electrical current does not flow uniformly throughout the cross section of the conductor at high frequencies. Instead, current flow is limited to a certain depth of the conductor cross section when the signal is switching rapidly at high frequencies. The frequency-dependent current penetration depth is illustrated as the skin depth in Figure 9.9. The net effect of the limited current flow in conductors at high frequencies is that resistance of a conductor increases as a function of frequency. The skin effect of conductors further illustrates that a lossless ideal LC transmission line cannot completely model a real-world PCB trace.

9.2.6 Dielectric Loss

The LC transmission line is often used as a first-order model that approximates the characteristics of traces on a system board. However, real-world signal traces are certainly not ideal lossless LC transmission lines, and signals do attenuate as a function of trace length and frequency. Figure 9.10 illustrates signal attenuation through a PCB trace as a function of trace length and data rate. The figure shows that signal attenuation increases at higher data rates and longer trace lengths. In the context of a DRAM memory system, a trace that runs for 10 inches and operates at 500 Mbps will lose less than 5% of the peak-to-peak signal strength. Moreover, Figure 9.10 shows that signal attenuation remains below 10% for the 10" trace that operates at data rates upward to 2.5 Gbps. In this sense, the issue of signal attenuation is a manageable issue for DRAM memory systems that operate at relatively modest data rates and relatively short trace lengths. However, the issue of signal attenuation is only one of several issues that system design engineers must account for in the design of high-speed DRAM memory systems and one of several issues that ensures the lossless LC transmission line model remains only as an approximate first-order model for PCB traces.

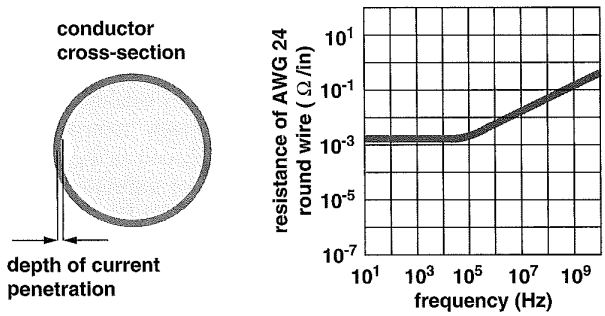
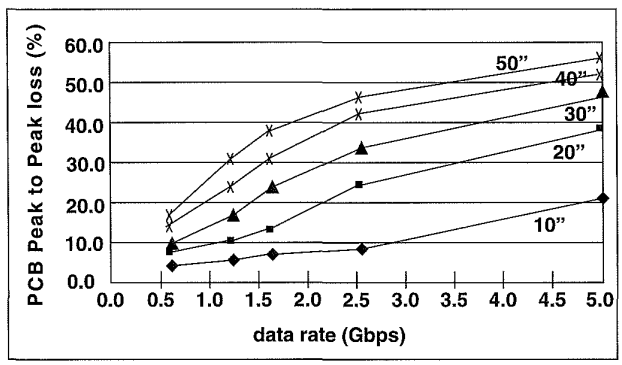


FIGURE 9.9: Illustration of skin depth in cross section of a circular conductor.



100 Ω differential PCB loss, as function of data rate, PCB lengths from 10" to 50" (8 mil line width)

FIGURE 9.10: Attenuation factor of a signal on a PCB trace. Graph taken from North East Systems Associates Inc. Copyright 2002, North East Systems Associates Inc. with permission.

One common way to model dielectric loss is to place a distributed shunt conductance across the line whose conductivity is proportional to frequency. Most transmission lines exhibit quite a large metallic loss from the skin effect at frequencies well below those where dielectric loss becomes important. In most modern memory systems, signal reflection is typically a far more serious problem for the relatively short trace length, multi-drop signaling system. For these reasons, dielectric loss is often ignored, but its onset is very fast when it does happen. For DDR2 SDRAM and other lower speed memory systems (<1 Gbps), dielectric loss is not a dominant effect and can be generally ignored in the signaling analysis. However, in higher data rate memory systems such as the Fully Buffered DIMM and DDR3 memory systems, dielectric loss should be considered for complete signaling analysis.

The physics of the dielectric loss can be described with basic electron theory. The dielectric material between the conductors is an insulator, and electrons that orbit the atoms in the dielectric material

are locked in place in the insulating material. When there is a difference in potential between two conductors, the excessive negative charge on one conductor repels electrons on the dielectric toward the positive conductor and disturbs the orbits of the electrons in the dielectric material. A change in the path of electrons requires more energy, resulting in the loss of energy in the form of attenuating voltage signals.

9.2.7 Electromagnetic Interference and Crosstalk

In an electrical signaling system, the movement of a voltage signal delivered through a transmission line involves the displacement of electrons in the direction of, or opposite to the direction of, the voltage signal delivery. The conservation of charge, in turn, means that as current flows in one direction, there must exist a current that flows in the opposite direction. Figure 9.11 shows that current flow on a transmission line must be balanced with current flow through a current return path back to the originating device. Collectively, the signal current and the return current form a closed-circuit loop. Typically, the return current flow occurs through the ground plane or an adjacent signal trace. In effect, a given signal and its current return path form a basic current loop where the magnitude of the current flow in the loop and the area of the loop determine the magnitude of the *Electromagnetic Interference* (EMI). In general, EMI generated by the delivery of a signal in a system creates inductive coupling between signal traces in a given system. Additionally, signal traces in a closely packed system board will also be susceptible to capacitive coupling with adjacent signal traces.

In this section, electronic noises injected into a given trace by signaling activity from adjacent traces are collectively referred to as *crosstalk*. Crosstalk may be induced as the result of a given signal trace's capacitive or inductive coupling to adjacent traces. For example, in the case where a signal and its presumed current return path (signal ground) are not routed closely to each other, and a different trace is instead routed closer to the signal trace than the current return path of the signal trace, this adjacent (victim) trace will be susceptible to EMI that emanates from the poorly designed current loop, resulting in crosstalk between the signal traces. The issue of crosstalk further deviates the modelling of real-world system board traces from the idealities of the lossless LC transmission line, where closely routed signal traces can become attackers on their neighboring signal traces and significantly impact the timing and signal integrity of these neighboring traces. The magnitude of the crosstalk injected by capacitively or inductively coupled signal traces depends on the peak-to-peak voltage swings of the attacking signals, the slew rate of the signals, the distance of the traces, and the travel direction of the signals.

To minimize crosstalk in high-speed signaling systems, signal traces are often routed closely with a dedicated current return path and are then routed with the dedicated current return path shielding active signal traces from each other. In this manner, the minimization of the current loop reduces EMI, and the spacing of active traces and the respective

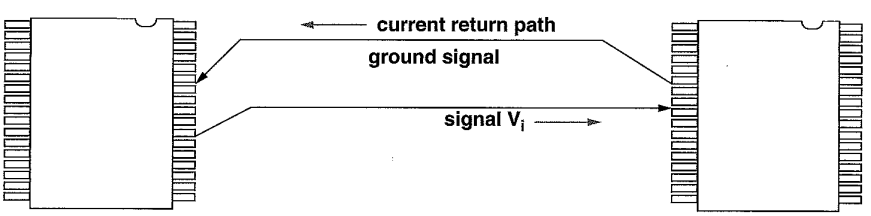


FIGURE 9.11: Voltage signal delivery and the return current, forming a current loop.

current return paths also reduces capacitive coupling. Unfortunately, the use of dedicated current return paths increases the number of traces required in the signaling system. As a result, dedicated shielding traces are typically found only in high-speed digital systems. For example, high-speed memory systems such as Rambus Corp's XDR memory system rely on differential traces over closely routed signaling pairs to ensure high signaling quality in the system, while lower cost, commodity-market-focused memory systems such as DDR2 SDRAM reserve differential signaling to clock signal and high-speed data strobe signals.

9.2.8 Near-End and Far-End Crosstalk

In general, two types of crosstalk effects exist in signal systems: near-end crosstalk and far-end crosstalk. These two types of crosstalk effects are, respectively, illustrated in Figures 9.12 and 9.13. Although Figures 9.12 and 9.13 illustrate crosstalk for capacitively coupled signal traces, near-end and far-end crosstalk effects are similar for inductively coupled signal traces. Figures 9.12 and 9.13 show that as a voltage signal travels from the source to the destination, it generates eletronic noises in an adjacent trace. At each point on the transmission line, the attacking signal generates a victim signal on the victim trace. The victim signal will then travel in two directions: in the same direction as the attacker signal and in the opposite direction of the attacker signal. Figure 9.12 shows that the victim signal traveling in the opposite direction of the attacker signal will result in a relatively long duration, low-amplitude noise at the near (source) end of the victim trace.

In contrast, Figure 9.13 shows that the electronic noise traveling in the same direction as the attacker signal will result in a victim signal that is relatively short in duration, but high in amplitude at the far (destination) end of the victim signal trace. In essence, the near-end and far-end crosstalk effects can be analogized to the Doppler effect. That is, as the attacker signal wave front moves from the source to the destination on a given transmission line, it creates sympathetic signals in closely coupled victim traces that are routed in close proximity. The sympathetic signals that travel backward toward the source of the attacking signal will appear as longer wavelengths and lower amplitude noise, and sympathetic signals that travel in the same direction as the attacking signal will add up to appear as a short wavelength, high-amplitude noise.

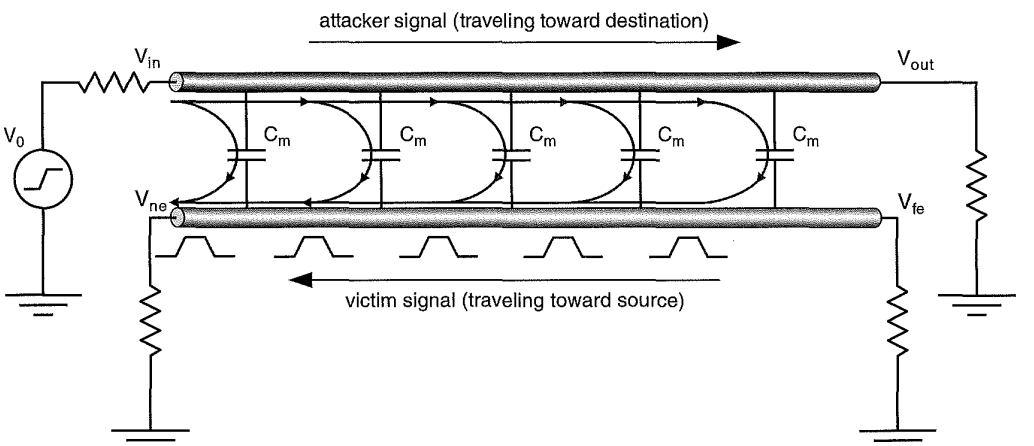


FIGURE 9.12: Near-end crosstalk.

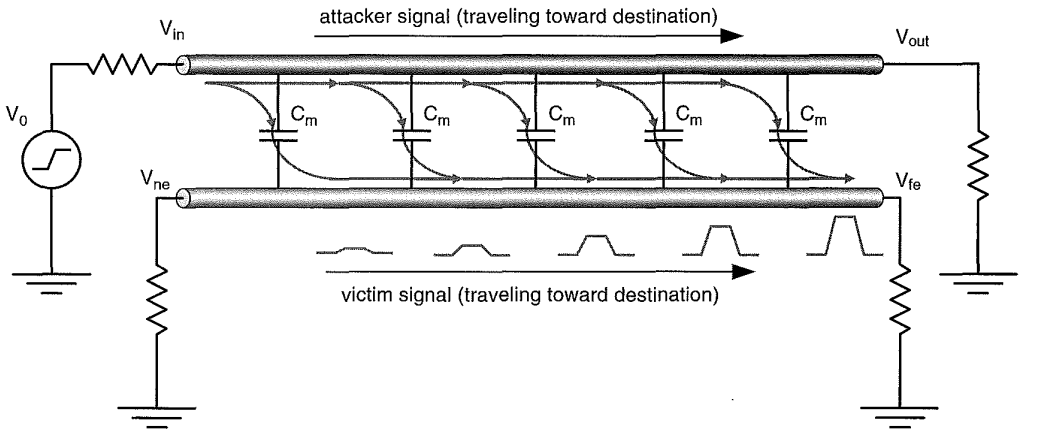


FIGURE 9.13: Far-end crosstalk.

Finally, an interesting general observation is that since the magnitude of the crosstalk depends on the magnitude of the attacking signals, an increase in the signal strength of one signal in the system would, in turn, increase the magnitude of the crosstalk experienced by other traces in the system, and the increase in signal strength of all signals in the system, in turn, exacerbates the crosstalk problem on the system level. Consequently, the issue of crosstalk must be solved by careful system design rather than a simple increase in the strength of signals in the system.

9.2.9 Transmission Line Discontinuities

In an ideal LC transmission line with uniform characteristic impedance throughout the length of the transmission line, a signal can, in theory, propagate down the transmission line without reflection or attenuation at any point in the transmission line. However, in the case where the transmission line consists of multiple segments with different characteristic impedances for each segment, signals propagating on the transmission line will be altered at the interface of each discontinuous segment.

Figure 9.14 illustrates that at the interface of any two mismatched transmission line segments, part of the incident signal will be transmitted and part of the incident signal will be reflected toward the source. Figure 9.14 also shows that the characteristics of the mismatched interface can be described in terms of the reflection coefficient ρ, and ρ can be computed from the formula $\rho = (Z_L - Z_S)/(Z_L + Z_S)$. With the formula for the reflection coefficient, the reflected signal at the interface of two transmission line segments can be computed by multiplying the voltage of the incident signal and the reflection coefficient. The voltage of the transmitted signal can be computed in a similar fashion since the sum of the voltage of the transmitted signal and the voltage of the reflected signal must equal the voltage of the incident signal.

In any classroom discussion about the reflection coefficient of transmission line discontinuities, there are three special cases that are typically examined in detail: the well-matched transmission line segments, the open-circuit transmission line, and the short-circuit transmission line. In the case where the characteristic impedances of two transmission line segments are matched, the reflection coefficient of that interface ρ is 0 and all of the signals are

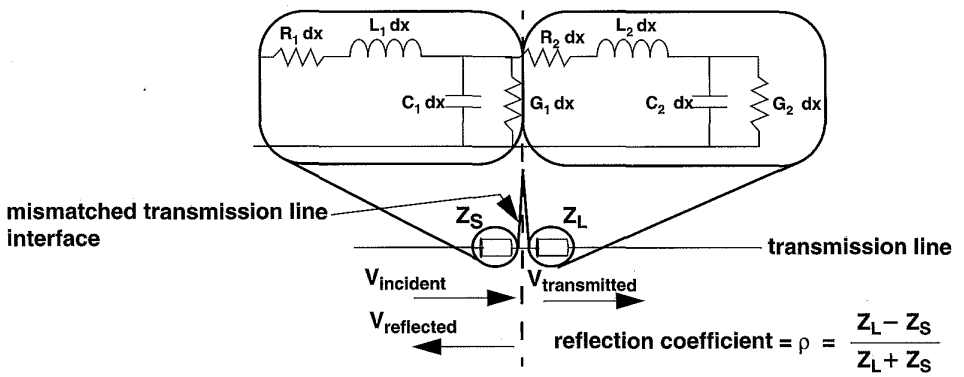


FIGURE 9.14: Signal reflection at an unmatched transmission line interface.

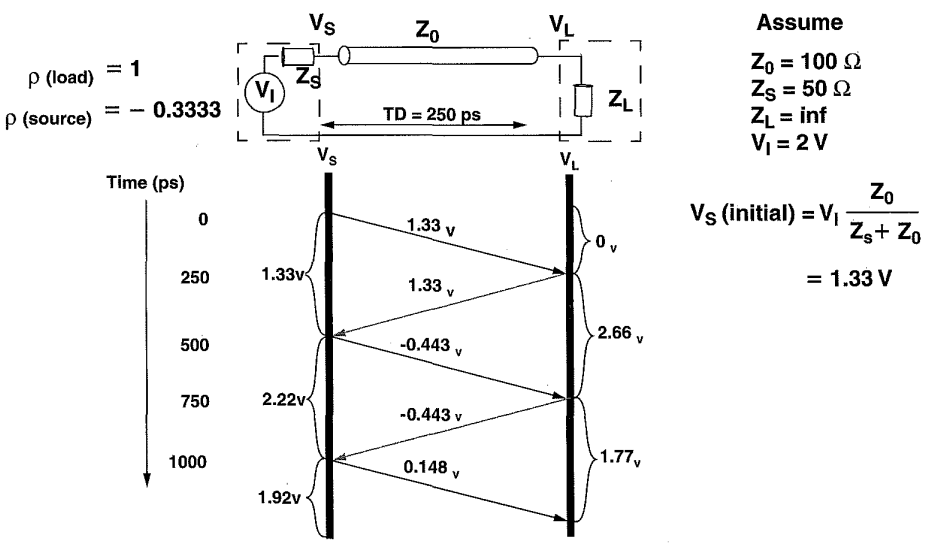


FIGURE 9.15: Illustration of signal reflection on a poorly matched transmission line.

transmitted from one segment to another segment. In the case that the load segment is an open circuit, the reflection coefficient is 1, the incident signal will be entirely reflected toward the source, and no part of the incident signal will be transmitted across the open circuit. Finally, in the case where the load segment is a short circuit, the reflection coefficient is −1, and the incident signal will be reflected toward the source with equal magnitude but opposite sign.

Figure 9.15 illustrates a circuit where the output impedance of the voltage source is different from the impedance of the transmission line. The transmission line also drives a load whose impedance is comparable to that of an open circuit. In the circuit illustrated to the

figure, there are three different segments, each with a different characteristic impedance. In this transmission line, there are two different impedance discontinuities. Figure 9.15 shows that the reflection coefficients of the two different interfaces are represented by ρ(source) and ρ(load), respectively. Finally, the figure also shows that the transmission line segment that connects the source to the load has finite length, and the signal flight time is 250 ps on the transmission line between the mismatched interfaces.

Figure 9.15 shows the voltage ladder diagram where the signal transmission begins with the voltage source driving a 0-V signal prior to time zero and switching instantaneously to 2 V at time zero. The figure also shows that the initial voltage V_S can be computed from the basic voltage divider formula, and the initial voltage is computed and illustrated as 1.33 V. The ladder diagram further shows that due to the signal flight time, the voltage at the interface of the load, V_L, remains at 0 V until 250 ps after the incident signal appears at V_S. The 1.33-V signal is then reflected with full magnitude by the load with the reflection coefficient of 1 back toward the voltage source. Then, after another 250 ps of signal flight time, the reflected signal reaches the interface between the transmission line and the voltage source. The reflected signal with the magnitude of 1.33 V is then itself reflected by the transmission line discontinuity at the voltage source, and the re-reflected signal of −0.443 V once again propagates toward the load.

Figure 9.16 illustrates that the instantaneous voltage on a given point of the transmission line can be computed by the sum of all of the incident and reflected signals. The figure also illustrates that the superposition of the incident and reflected signals shows that the output signal at V_L appears as a severe ringing problem that eventually converges around the value driven by the voltage source, 2 V. However, as the example in Figure 9.16 illustrates, the convergence only occurs after several round-trip signal flight times on the transmission line.

9.2.10 Multi-Drop Bus

In commodity DRAM memory systems such as SDRAM, DDR SDRAM, and similar DDRx SDRAM, multi-drop busses are used to carry command, address, and data signals from the memory controller to multiple DRAM devices. Figure 9.17 shows that from the perspective of the PCB traces that carry

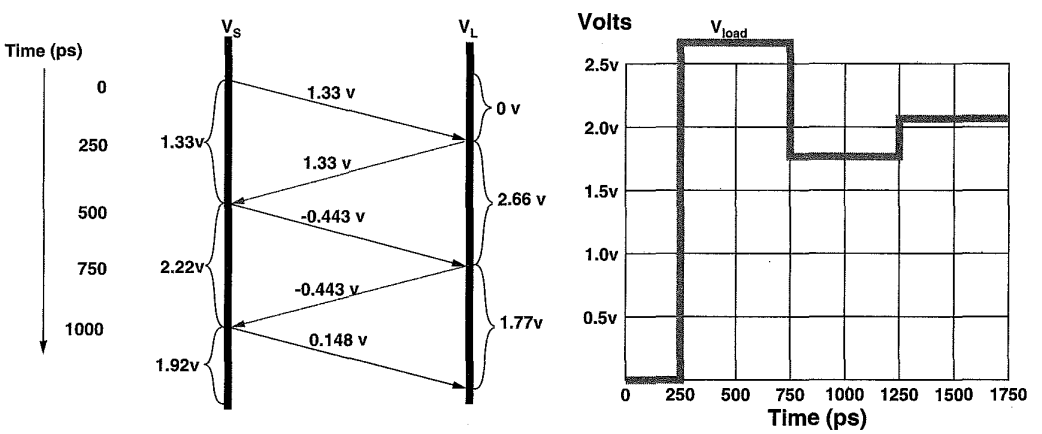


FIGURE 9.16: Signal waveform construction from multiple reflections.

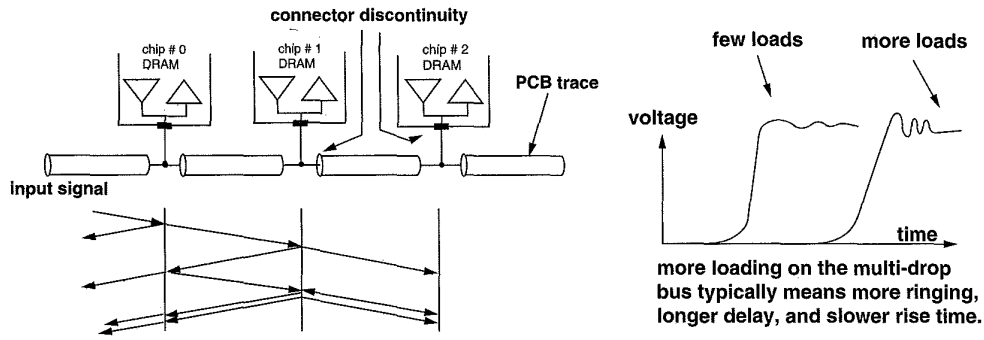


FIGURE 9.17: Multi-drop bus in a commodity DRAM memory system.

signals on the system board, each DRAM device on the multi-drop bus appears as an impedance discontinuity. The figure shows an abstract ladder diagram where a signal propagating on the PCB trace will be partly reflected and transmitted across each impedance discontinuity. Typical effects of signal propagation across multiple impedance discontinuities are more ringing, a longer delay, and a slower rise time.

The loading characteristics of a multi-drop bus, as illustrated in Figure 9.17, means that the effects of impedance discontinuities must be carefully controlled to enable a signaling system to operate at high data rates. As part of the effort to enable higher data rates in successive generations of DRAM memory systems, specifications that define tolerances of signal trace lengths, impedance characteristics, and the number of loads on the multi-drop bus have become ever more stringent. For example, in SDRAM memory systems that operate with the data rate of 100 Mbps, as many as eight SDRAM devices can be connected to the data bus, but in DDR3 SDRAM memory systems that operate with the data rate of 800 Mbps and above, the initial specification requires that no more than two devices can be connected to the same data bus in a connection scheme referred to as point-to-two-point (P22P), where the controller is specified to be limited to the connection of two DRAM devices located adjacent to each other.

9.2.11 Socket Interfaces

One feature that is demanded by end-users in commodity DRAM memory systems is the feature that allows the end-user to configure the capacity of the DRAM memory system by adding or removing memory modules as needed. However, the use of memory modules means that socket interfaces are needed to connect PCB traces on the system board to PCB traces on the memory modules that then connect to the DRAM devices. Socket interfaces are highly problematic for a transmission line in the sense that a socket interface represents a capacitive discontinuity for the transmission line, even in the case where the system only has a single memory module and the characteristic impedances of the traces on the system board and the memory module are well matched to each other.

To ensure that DRAM memory systems can operate at high data rates, memory system design engineers must carefully model each component of the memory system as well as the overall behavior of the system in response to a signal injected into the system. Figure 9.18 illustrates an abstract model of the data bus of a DDR SDRAM memory system, and it shows that a signal transmitted by the controller will first propagate on PCB traces in the system board. As the propagated signal encounters a socket interface, part of the signal will be reflected back toward the controller and part of the signal will continue

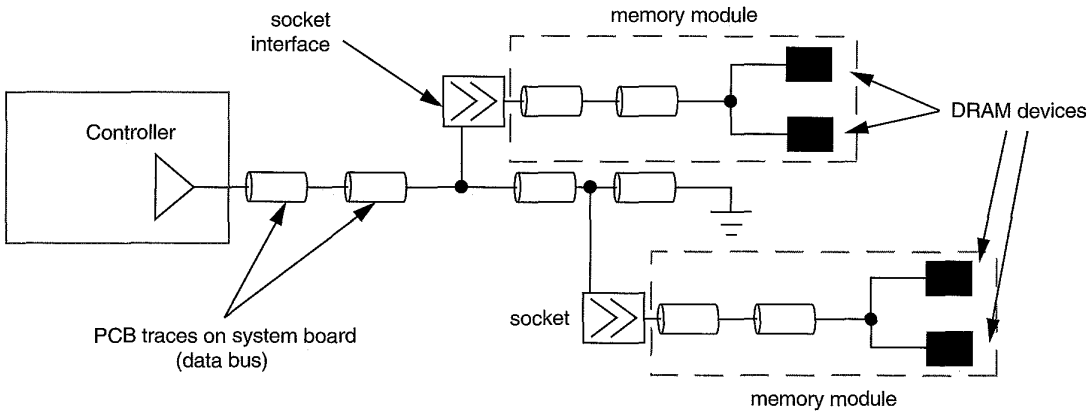


FIGURE 9.18: Transmission line representation of data bus in a DDR SDRAM memory system.

to propagate down the PCB trace, while most of the signal will be transmitted through the socket interface onto the memory module where the signal continues propagation toward the DRAM devices.

9.2.12 Skew

Figure 9.8 shows that wave velocity on a transmission line can be computed with the formula 1/SQRT(LC). For the transmission line with the specific impedance characteristics given in Figure 9.8, a signal would travel through the distance of 20 cm/ns. As a result, any signal path lengths that are not well matched to each other in terms of distances would introduce some amounts of timing skew between signals that travel on those paths.

Figure 9.19 illustrates the concept of signal skew in a poorly designed signaling system. In Figure 9.19, two signal traces carry signals on a parallel bus from the controller to modules labelled as module *A* and module *B*. Figure 9.19 shows that due to the poor design, path 1 which carries bus signal #1 is shorter than path 2 which carries bus signal #2. In this system, the different path lengths introduce static skew between bus signal #1 and bus signal #2.

In this chapter, *skew* is defined as the timing differential between two signals in a system. Skew can exist between data signals of a wide parallel data bus or between data signals and the clock signal. Moreover, signal skew can be introduced into a signaling system by differences in path lengths or the electrical loading characteristics of the respective signal paths. As a result, skew minimization is an absolute requirement in the implementation of high-speed, wide, and parallel data busses. Fortunately, the skew component of the timing budget is typically static, and with careful design, the impact of the data-to-data and data-to-clock skew can be minimized. For example, the PCB image in Figure 9.19 shows the trace routing on the system board of a commodity personal computer, and it illustrates that system design engineers often purposefully add extra twists and turns to signal paths to minimize signal skew between signal traces of a parallel bus.

9.2.13 Jitter

In the broad context of analog signaling, jitter can be defined as unwanted variations in amplitude or timing between successive pulses on a given signal line. In this chapter, the discussion on jitter is limited to the context of DRAM memory systems with digital voltage levels, and only the short-term phase variations that exist between successive cycles of a given signal are considered. For example, electronic

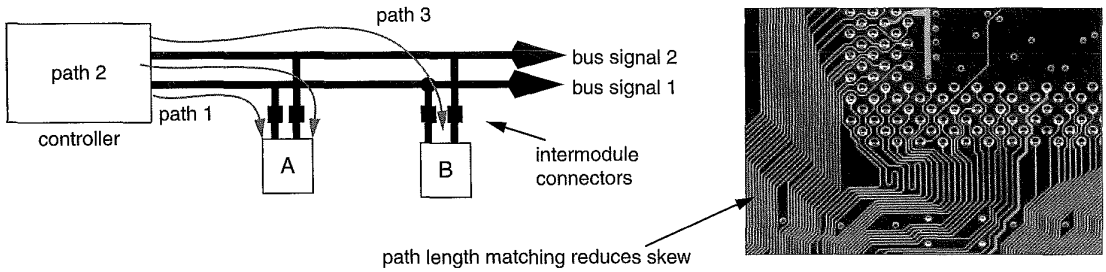


FIGURE 9.19: Mismatched path lengths introduces signal-to-signal skew, and PCB with path-length-matched signal traces.

components are sensitive to short-term variations in supply voltage and temperature, and effects such as crosstalk depend on transitional states of adjacent signals. Collectively, the impact of short-term variations in supply voltage, temperature, and crosstalk can vary dramatically between successive cycles of a given signal. As a result, the propagation time of a signal on a given signal path can exhibit subtle fluctuations on a cycle-by-cycle basis. These subtle variations in timing are defined as *jitter* on a given signal line.

To ensure correctness of operation, signaling systems account for timing variations introduced into the system by both skew and jitter. However, due to the unpredictable nature of the effects that cause jitter, it is often difficult to fully characterize jitter on a given signal line. As a result, jitter is also often more difficult than skew to deal with in terms of timing margins that must be devoted to account for the timing uncertainty.

9.2.14 Inter-Symbol Interference (ISI)

In this chapter, crosstalk, signal reflections, and other effects that impact signal integrity and timing are examined separately. However, the result of these effects can be summarized in the sense that they all degrade the performance of the signaling system. Additionally, multiple, consecutive signals on the same transmission line can have collective, residual effects that can interfere with the transmission of subsequent signals on the same transmission line. The intra-trace inference is commonly referred to as *inter-symbol interference* (ISI).

Inter-symbol interference is intrinsically a band-pass filter issue. The interconnect is a non-linear low-pass filter. The energy of the driver signal resides mostly within the third harmonic frequency. But the interconnect low-pass filter is non-linear, which causes the dispersion of the signal. In other words, a given signal that is not promptly removed from the transmission medium by the signal termination mechanism may disperse slowly and affect later signals that make use of the same transmission medium. For example, a single pulse has a long tail beyond its "ideal" pulse range. If there are consecutive "1"s, the accumulated tails may add up to overwrite the following "0." The net effect of ISI is that the interference degrades performance and limits the signaling rate of the system.

9.3 Termination

Previous sections illustrate that signal propagation on a transmission line with points of impedance discontinuity will result in multiple signal reflections, with one set of reflections at each point of impedance discontinuity. Specifically, the example in Figure 9.15 shows that in a system where the input impedance of

the load[4] differs significantly from the characteristic impedance of the transmission line, the impedance discontinuity at the interface between the transmission line and the load results in multiple, significant signal reflections that delay the settle time of the transmitted signal. To limit the impact of signal reflection at the end of a transmission line, high-speed system design engineers typically place termination elements whose resistive value matches the characteristic impedance of the transmission line. The function of the termination element is to remove the signal from the transmission line and eliminate the signal reflection caused by the impedance discontinuity at the load interface. Figure 9.20 shows the placement of the termination element Z_T at the end of the transmission line. Ideally, in a well-designed system, the resistive value of the termination element, Z_T, matches exactly the characteristic impedance of the transmission line, Z_0. The signal is then removed from the transmission line by the termination element, and no signal is reflected toward the source of the signal.

9.3.1 Series Stub (Serial) Termination

The overriding consideration in the design and standardization process of modern DRAM memory systems designed for the commodity market is the cost of the DRAM devices. To minimize the manufacturing cost of the DRAM devices, relatively low-cost packaging types such as SOJ and TSOP are used in SDRAM memory systems, and the TSOP is used in DDR SDRAM memory systems.[5]

Unfortunately, the input pin of the low-cost SOJ and TSOP packages contains relatively large inductance and capacitance characteristics and typically represents a poorly matched load for any transmission line. The poorly matched impedance between the system board trace and device packages is not a problem for DRAM memory systems with relatively low operating frequencies (below 200 MHz). However, the mismatched impedance issue gains more urgency with each attempt to increase the data rate of the DRAM memory system.

Figure 9.21 shows the series stub termination scheme used in DDR SDRAM memory systems. Unlike the ideal termination element, the series stub terminator is not designed to remove the signal from the transmission line once the signal has been delivered to the receiver. Rather, the series resistor is designed to increase the damping ratio and to provide an artificial impedance discontinuity that isolates the complex impedances within the DRAM package, resulting in the reduction of signal reflections back onto the PCB trace from within the DRAM device package.

9.3.2 On-Die (Parallel) Termination

The use of series termination resistors in SDRAM and DDR SDRAM memory systems to isolate the complex impedances within the DRAM

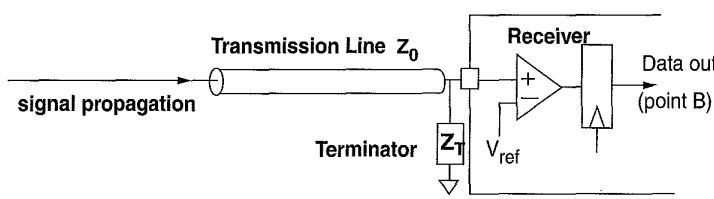


FIGURE 9.20: A well-matched termination element removes the signal at the end of the transmission line.

[4]In this case, the load is the receiver of the signal.

[5]SOJ stands for Small Outline J-lead, and TSOP stands for Thin Small Outline Package. Due to their proliferation, these packages currently enjoy cost advantages when compared to Ball Grid Array (BGA) type packages.

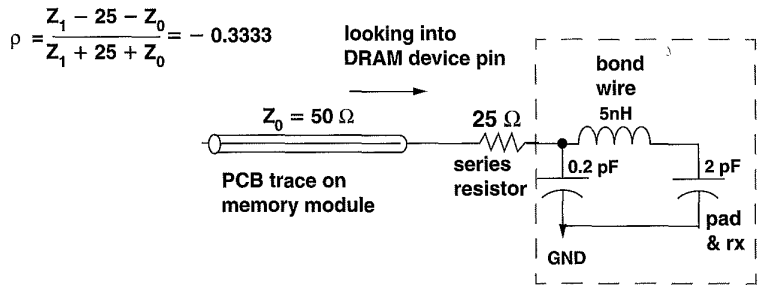


FIGURE 9.21: Series stub termination in DDRx SDRAM devices.

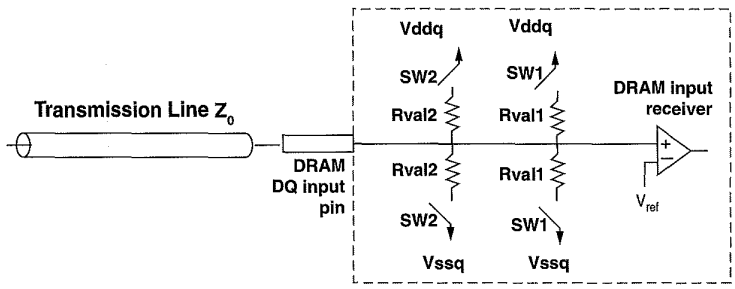


FIGURE 9.22: On-die termination scheme of a DDR2 SDRAM device.

device package means that the burden is placed on system design engineers to add termination resistors to the DRAM memory system. In DDR2 and DDR3 SDRAM devices, the use of the higher cost Fine Ball Grid Array (FBGA) package enables DRAM device manufacturers to remove part of the inductance that exists in the input pins of SOJ and TSOP packages. As a result, DDR2 and DDR3 SDRAM devices could then adopt an on-die termination scheme that more closely represents the ideal termination scheme illustrated in Figure 9.20.

Figure 9.22 shows that in DDR2 devices, depending on the programmed state of the control register and the value of the on-die termination (ODT) signal line, switches SW1 and SW2 can be controlled independently to provide different termination values as needed. The programmability of the on-die termination of DDR2 devices, in turn, enables the respective DRAM devices to adjust to different system configurations without having to assume worst-case system configurations.

9.4 Signaling

In DRAM memory systems, the signaling protocol defines the electrical voltage or current levels and the timing specifications used to transmit and receive commands and data in a given system. Figure 9.23 shows the eye diagram for a basic binary signaling system commonly used in DRAM memory systems, where the voltage values V_0 and V_1 represent the two states in the binary signaling system. In the figure, t_{cycle} represents the cycle time of one signal transfer in the signaling system. The cycle time can be broken down into different components: the signal transition time t_{tran}, the skew and jitter timing budget t_{skew}, and the valid bit time t_{eye}.

To achieve high operating data rates, the cycle time, t_{cycle}, must be as short as possible. The goal in the design and implementation of a high-speed signaling system is then to minimize the time spent by signals on state transition, account for possible skew and jitter, and respect signal setup and hold time requirements. The voltage and timing requirements of the signaling protocol must be respected in all cases regardless of the existence of transient voltage noises such as those caused by crosstalk or transient timing noises such as those caused by temperature-dependent signal jitter. In the design and verification process of high-speed signaling systems, eye diagrams such as the one illustrated in Figure 9.23 are often used to describe the signal quality of a signaling system. A high-quality signaling system with properly matched and terminated transmission lines will minimize skew and jitter, resulting in eye diagrams with well-defined eye openings and minimum timing and voltage uncertainties.

9.4.1 Eye Diagrams

Figure 9.24 is an example that shows the practical use of eye diagrams. The eye diagram of a signal in a system designed without termination is shown on the left, and the eye diagram of a signal in a system designed with termination is shown on the right. Figure 9.24 illustrates that in a high-quality signaling system, the eye openings are large, with clearly defined voltage and timing margins. As long as the eye opening of the signal remains intact, buffers can effectively eliminate voltage noises and boost binary voltage levels to their respective maximum and minimum values. Unfortunately, timing noises cannot be recovered by a simple buffer, and once jitter is introduced into the signaling system, the timing uncertainty will require a larger timing budget to account for the jitter. Consequently, a longer cycle time and lower data rate may be needed to ensure the correctness of signal transmission in the system—for a poorly designed signaling system.

9.4.2 Low-Voltage TTL (Transistor-Transistor Logic)

Figure 9.25 illustrates the input and output voltage response for a low-voltage TTL (LVTTL) device. The LVTTL signaling protocol is used in SDRAM memory systems and other DRAM memory of its generation, such as Extended Data-Out DRAM (EDO DRAM), Virtual Channel DRAM (VCDRAM), and Enhanced SDRAM (ESDRAM) memory systems. The LVTTL signaling specification is simply a reduced voltage specification of the venerable TTL signaling specification that operates with a 3.3-V voltage supply rather than the standard 5-V voltage supply.

Similar to the TTL devices, LVTTL devices do not supply voltage references to the receivers of the signals. Rather, the receivers are expected to provide internal references so that input voltages lower than 0.8 V are resolved as one state of the binary signal and input voltages higher than 2.0 V are resolved as the alternate state of the binary signal. LVTTL devices are expected

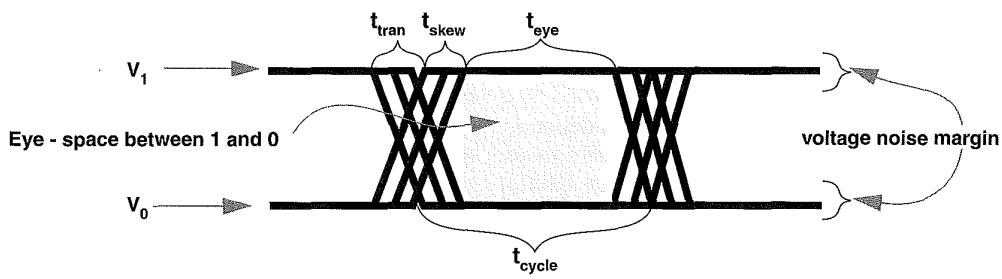


FIGURE 9.23: Eye diagram for binary signaling.

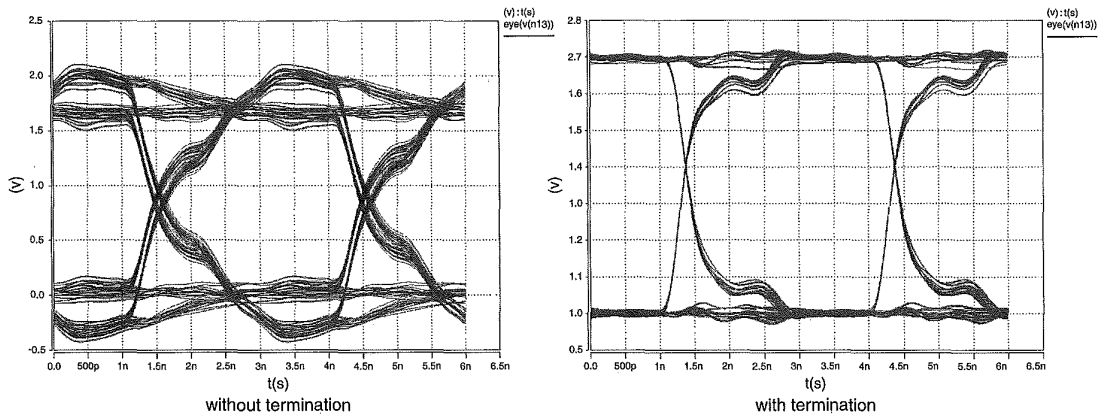


FIGURE 9.24: Eye diagrams of a signaling system with and without termination.

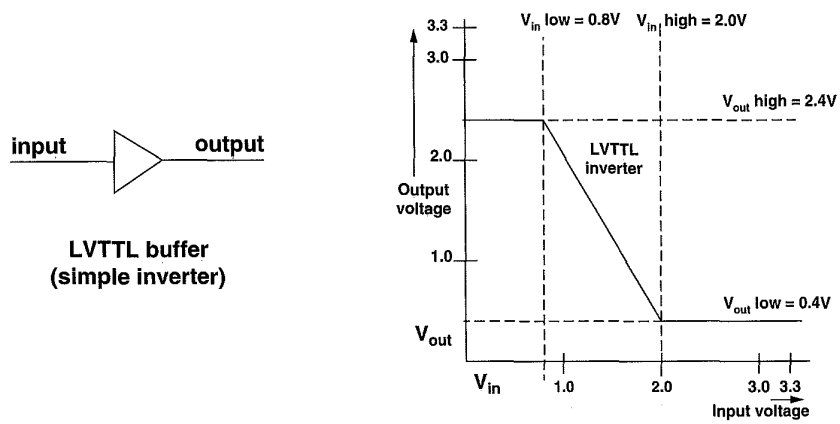


FIGURE 9.25: Inverter buffer and comparator input.

to drive voltage-high output signals above 2.4 V and low output signals below 0.4 V. For a LVTTL signal to switch state, the signal must traverse a large voltage swing. The large voltage swing and the large voltage range of undefined state between 0.8 and 2.0 V effectively limits the use of TTL and LVTTL signaling protocols to relatively low-frequency systems. For example, SDRAM memory systems are typically limited to operating frequencies below 167 MHz. Subsequent generations of DRAM memory systems have since migrated to more advanced signaling protocols such as Series Stub Terminated Logic (SSTL).

9.4.3 Voltage References

In modern DRAM memory systems, voltage-level-based signaling mechanisms are used to transport command, control, address, and data from the controller to and from the DRAM devices. However, for the delivered signals to be properly resolved by the receiver, the voltage level of the delivered signals must be compared to a reference voltage. While the use of implicit voltage references in TTL and LVTTL devices is sufficient for the relatively low operating frequency ranges and generous voltage range of the TTL and LVTTL signaling protocols, implicit voltage references are inadequate in modern DRAM memory systems that operate at dramatically higher data rates and with far smaller voltage ranges. As a result, voltage references are used in modern DRAM memory systems just as they are typically used in high-speed ASIC signaling systems.

Figure 9.26 illustrates two common schemes for reference voltage comparison. The diagram on the left illustrates the scheme where multiple input signal pins share a common voltage reference that is delivered by the transmitter along with the data signals. The diagram on the right shows differential signaling where each signal is delivered with its complement. The common voltage scheme is used in DRAM memory systems where low cost predominates over the requirement of high data rate, and differential signaling is used to achieve a higher data rate at the cost of higher pin count in high-speed DRAM memory systems. Aside from the advantage of being able to make use of the full voltage swing to resolve signals to one state or another, complementary pinpairs are always routed closely together, and the minimal distance allows differential pin pairs to exhibit superior common-mode noise rejection characteristics.

9.4.4 Series Stub Termination Logic

Figure 9.27 illustrates the voltage levels used in 2.5-V Series Stub Termination Logic signaling (SSTL_2). SSTL_2 is used in DDR SDRAM memory systems, and DDR2 SDRAM memory systems use SSTL_18 as the signaling protocol. SSTL_18 is simply a reduced voltage version of SSTL_2, and SSTL_18 is specified to operate with a supply voltage of 1.8 V.

Figure 9.27 shows that, unlike LVTTL, SSTL_2 signaling makes use of a common voltage reference to accurately resolve the value of the input signal. The figure further shows that where V_{ref} is set to 1.25 V, signals in the range of 0 to 1.1 V at the input of the SSTL_2 buffer are resolved as voltage low and signals in the range of 1.4 to 2.5 V are resolved as voltage high. In this manner, SSTL_2 exhibits better noise margins for the low-voltage range and nearly comparable noise margins for the high-voltage range when

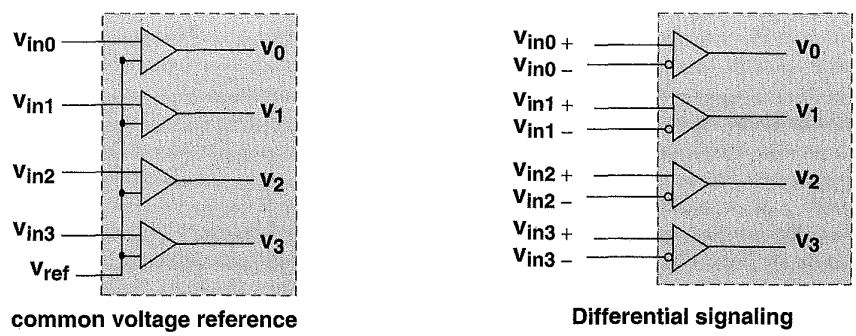


FIGURE 9.26: Local and remote voltage references.

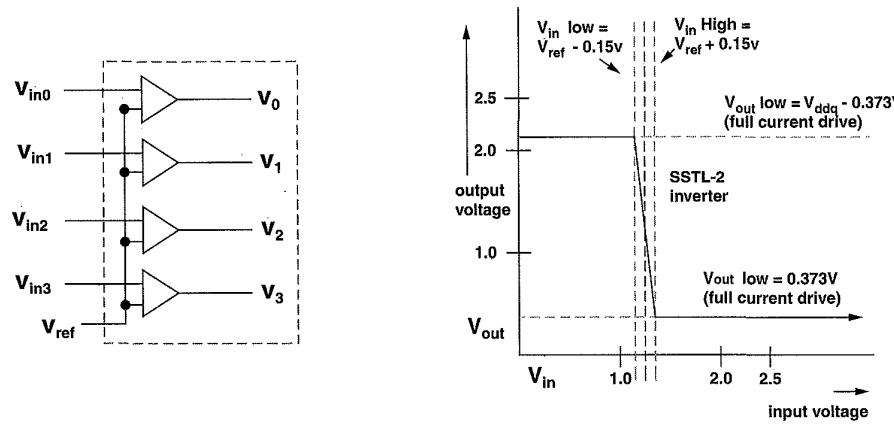


FIGURE 9.27: SSTL_2 voltage level illustration.

compared to LVTTL despite the fact that the supply voltage used in SSTL_2 is far lower than the supply voltage used in LVTTL.[6]

9.4.5 RSL and DRSL

In recent years, Rambus Corp. introduced two different signaling systems that are used in two different high-speed DRAM memory systems. The Rambus signaling level (RSL) signaling protocol was used in the Direct RDRAM memory system, and the differential Rambus signaling level (DRSL) signaling protocol was used in the XDR memory system. RSL and DRSL are interesting due to the fact that they are different from LVTTL and SSTL-x signaling protocols.

Figure 9.28 illustrates that both RSL and DRSL utilize low-voltage swings compared to SSTL and LVTTL. In SSTL and LVTTL signaling systems, signals swing the full voltage range from 0 V to V_{ddq}, enabling a common voltage supply level for the DRAM core as well as the signaling interface at the cost of longer signal switch times. In the RSL signaling system, signals swing from 1.0 to 1.8 V; in the DRSL signaling system, signals swing from 1.0 to 1.2 V. While the low voltage swing enables fast signal switch time, the different voltage levels mean that the signaling interface must be carefully isolated from the DRAM core, thus requiring more circuitry on the DRAM device.

Finally, Figure 9.28 shows that similar to SSTL, RSL uses a voltage reference to resolve signal states, but DRSL does not use shared voltage references. Rather, DRSL is a bidirectional point-to-point differential signaling protocol that uses current mirrors to deliver signals between the memory controller and XDR DRAM devices.

9.5 Timing Synchronization

Modern digital systems, in general, and modern DRAM memory systems, specifically, are often designed as synchronous state machines. However, the address, command, control, data, and clock

[6]DDR SDRAM was originally only specified with speed bins up to 166 MHz at a supply voltage level, V_{ddq}, of 2.5 V. However, market trends and industry pressure led to the addition of the 200-MHz (400-Mbps) speed bin in DDR SDRAM memory systems. The 200-MHz DDR SDRAM memory system is specified to operate with a supply voltage, V_{ddq}, of 2.6 V.

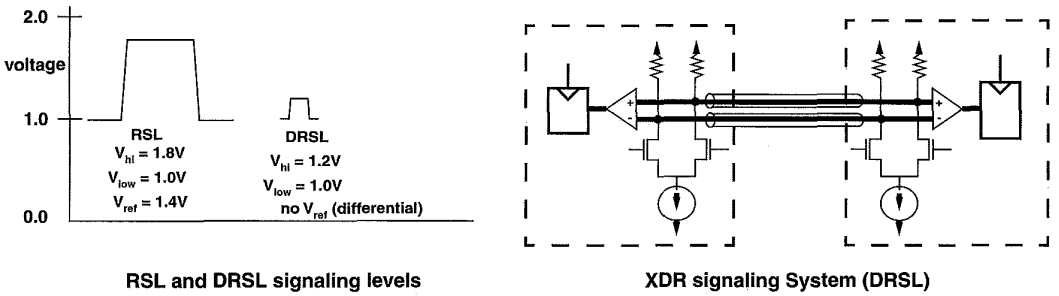


FIGURE 9.28: Rambus signaling levels (RSL and DRSL), and the DRSL signaling system.

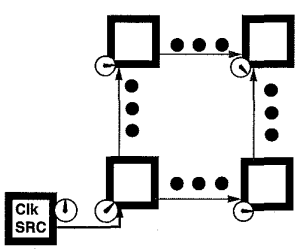


FIGURE 9.29: System-level synchronization with distributed clock signals.

signals in a modern high-speed DRAM memory system must all propagate on the same type of signal traces in PCBs. As a result, the clock signal is subject to the same issues of signal integrity and propagation delay that impact other signals in the system. Figure 9.29 abstractly illustrates the problem in high-speed digital systems, where the propagation delay of the clock signal may be greater than a non-negligible fraction of a clock period. In such a case, the notion of synchronization must be reexamined to account for the phase differences in the distribution of the clock signal.

In general, there are three types of clocking systems used in synchronous DRAM memory systems: a global clocking system, a source-synchronous clock forwarding system, and a phase or delay compensated clocking system. The global clocking system assumes that the propagation delay of the clock signal between different devices in the system is relatively minor compared to the length of the clock period. In such a case, the timing variance due to the propagation delay of the clock signal is simply factored into the timing margin of the signaling protocol. As a result, the use of global clocking systems is limited to relatively lower frequency DRAM memory systems such as SDRAM memory systems. In higher data rate DRAM memory systems such as Direct RDRAM and DDR SDRAM memory systems, the global clocking scheme is replaced with source-synchronous clock signals to ensure proper synchronization for data transport between the memory controller and the DRAM devices. Finally, as signaling data rates continue to climb, ever more sophisticated circuits are deployed to ensure proper synchronization for data transport between the memory controller and the DRAM devices. Specifically, *Phase-Locked Loop* (PLL) or *Delay-Locked Loop* (DLL) circuits are used in DRAM memory controllers to actively compensate for signal skew. In the following sections, the clocking schemes used in modern DRAM memory systems to enable system synchronization are examined.

9.5.1 Clock Forwarding

Figure 9.30 illustrates the trivial case where a reference clock signal is sent along with the data

signal from the transmitter to the receiver. The clock forwarding technique is used in DRAM memory systems such as DDR SDRAM and Direct DRAM, where subsets of signals that must be transported in parallel from the transmitter to the receiver are routed with clock or strobe signals that provide the synchronization reference. For example, the 16-bit-wide data bus in Direct RDRAM memory systems is divided into two sets of signals with separate reference clock signals for each set, and DDR SDRAM memory systems provide separate DQ strobe signals to ensure that each 8-bit subset of the wide data bus has an independent clocking reference.

The basic assumption here is that by routing the synchronization reference signal along with a small groups of signals, system design engineers can more easily minimize variances in the electrical and mechanical characteristics between a given signal and its synchronization reference signal, and the signal-to-clock skew is minimized by design.

Figure 9.30 also shows that the concept of clock forwarding can be combined with more advanced circuitry that further minimizes signal-to-clock skew.

9.5.2 Phase-Locked Loop (PLL)

Two types of circuits are used in modern digital systems to actively manage the distribution and synchronization of the clock signal: a PLL and a DLL. PLLs and DLLs are used in modern high-speed DRAM devices to adjust and compensate for the skew and buffering delays of the clock signal. Figure 9.31 illustrates a basic block diagram of a PLL that uses an input clock signal and a *voltage-controlled oscillator* (VCO) in a closed-loop configuration to generate an output clock signal. With the use of the VCO, a PLL is capable of adjusting the phase of the output clock signal relative to the input clock signal, and it is also capable of frequency multiplication. For example, in an XDR memory system where the data bus interface

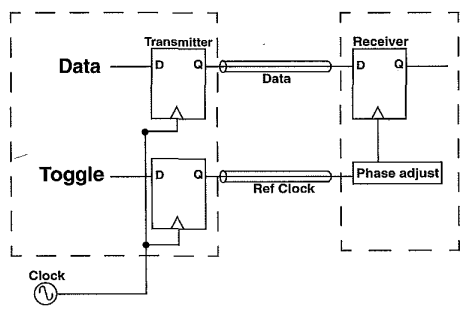


FIGURE 9.30: Clock forwarding.

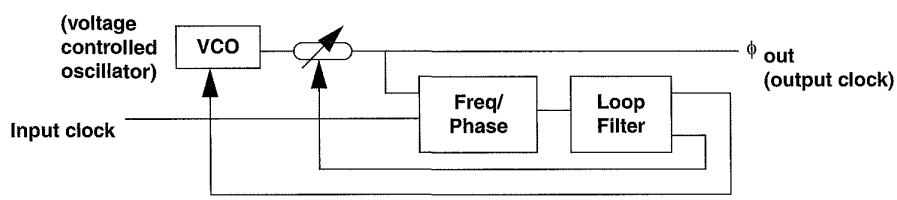


FIGURE 9.31: Basic PLL block diagram.

operates at a data rate of 3.2 Gbps, a relatively low-frequency clock signal that operates at 400 MHz is used to synchronize data transfer between the memory controller and the DRAM devices. In the XDR DRAM memory system, PLL circuits are used to multiply the 400-MHz clock frequency and generate a 1.6-GHz output clock signal from the slower input clock frequency. The devices in the XDR memory system then make use of the phase-compensated 1.6-GHz clock signal as the clock reference signal for data movement. In this manner, two silicon devices can operate synchronously at relatively high frequencies without the transport of a high-frequency clock signal on the system board.

PLLs can be implemented as a discrete semiconductor device, but they are typically integrated into modern high-speed digital semiconductor devices such as microprocessors and FPGAs. Depending on the circuit implementation, PLLs can be designed to lock in a range of input clock frequencies and provide the phase compensation and frequency multiplication needed in modern high-speed digital systems.

9.5.3 Delay-Locked Loop (DLL)

Figure 9.32 illustrates the basic block diagram of a DLL. The difference between the DLL and the PLL is that a DLL simply inserts a voltage-controlled phase delay between the input and output clock signals. In a PLL, a voltage-controlled oscillator synthesizes a new clock signal whose frequency may be a multiple of the frequency of the input clock signal, and the phase delay of the newly synthesized clock signal is also adjustable for active skew compensation. In this manner, while DLLs only delay the incoming clock signals, the PLLs actually synthesize a new clock signal as the output.

The lack of a VCO means that DLLs are simpler to implement and more immune to jitter induced by voltage supply or substrate noise. However, since DLLs merely add a controllable phase delay to the input clock signal and produce an output clock signal, jitter that is present in the input clock signal is passed directly to the output clock signal. In contrast, the jitter of the input clock signal can be better filtered out by a PLL.

The relatively lower cost of implementation compared to PLLs makes DLLs attractive for use in commodity DRAM devices designed to minimize manufacturing costs and in any application where clock synthesis is not required. Specifically, DLLs are found in DRAM devices such as DDRx SDRAM and GDDRx (Graphics DDRx) devices.[7]

9.6 Selected DRAM Signaling and Timing Issues

Signaling in DRAM memory systems is alike in many ways to signaling in logic-based[8] digital systems and different in other ways. In particular,

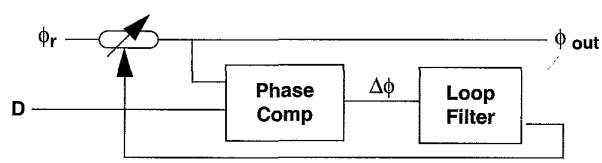


FIGURE 9.32: Basic DLL block diagram.

[7]DDRx denotes different generations of SDRAM devices such as DDR SDRAM, DDR2 SDRAM, and DDR3 SDRAM. Similarly, GDDRx denotes different generations of GDDR devices such as GDDR2 and GDDR3.

[8]Logic based, as opposed to memory based.

the challenges of designing high data rate signaling systems such as proper impedance matching, skew, jitter, and crosstalk minimization, as well as a system synchronization mechanism, are the same for all digital systems. However, commodity DRAM memory systems such as SDRAM, DDR SDRAM, and DDR2 SDRAM have some unique characteristics that are unlike high-speed signaling systems in non-DRAM-based digital systems. For example, in commodity DRAM memory systems, the burdens of timing control and synchronization are largely placed in the sophisticated memory controllers, and commodity DRAM devices are designed to be as simple and as inexpensive to manufacture as possible. The asymmetric master-slave relationship between the memory controller and the multiple DRAM devices also constrains system signaling and timing characteristics of commodity DRAM memory systems in ways not commonly seen in more symmetric logic-based digital systems.

Figure 9.18 illustrates the data bus structure of a commodity DRAM memory system. Figure 9.33 illustrates the command and address bus structure of the same DRAM memory system. Despite the superficial resemblance to the data bus structure illustrated in Figure 9.18, Figure 9.33 shows that in a commodity DRAM memory system, all of the DRAM devices within a DRAM memory system are typically connected to a given trace on the command and address bus. Figure 9.34 illustrates the topology of a typical commodity DRAM memory system where the address and command busses are connected to every device in the system.

The difference in the loading characteristics means that signals propagate far slower on the command and address bus than on the data bus in a typical commodity DRAM memory system. To alleviate the constraints placed on timing by the large number of loads on the command and address busses, numerous strategies have been deployed in modern DRAM memory systems. One strategy to alleviate timing and signaling constraints on

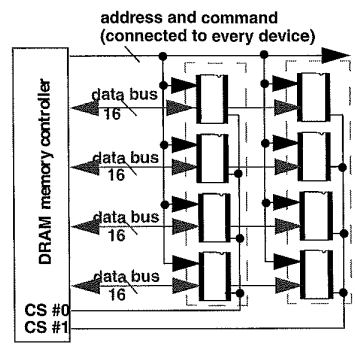


FIGURE 9.33: Typical topology of commodity DRAM memory systems such as DDRx SDRAM. Address and command busses are connected to every device.

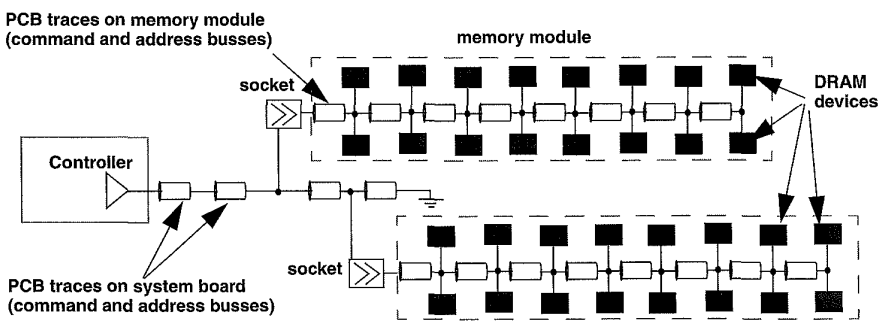


FIGURE 9.34: Transmission line model of DDR SDRAM memory system command and address busses.

the command and address buses is to make use of buffered or registered memory modules. For example, *registered dual in-line memory modules* (RDIMM) use separate buffer chips on the memory module to buffer command and address signals from the memory controller to the DRAM devices. Figure 9.35 illustrates that, from the perspective of the DRAM memory controller, the signal delivery path is limited to the system board through the socket interface and up to the input of the buffer chips on the memory modules. The buffer chips then drive the signal to the DRAM devices in a separate transmission line. The benefits of the buffer devices in the registered memory modules are shorter transmission lines, fewer electrical loads per transmission line, and fewer transmission line segments. The drawbacks of the buffer devices in the registered memory modules are higher cost associated with the buffer devices and the additional latency required for the address and command signal buffering and forwarding.

9.6.1 Data Read and Write Timing in DDRx SDRAM Memory Systems

Figure 9.33 illustrates that the command and address busses in a commodity DRAM memory system are typically more heavily loaded than the data bus, and timing margins are tighter on the command and address busses when compared to the data bus. Modern DDRx SDRAM memory systems capitalize on the lighter loading characteristics of the data bus by operating the data bus at twice the data rate as the command and address busses. The difference in operating the various busses at different data rates as well as the difference between the role of the DRAM memory controller and the commodity DRAM devices introduces several interesting aspects in the timing and synchronization of DDRx SDRAM devices. Figure 9.36 illustrates several interesting aspects of data read and write timing in commodity DDRx SDRAM devices: the use of a *Data Strobe Signal* (DQS) in DDRx SDRAM memory systems as the source-synchronous timing reference signal on the data bus, the

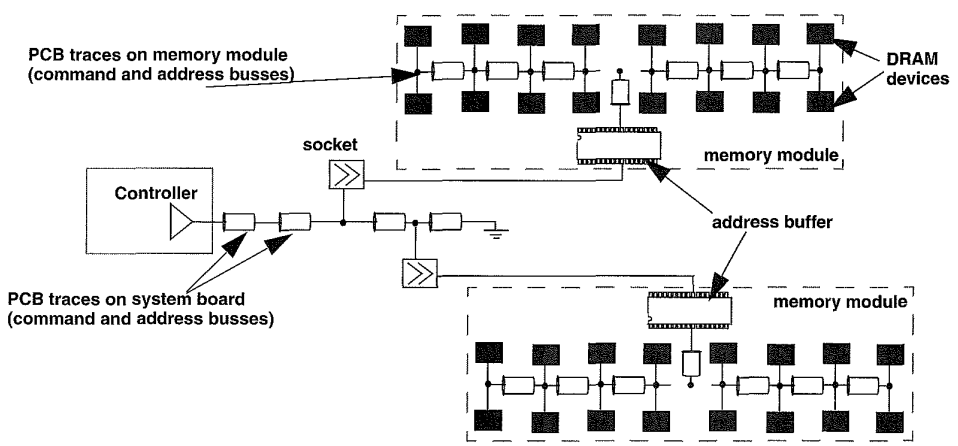


FIGURE 9.35: Register chips buffer command, control, and address signals in a Registered memory system.

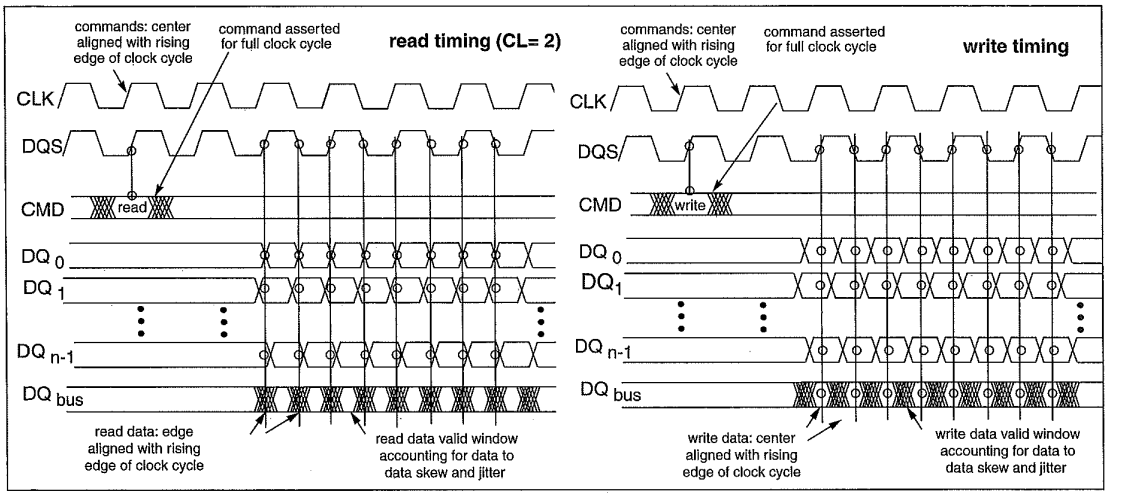


FIGURE 9.36: Column read, column write command and data timing in DDRx SDRAM memory systems.

transmission of symbols on half clock cycle boundaries on the data bus as opposed to full clock cycle boundaries on the command and address busses, and the difference in phase relationships between read and write data relative to the DQS signal.

First, Figure 9.36 illustrates that in DDRx SDRAM memory systems, address and command signals are asserted by the memory controller for a full clock cycle, while symbols on the data bus are only transmitted for half cycle durations. The doubling of the data rate on the data bus is an effective strategy that significantly increases bandwidth of the memory system from previous generation SDRAM memory systems without wholesale changes in the topology or structure of the DRAM memory system. However, the doubling of the data rate, in turn, significantly increases the burden on timing margins on the data bus. Figure 9.36 illustrates that valid data windows on the command, address, and data busses are constrained by the worst-case data skew and data jitter that exist in wide, parallel busses. Typically, in a DDRx SDRAM memory system with a 64-bit-wide data bus, the data bus is divided into smaller groupings of 8 signals per group, and a separate DQS signal is used to provide timing reference for each group of data bus signals. In this manner, the amount of timing budget allocated to account for skew and jitter is minimized on a group-by-group basis. Otherwise, much larger timing budgets would have to be allocated to account for skew and jitter across the entire width of the 64-bit-wide data bus. Figure 9.23 illustrates that where the timing budget devoted to t_{skew} increases, t_{cycle} must be increased proportionally, resulting in lower operation frequency.

Figure 9.36 also shows that read and write data are sent and received on different phases of the DQS signal in DDRx SDRAM memory systems. That is, DRAM devices return data bursts for read commands on each edge of the DQS signal. The DRAM memory controller then determines the centers of the valid data windows and latches in data at the appropriate time. In contrast, the DRAM memory controller delays the DQS signal by 90° relative to write data. The DRAM devices then latch in data relative to each edge of the DQS signal without shifting or delaying the data. One reason that the phase relationship

between the read data and the DQS signal differs from the phase relationship between write data and the DQS signal is that the difference in the phase relationships shifts the burden of timing synchronization from the DRAM devices and places it in the DRAM memory controller. That is, the commodity DDRx SDRAM devices are designed to be inexpensive to manufacture, and the DRAM controller is responsible for providing the correct phase relationship between write data and the DQS signal. Moreover, the DRAM memory controller must also adjust the phase relationship between read data and the DQS signal so that valid data can be latched in the center of the valid data window despite the fact that the DRAM devices place read data onto the data bus with the same phase relative to the DQS signal.

9.6.2 The Use of DLL in DDRx SDRAM Devices

In DDRx SDRAM memory systems, data symbols are, in theory, sent and received on the data bus relative to the timing of the DQS signal as illustrated in Figure 9.30, rather than a global clock signal used by the DRAM memory controller. Theoretically, the DQS signal, in its function as the source-synchronous clocking reference signal, operates independently from the global clock signal. However, where the DQS signal does operate independently from the global clock signal, the DRAM memory controller must either operate asynchronously or devote additional stages to buffer read data returning from the DRAM devices. A complete decoupling of the DQS signal from the global clock signal is thus undesirable. To mitigate the effect of having two separate and independent clocking systems, on-chip DLLs have been implemented in DDR SDRAM devices to synchronize the read data returned by the DRAM devices, the DQS signals, and the memory controller's global clock signal. The DLL circuit in a DDR DRAM thus ensures that data returned by DRAM devices in response to a read command is actually returned in synchronization with the memory controller's clock signal.

Figure 9.37 illustrates the use of an on-chip DLL in a DDR SDRAM device, and the effect that the DLL has upon the timing of the DQS and data signals. The upper diagram illustrates the operation of a DRAM device without the use of an on-chip DLL, and the lower diagram iillustrates the operation of a DRAM device with the use of an on-chip DLL. Where the DRAM device operates without an on-chip DLL, the DRAM device internally buffers the global clock signal and uses it to generate the source-synchronous DQS signal and return data to the DRAM memory controller relative to the timing of the buffered clock signal. However, the buffering of the clock signal introduces phase delay into the DQS signal relative to the global clock signal. The result of the phase delay is that data on the DQ signal lines is aligned closely with the DQS signal, but can be significantly out of phase alignment with respect to the global clock signal.

The lower diagram in Figure 9.37 illustrates that where the DRAM device operates with an on-chip DLL, the DRAM device also internally buffers the global clock signal, but the DLL then introduces more delay into the DQS signal. The net effect is to delay the output of the DRAM device by a full clock cycle so that the DQS signal along with the DQ data signals becomes effectively phase aligned to the global clock signal.

9.6.3 The Use of PLL in XDR DRAM Devices

In an effort devoted to the pursuit of high bandwidth in DRAM memory systems, the Rambus Corp. has designed several different clocking and timing synchronization schemes for its line of DRAM memory systems. One scheme utilized in the XDR memory system involves the use of PLLs in both the DRAM devices and the DRAM memory controller. Figure 9.38 abstractly illustrates an XDR DRAM device with two data pin pairs on the data bus along with a differential clock pin pair. The figure shows that the signals in the XDR DRAM memory system are transported via differential pin pairs. Figure 9.38 further illustrates that a relatively low system clock signal that operates at 400 MHz is used as a global clock reference between the XDR DRAM controller and the XDR DRAM device. The XDR DRAM device then makes use of a PLL to synchronize data latch operations at a higher data rate relative to the global clock signal. Figure 9.38 shows that the use of the PLL enables the XDR DRAM device to synchronize the transportation of eight data symbols per clock cycle. Moreover, the XDR DRAM controller contains a set of adjustable delay elements labelled as FlexPhase. The XDR DRAM

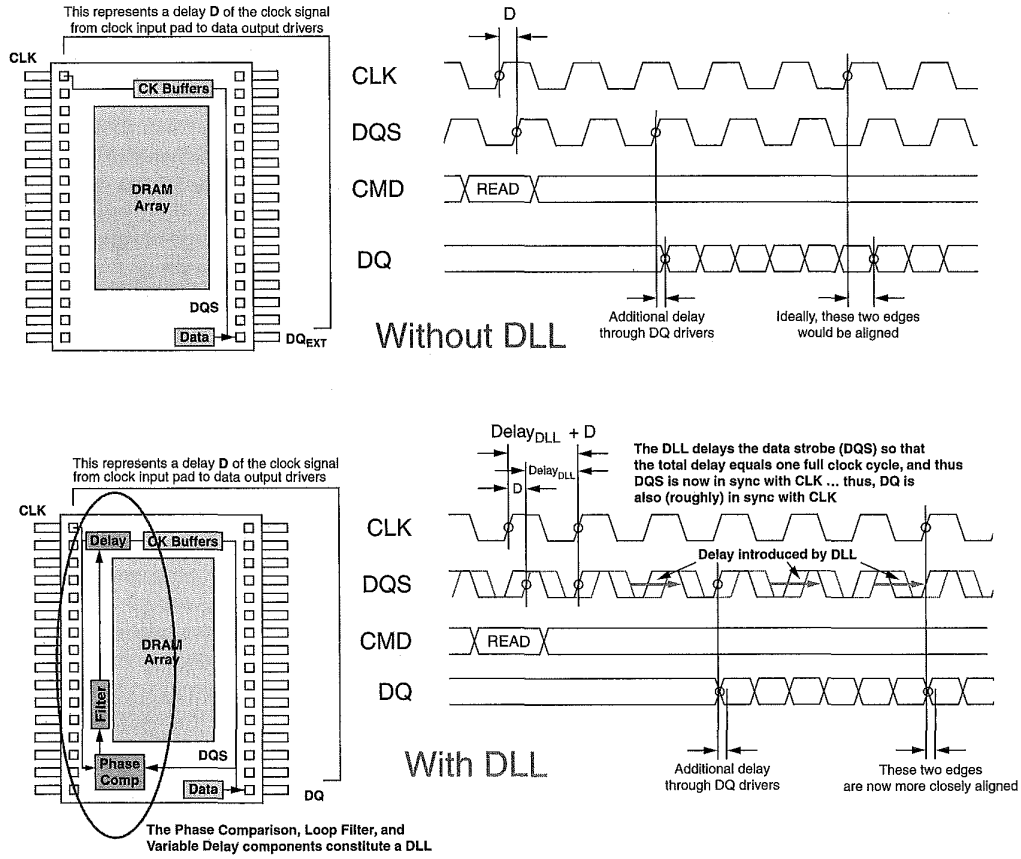


FIGURE 9.37: The use of DLL in DDRx SDRAM devices.

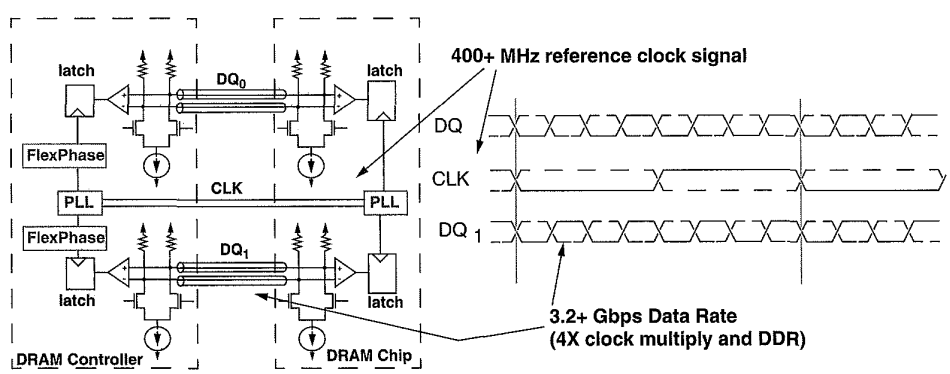


FIGURE 9.38: Use of PLL in XDR DRAM devices.

memory controller can set the adjustable delay in the FlexPhase circuitry to effectively remove timing uncertainties due to signal path length differentials. To some extent, the FlexPhase mechanism can also account for drift in skew characteristics as the system environment changes. The XDR memory controller can adjust the delay specified by the FlexPhase circuitry by occasionally suspending data movement operations and reinitializing the FlexPhase adjustable delay setting based on the transportation of predetermined test patterns.

9.7 Summary

This chapter covers the basic concepts of a signaling system that form the essential foundation for data transport between discrete devices. Specifically, the physical requirements of high data rate signaling in modern multi-device DRAM memory systems directly impact the design space of system topology and memory-access protocol. Essentially, this chapter illustrates that the task of transporting command, address, and data between the DRAM memory controller and DRAM devices becomes more difficult with each generation of DRAM memory systems that operate at higher data rates. Increasingly, sophisticated circuitry such as DLLs and PLLs is being deployed in DRAM devices, and the desire to continue to increase DRAM memory system bandwidth has led to restrictions to the topology and configuration of modern DRAM memory systems.

CHAPTER 10

DRAM Memory System Organization

Previous chapters examine the basic building blocks of DRAM devices and signaling issues that constrain the transmission and subsequent storage of data into the DRAM devices. In this chapter, basic terminologies and building blocks of DRAM memory systems are described. Using the building blocks described in the previous chapters, the text in this chapter examines the construction, organization, and operation of multiple DRAM devices in a larger memory system. This chapter covers the terminologies and topology, as well as the organization of various types of memory modules.

10.1 Conventional Memory System

The number of storage bits contained in a given DRAM device is constrained by the manufacturing process technology, the cell size, the array efficiency, and the effectiveness of the defect-cell remapping mechanism for yield enhancement. As the manufacturing process technology advances in line with Moore's Law, the number of storage bits contained in a given DRAM device doubles every few years. However, the unspoken corollary to Moore's Law states that software written by software companies in the Pacific Northwest and elsewhere will automatically expand to fill available memory in a given system. Consequently, the number of storage bits contained in a single DRAM device at any given instance in time has been and will continue to be inadequate to serve as the main memory for most computing platforms with the exception of specialty embedded systems.

In the past few decades, the growth rate of DRAM device storage capacity has roughly paralleled the growth rate of the size of memory systems for desktop computers, workstations, and servers. The parallel growth rates have dictated system designs in that multiple DRAM devices must be connected together to form memory systems in most computing platforms. In this chapter, the organization of different multi-chip DRAM memory systems and different interconnection strategies deployed for cost and performance concerns are explored.

In Figure 10.1, multiple DRAM devices are interconnected together to form a single memory system that is managed by a single memory controller. In modern computer systems, one or more *DRAM memory controllers* (DMCs) may be contained in the processor package or integrated into a system controller that resides outside of the processor package. Regardless of the location of the DRAM memory controller, its functionality is to accept read and write requests to a given address in memory, translate the request to one or more commands to the memory system, issue those commands to the DRAM devices in the proper sequence and proper timing, and retrieve or store data on behalf of the processor or I/O devices in the system. The internal structures of a system controller are examined in a separate chapter. This chapter focuses on the organization of DRAM devices in the context of multi-device memory systems.

10.2 Basic Nomenclature

The organization of multiple DRAM devices into a memory system can impact the performance of the

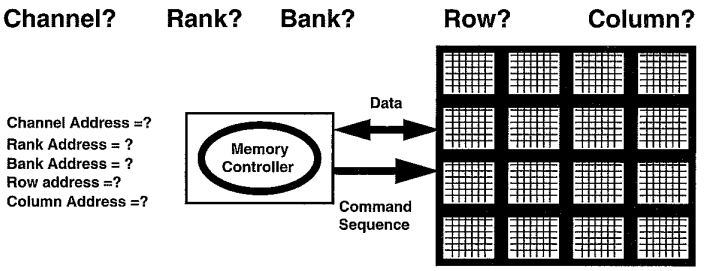


FIGURE 10.1: Multiple DRAM devices connected to a processor through a DRAM memory controller.

memory system in terms of system storage capacity, operating data rates, access latency, and sustainable bandwidth characteristics. It is therefore of great importance that the organization of multiple DRAM devices into larger memory systems be examined in detail. However, the absence of commonly accepted nomenclature has hindered the examination of DRAM memory-system organizations. Without a common basis of well-defined nomenclature, technical articles and data sheets sometimes succeed in introducing confusion rather than clarity into discussions on DRAM memory systems. In one example, a technical data sheet for a system controller used the word *bank* in two bulleted items on the same page to mean two different things. In this data sheet, one bulleted item proclaimed that the system controller could support 6 *banks* (of DRAM devices). Then, several bulleted items later, the same data sheet stated that the same system controller could support SDRAM devices with 4 *banks*. In a second example, an article in a well-respected technical journal examined the then-new i875P system controller from Intel and proceeded to discuss the performance advantage of the system controller due to the fact that the i875P system controller could control 2 *banks* of DRAM devices (it can control two entire channels).

In these two examples, the word *bank* was used to mean three different things. While the meaning of the word *bank* can be inferred from the context in each case, the overloading and repeated use of the word introduces unnecessary confusion into discussions about DRAM memory systems. In this section, the usage of channel, rank, bank, row, and column is defined, and discussions in this and subsequent chapters will conform to the usage in this chapter.

10.2.1 Channel

Figure 10.2 shows three different system controllers with slightly different configurations of the DRAM memory system. In Figure 10.2, each system controller has a single *DRAM memory controller* (DMC), and each DRAM memory controller controls a single channel of memory. In the example labelled as the *typical system controller*, the system controller controls a single 64-bit-wide channel. In modern DRAM memory systems, commodity DRAM memory modules are standardized with 64-bit-wide data busses, and the 64-bit data bus width of the memory module matches the data bus width of the typical personal computer system controller.[1] In the example labelled as *Intel i875P system controller*, the system controller connects to a single channel of DRAM with a 128-bit-wide data bus. However, since commodity DRAM modules have 64-bit-wide data busses,

[1]Commodity memory modules designed for error correcting memory systems are standardized with a 72-bit-wide data bus.

the i875P system controller requires matching pairs of 64-bit wide memory modules to operate with the 128-bit-wide data bus. The paired-memory module configuration of the i875P is often referred to as a *dual channel* configuration. However, since there is only one memory controller, and since both memory modules operate in lockstep to store and retrieve data through the 128-bit-wide data bus, the paired-memory module configuration is, logically, a 128-bit-wide single channel memory system. Also, similar to SDRAM and DDR SDRAM memory systems, standard Direct RDRAM memory modules are designed with 16-bit-wide data busses, and high-performance system controllers that use Direct RDRAM, such as the Intel i850 system controller, use matched pairs of Direct RDRAM memory modules to form a 32-bit-wide channel that operates in lockstep across the two physical channels of memory.

In contrast to system controllers that use a single DRAM memory controller to control the entire memory system, Figure 10.3 shows that the Alpha EV7 processor and the Intel i925x system controller each have

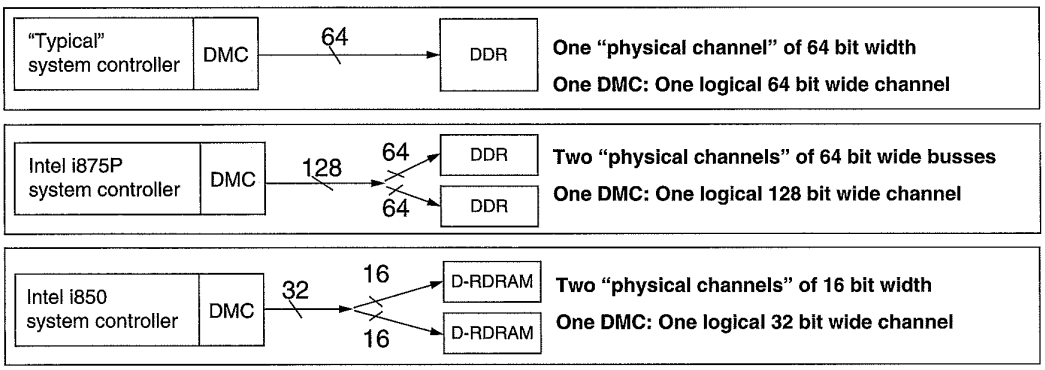


FIGURE 10.2: Systems with a single memory controller and different data bus widths.

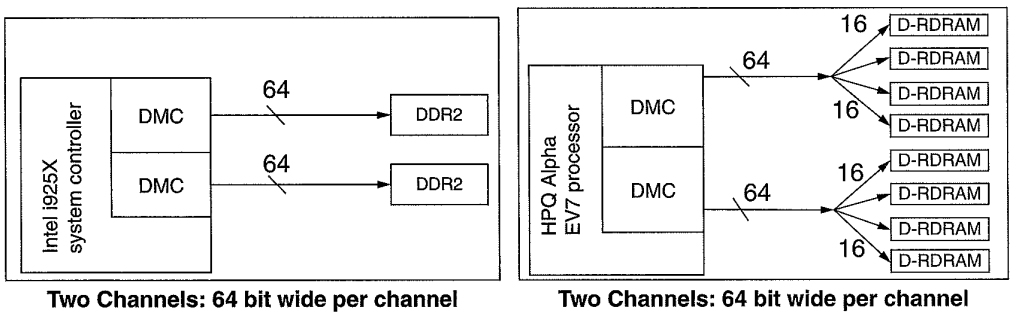


FIGURE 10.3: Systems with two independent memory controllers and two logical channels of memory.

two DRAM controllers that independently control 64-bit-wide data busses.[2] The use of independent DRAM memory controllers can lead to higher sustainable bandwidth characteristics, since the narrower channels lead to longer data bursts per cacheline request, and the various inefficiencies dictated by DRAM-access protocols can be better amortized. As a result, newer system controllers are often designed with multiple memory controllers despite the additional die cost.

Modern memory systems with one DRAM memory controller and multiple physical channels of DRAM devices such as those illustrated in Figure 10.2 are typically designed with the physical channels operating in lockstep with respect to each other. However, there are two variations to the single-controller-multiple-physical-channel configuration. One variation of the single-controller-multiple-physical-channel configuration is that some system controllers, such as the Intel i875P system controller, allow the use of mismatched pairs of memory modules in the different physical channels. In such a case, the i875P system controller operates in an *asymmetric* mode and independently controls the physical channels of DRAM modules. However, since there is only one DRAM memory controller, the multiple physical channels of mismatched memory modules cannot be accessed concurrently, and only one channel of memory can be accessed at any given instance in time. In the asymmetric configuration, the maximum system bandwidth is the maximum bandwidth of a single physical channel.

A second variation of the single-controller-multiple-physical-channel configuration can be found in high-performance FPM DRAM memory systems that were designed prior to the emergence of SDRAM-type DRAM devices that can burst out multiple columns of data with a given column access command. Figure 10.4 illustrates a sample timing diagram of a column access in an SDRAM memory system. Figure 10.4 shows that an SDRAM device is able to return a burst of multiple columns of data for a single column access command. However, an FPM DRAM device supported neither single-access-multiple-burst capability nor the ability to pipeline multiple column access commands. As a result, FPM DRAM

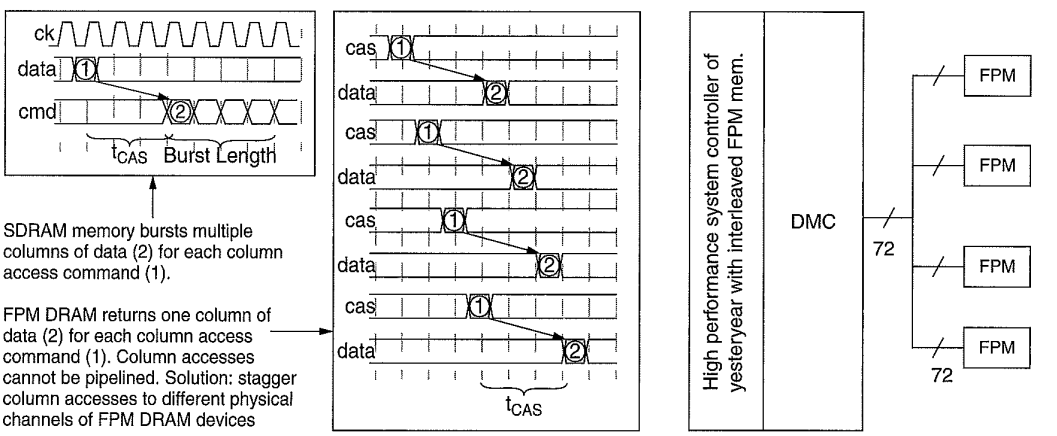


FIGURE 10.4: High-performance memory controllers with four channels of interleaved FPM DRAM devices.

[2]Ignoring additional bitwidths used for error correction and cache directory.

devices need multiple column accesses to retrieve the multiple columns of data for a given cacheline access, column accesses that cannot be pipelined to a single FPM DRAM device.

One solution deployed to overcome the shortcomings of FPM DRAM devices is the use of multiple FPM DRAM channels operating in an interleaved fashion. Figure 10.4 also shows how a sophisticated FPM DRAM controller can send multiple column accesses to different physical channels of memory so that the data for the respective column accesses appears on the data bus in consecutive cycles. In this configuration, the multiple FPM DRAM channels can provide the sustained throughput required in high-performance workstations and servers before the appearance of modern synchronous DRAM devices that can burst through multiple columns of data in consecutive cycles.

10.2.2 Rank

Figure 10.5 shows a memory system populated with 2 ranks of DRAM devices. Essentially, a *rank* of memory is a *"bank"* of one or more DRAM devices that operate in lockstep in response to a given command. However, the word *bank* has already been used to describe the number of independent DRAM arrays within a DRAM device. To lessen the confusion associated with overloading the nomenclature, the word *rank* is now used to denote a set of DRAM devices that operate in lockstep to respond to a given command in a memory system.

Figure 10.5 illustrates a configuration of 2 ranks of DRAM devices in a classical DRAM memory system topology. In the classical DRAM memory system topology, address and command busses are connected to every DRAM device in the memory system,

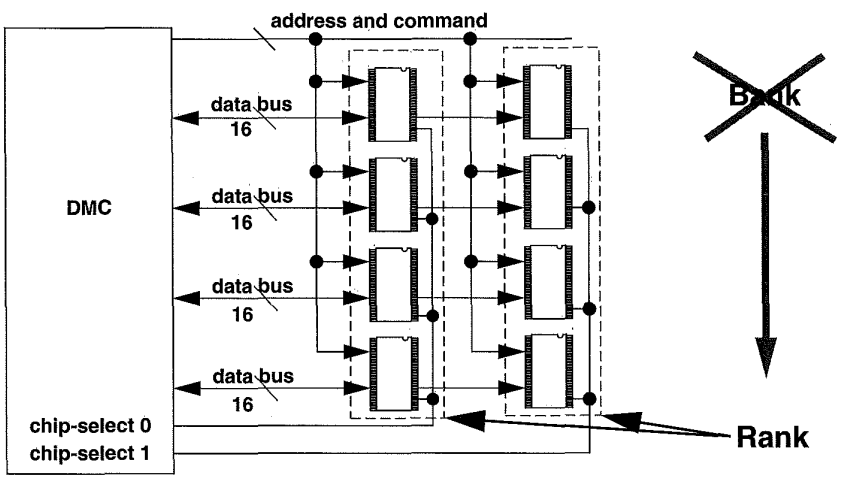


FIGURE 10.5: Memory system with 2 ranks of DRAM devices.

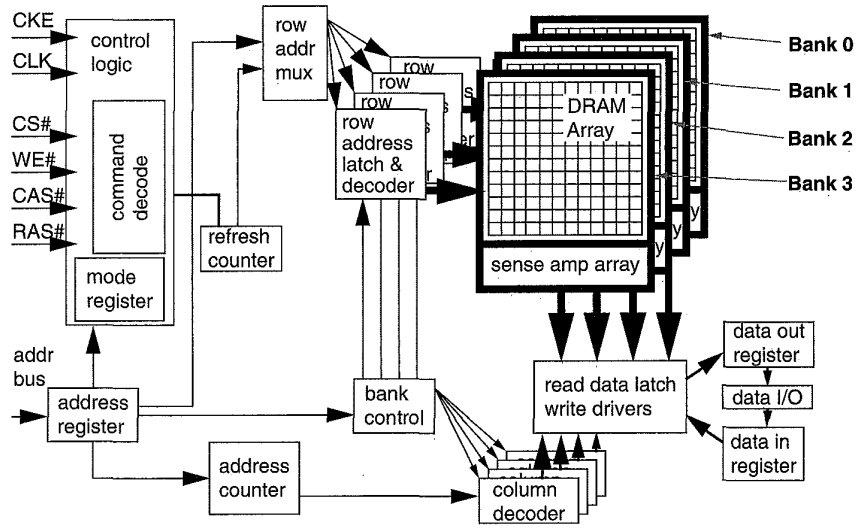


FIGURE 10.6: SDRAM device with 4 banks of DRAM arrays internally.

but the wide data bus is partitioned and connected to different DRAM devices. The memory controller in this classical system topology then uses chip-select signals to select the appropriate rank of DRAM devices to respond to a given command.

In modern memory systems, multiple DRAM devices are commonly grouped together to provide the data bus width and capacity required by a given memory system. For example, 18 DRAM devices, each with a 4-bit-wide data bus, are needed in a given rank of memory to form a 72-bit-wide data bus. In contrast, embedded systems that do not require as much capacity or data bus width typically use fewer devices in each rank of memory—sometimes as few as one device per rank.

10.2.3 Bank

As described previously, the word *bank* had been used to describe a set of independent memory arrays inside of a DRAM device, a set of DRAM devices that collectively act in response to commands, and different physical channels of memory. In this chapter, the word *bank* is only used to denote a set of independent memory arrays inside a DRAM device.

Figure 10.6 shows an SDRAM device with 4 banks of DRAM arrays. Modern DRAM devices contain multiple banks so that multiple, independent accesses to different DRAM arrays can occur in parallel. In this design, each bank of memory is an independent array that can be in different phases of a row access cycle. Some common resources, such as I/O gating that allows access to the data pins, must be shared between different banks. However, the multi-bank architecture allows commands such as read requests to different banks to be pipelined. Certain commands, such as refresh commands, can also be engaged in multiple banks in parallel. In this manner, multiple banks can operate independently or concurrently depending on the command. For example, multiple banks within a given DRAM device can be activated independently from each other—subject to the power constraints of the DRAM device that may specify how closely such activations can occur in a given period of

time. Multiple banks in a given DRAM device can also be precharged or refreshed in parallel, depending on the design of the DRAM device.

10.2.4 Row

In DRAM devices, a *row* is simply a group of storage cells that are activated in parallel in response to a row activation command. In DRAM memory systems that utilize the conventional system topology such as SDRAM, DDR SDRAM, and DDR2 SDRAM memory systems, multiple DRAM devices are typically connected in parallel in a given rank of memory. Figure 10.7 shows how DRAM devices can be connected in parallel to form a rank of memory. The effect of DRAM devices connected as ranks of DRAM devices that operate in lockstep is that a row activation command will activate the same addressed row in all DRAM devices in a given rank of memory. This arrangement means that the size of a row—from the perspective of the memory controller—is simply the size of a row in a given DRAM device multiplied by the number of DRAM devices in a given rank, and a DRAM row spans across the multiple DRAM devices of a given rank of memory.

A *row* is also referred to as a DRAM page, since a row activation command in essence caches a page of memory at the sense amplifiers until a subsequent precharge command is issued by the DRAM memory controller. Various schemes have been proposed to take advantage of locality at the DRAM page level. However, one problem with the exploitation of locality at the DRAM page level is that the size of the DRAM page depends on the configuration of the DRAM device and memory modules, rather than the architectural page size of the processor.

10.2.5 Column

In DRAM memory systems, a column of data is the smallest addressable unit of memory. Figure 10.8 illustrates that, in memory systems such as SDRAM and DDRx[3] SDRAM with topology similar to the memory system illustrated in Figure 10.5, the size of

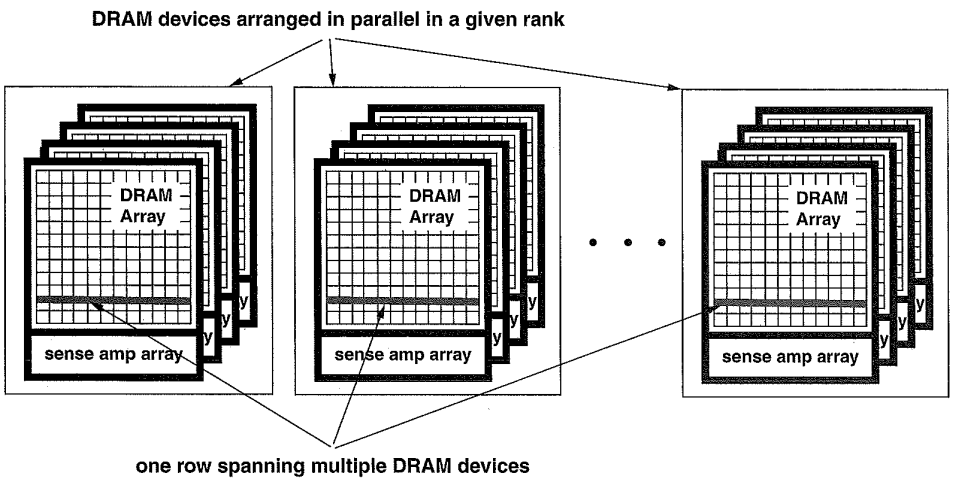


FIGURE 10.7: Generic DRAM devices with 4 banks, 8196 rows, 512 columns per row, and 16 data bits per column.

[3]DDRx denotes DDR SDRAM and evolutionary DDR memory systems such as DDR2 and DDR3 SDRAM memory systems, inclusively.

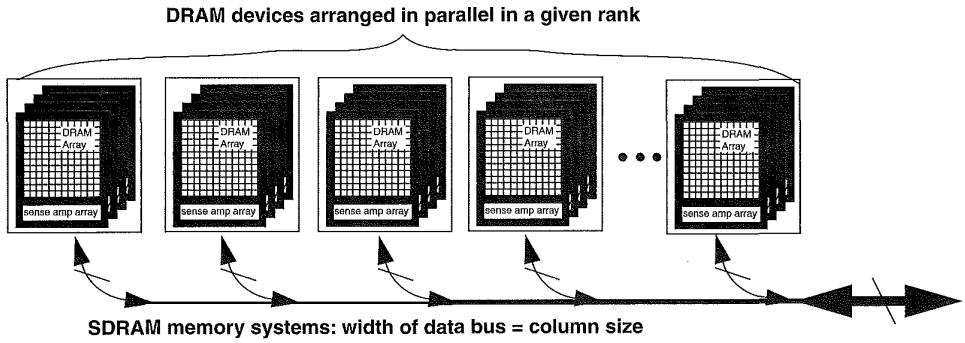


FIGURE 10.8: Classical DRAM system topology; width of data bus equals column size.

a column of data is the same as the width of the data bus. In a Direct RDRAM device, a column is defined as 16 bytes of data, and each read command fetches a single column of data 16 bytes in length from each physical channel of Direct RDRAM devices.

A *beat* is simply a data transition on the data bus. In SDRAM memory systems, there is one data transition per clock cycle, so one beat of data is transferred per clock cycle. In DDRx SDRAM memory systems, two data transfers can occur in each clock cycle, so two beats of data are transferred in a single clock cycle. The use of the beat terminology avoids overloading the word *cycle* in DDRx SDRAM devices.

In DDRx SDRAM memory systems, each column access command fetches multiple columns of data depending on the programmed burst length. For example, in a DDR2 DRAM device, each memory read command returns a minimum of 4 columns of data. The distinction between a DDR2 device returning a minimum burst length of 4 *beats* of data and a Direct RDRAM device returning a single column of data over 8 beats is that the DDR2 device accepts the address of a specific column and returns the requested columns in different orders depending on the programmed behavior of the DRAM device. In this manner, each column is separately addressable. In contrast, Direct RDRAM devices do not reorder data within a given burst, and a 16-byte burst from a single channel of Direct RDRAM devices is transmitted in order and treated as a single column of data.

10.2.6 Memory System Organization: An Example

Figure 10.9 illustrates a DRAM memory system with 4 ranks of memory, where each rank of memory consists of 4 devices connected in parallel, each device contains 4 banks of DRAM arrays internally, each bank contains 8192 rows, and each row consists of 512 columns of data. To access data in a DRAM-based memory system, the DRAM memory controller accepts a physical address and breaks down the address into respective address fields that point to the specific channel, rank, bank, row, and column where the data is located.

Although Figure 10.9 illustrates a uniformly organized memory system, memory system organizations of many computer systems, particularly end-user configurable systems, may be typically non-uniformly organized. The reason that the DRAM memory systems organizations in many computer systems are typically non-uniform is because most computer systems are designed to allow end-users to upgrade the capacity of the memory system by inserting and removing commodity memory modules. To support memory capacity upgrades by the end-user, DRAM controllers have to be designed to

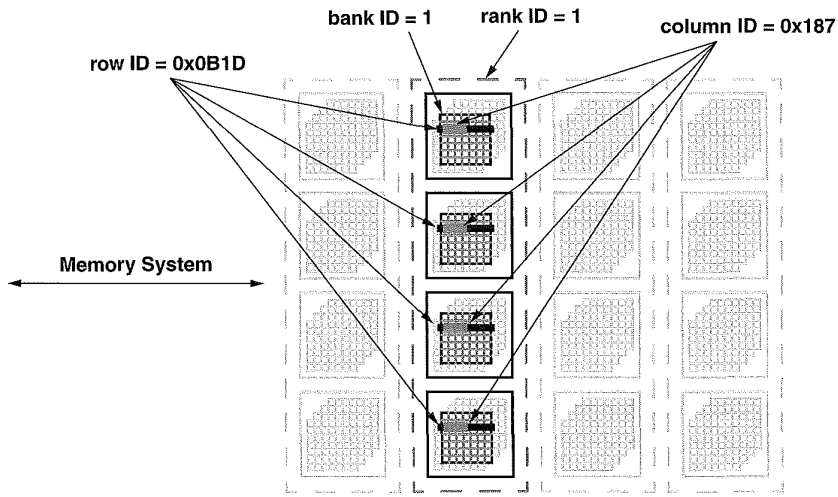


FIGURE 10.9: Location of data in a **DRAM** memory system.

flexibly adapt to different configurations of DRAM devices and modules that the end-user could place into the computer system. This support is provided for through the use of address range registers whose functionality is examined separately in the chapter on memory controllers.

10.3 Memory Modules

The first generations of computer systems allowed end-users to increase memory capacity by providing sockets on the system board where additional DRAM devices could be inserted. The use of sockets on the system board made sense in the era where the price of DRAM devices was quite expensive relative to the cost of the sockets on the system board. In these early computer systems, system boards were typically designed with sockets that allowed end-users to remove and insert individual DRAM devices, usually contained in dual in-line packages (DIPs). The process of memory upgrade was cumbersome and difficult, as DRAM devices had to be individually removed and inserted into each socket. Pins on the DRAM devices may have been bent and not visually detected as such. Defective DRAM chips were difficult to locate, and routing of sockets for a large memory system required large surface areas on the system board. Moreover, it was physically possible to place DRAM devices in the wrong orientation in the socket—180° from the intended placement. Correct placement with proper orientation depended on clearly labelled sockets, clearly labelled devices, and an end-user that paid careful attention while inserting the devices into the sockets.[4] The solution to the problems associated with memory upgradability was the creation and use of memory modules.

Memory modules are essentially miniature system boards that hold a number of DRAM devices. Memory modules provide an abstraction at the module interface so that different manufacturers can manufacture memory upgrades for a given computer system with different DRAM devices. DRAM memory

[4]The author of this text can personally attest to the consequences of inserting chips into sockets with incorrect orientation.

modules also reduce the complexity of the memory upgrade process. Instead of the removal and insertion of individual DRAM chips, memory upgrades with modules containing multiple DRAM chips can be quickly and easily inserted into and removed from a module socket. The first generations of memory modules typically consisted of specially created, system-specific memory modules that a given computer manufacturer used in a given computer system. Over the years, memory modules have obtained a level of sophistication, and they are now specified as a part of the memory-system definition process.

10.3.1 Single In-line Memory Module (SIMM)

In the late 1980s and early 1990s, the personal computer industry first standardized on the use of 30-pin SIMMs and then later moved to 72-pin SIMMs. SIMMs, or *Single In-line Memory Modules,* are referred to as such due to the fact that the contacts on either side of the bottom of the module are electrically identical.

A 30-pin SIMM provides interconnects to 8 or 9 signals on the data bus, as well as power, ground, address, command, and chip-select signal lines between the system board and the DRAM devices. A 72-pin SIMM provides interconnects to 32 to 36 signals on the data bus in addition to the power, ground, address, command, and chip-select signal lines. Typically, DRAM devices on a 30 pin, 1 Megabyte SIMM collectively provide a 9-bit, parity protected data bus interface to the memory system. Personal computer systems in the late 1980s typically used sets of four matching 30-pin SIMMs to provide a 36-bit-wide memory interface to support parity checking by the memory controller. Then, as the personal computer system moved to support memory systems with wider data busses, the 30-pin SIMM was replaced by 72-pin SIMMs in the early 1990s.

10.3.2 Dual In-line Memory Module (DIMM)

In the late 1990s, as the personal computer industry transitioned from FPM/EDO DRAM to SDRAM, 72-pin SIMMs were, in turn, phased out in favor of *Dual In-line Memory Modules* (DIMMs). DIMMs are physically larger than SIMMs and provide a 64- or 72-bit-wide data bus interface to the memory system. The difference between a SIMM and a DIMM is that contacts on either side of a DIMM are electrically different. The electrically different contacts allow a denser routing of electrical signals from the system board through the connector interface to the memory module.

Typically, a DIMM designed for the commodity desktop market contains little more than the DRAM devices and passive resistor and capacitors. These DIMMs are not buffered on either the address path from the memory controller to the DRAM devices or the datapath between the DRAM devices and the memory controller. Consequently, these DIMMs are also referred to as *Unbuffered DIMMs* (UDIMMs).

10.3.3 Registered Memory Module (RDIMM)

To meet the widely varying requirements of systems with end-user configurable memory systems, memory modules of varying capacity and timing characteristics are needed in addition to the typical UDIMM. For example, workstations and servers typically require larger memory capacity than those seen for the desktop computer systems. The problem associated with large memory capacity memory modules is that the large number of DRAM devices in a memory system tends to overload the various multi-drop busses. The large number of DRAM devices, in turn, creates the loading problem on the various address, command, and data busses.

Registered Dual In-line Memory Modules (RDIMMs) alleviate the issue of electrical loading of large numbers of DRAM devices in a large memory system through the use of registers that buffer the address and control signals at the interface of the memory module. Figure 10.10 illustrates that registered memory modules use registers at the interface of the memory module to buffer the address and control signals. In this manner, the registers greatly reduce the number of electrical loads that a memory controller must drive directly, and the signal interconnects in the memory system are divided into two separate segments: between the memory controller and the register and between the register and DRAM devices. The segmentation allows timing characteristics of the memory system to be optimized by limiting

the number of electrical loads, as well as by reducing the path lengths of the critical control signals in individual segments of the memory system. However, the drawback to the use of the registered latches on a memory module is that the buffering of the address and control signals introduces delays into the memory-access latency, and the cost of ensuring signal integrity in a large memory system is paid in terms of additional latency for all memory transactions.

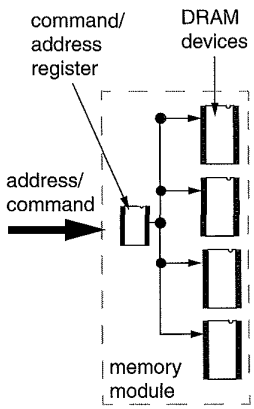


FIGURE 10.10: Registered latches buffer the address and command and also introduce additional latency into the DRAM access.

10.3.4 Small Outline DIMM (SO-DIMM)

Over the years, memory module design has become ever more sophisticated with each new generation of DRAM devices. Currently, different module specifications exist as standardized, multi-source components that an end-user can purchase and reasonably expect trouble-free compatibility between memory modules manufactured by different module manufacturers at different times. To ensure system-level compatibility, memory modules are specified as part of the memory system standards definition process. More specifically, different types of memory modules are specified, with each targeting different markets. Typically, UDIMMs are used in desktop computers, RDIMMs are used in workstation and server systems, and the *Small Outline Dual In-line Memory Module* (SO-DIMM) has been designed to fit into the limited space found in mobile notebook computers.

Figure 10.11 shows the standardized placement of eight DDR2 SDRAM devices in *Fine Ball Grid Array* (FBGA) packages along with the required serial termination resistors and decoupling capacitors on

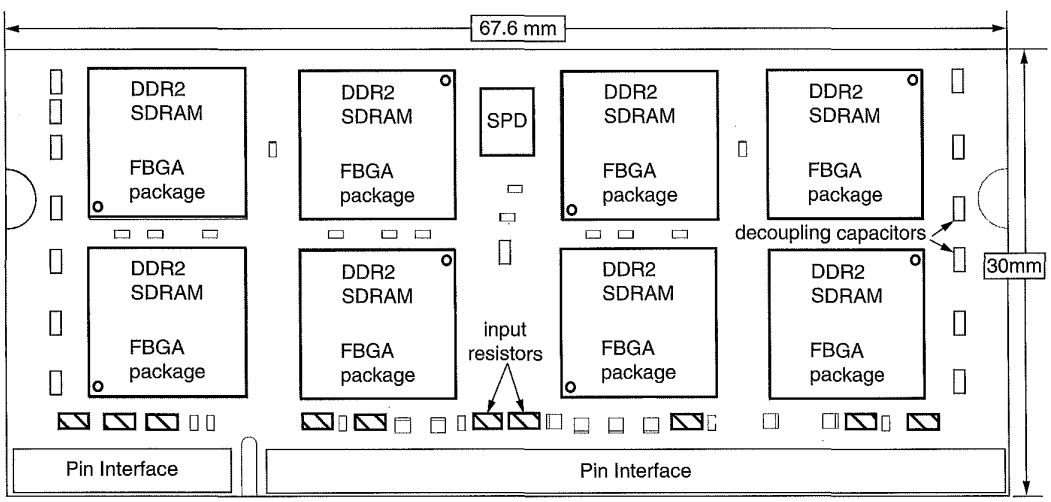


FIGURE 10.11: Component placement specification for a DDR2 SO-DIMM.

a 200-pin SO-DIMM. Figure 10.11 shows that the outline of the SO-DIMM is standardized with specific dimensions: 30 mm × 67.6 mm. The specification of the SO-DIMM dimension illustrates the point that as part of the effort to ensure system-level compatibility between different memory modules and system boards, mechanical and electrical characteristics of SO-DIMMs, UDIMMs, and RDIMMs have been carefully defined. Currently, commodity DRAM devices and memory modules are defined through long and arduous standards-setting processes by DRAM device manufacturers and computer-system design houses.

The standards-setting process enables DRAM manufacturers to produce DRAM devices that are functionally compatible. The standards-setting process further enables memory-module manufacturers to take the functionally compatible DRAM devices and construct memory modules that are functionally compatible with each other. Ultimately, the multi-level standardization enables end-users to freely purchase memory modules from different module manufacturers, using DRAM devices from different DRAM manufacturers, and to enjoy reasonably trouble-free interoperability. Currently, standard commodity DRAM devices and memory modules are specified through the industry organization known as the JEDEC Solid-State Technology Association.[5]

Finally, to further minimize problems in achieving trouble-free compatibility between different DRAM devices and memory module manufacturers, JEDEC provides reference designs to memory module manufacturers, complete with memory module raw card specification, signal trace routings, and a bill of materials. The reference designs further enable memory module manufacturers to minimize their expenditure of engineering resources in the process to create and validate memory module designs, thus lowering the barrier of entry to the manufacturing of high-quality memory modules and enhancing competition in the memory module manufacturing business.

10.3.5 Memory Module Organization

Modern DRAM memory systems often support large varieties of memory modules to give end-users the flexibility of selecting and configuring the desired memory capacity. Since the price of DRAM devices fluctuates depending on the unpredictable commodity market, one memory module organization may be less expensive to manufacture than another organization at a given instance in time, while the reverse may be true at a different instance in time. As a result, a memory system that supports different configurations of memory modules allows end-users the flexibility to purchase and use the most economically organized memory module. However, one issue that memory-system design engineers must account for in providing the flexibility of memory system configuration to the end-user is that the flexibility translates into large combinations of memory modules that may be placed into the memory system at one time. Moreover, multiple organizations often exist for a given memory module capacity, and memory system design engineers must often account for not only different combinations of memory modules of different capacities, but also different modules of different organizations for a given capacity.

Table 10.1 shows that a 128-MB memory module can be constructed from a combination of 16 64-Mbit DRAM devices, 8 128-Mbit DRAM devices, or 4 256-Mbit DRAM devices. Table 10.1 shows that the different memory-module organizations not only use different numbers of DRAM devices, but also present different numbers of rows and columns to the memory controller. To access the memory on the memory module, the DRAM controller must recognize and support the organization of the memory module inserted by the end-user into the memory system. In some cases, new generations of DRAM devices can enable memory module organizations that a memory controller was not designed to support, and incompatibility follows naturally.

[5]JEDEC was once known as the **J**oint **E**lectron **D**evice **E**ngineering **C**ouncil.

10.3.6 Serial Presence Detect (SPD)

Memory modules have gradually evolved as each generation of new memory modules gains additional levels of sophistication and complexity. Table 10.1 shows that a DRAM memory module can be organized as multiple ranks of DRAM devices on the same memory module, with each rank consisting of multiple DRAM devices, and the memory module can have differing numbers of rows and columns. What is not shown in Table 10.1 is that each DRAM memory module may, in fact, have different minimum timing characteristics in terms of minimum t_{CAS}, t_{RAS}, t_{RCD}, and t_{RP} latencies. The variability of the DRAM modules, in turn, increases the complexity that a memory-system design engineer must deal with.

To reduce the complexity and eliminate the confusion involved in the memory upgrading process, the solution adopted by the computer industry is to store the configuration information of the memory module on a read-only memory device whose content can be retrieved by the memory controller as part of the system initialization process. In this manner, the memory controller can obtain the configuration and timing parameters required to optimally access data from DRAM devices on the memory module. Figure 10.12 shows the image of a small flash memory device on a DIMM. The small read-only memory device is known as a *Serial Presence Detect* (SPD) device, and it stores a wide range of variations that can exist between different memory modules. Table 10.2 shows some parameters and values that are stored in the SPD of the DDR SDRAM memory module.

TABLE 10.1 Four different configurations for a 128-MB SDRAM memory module

Capacity	Device Density	Number of Ranks	Devices per Rank	Device Width	Number of Banks	Number of Rows	Number of Columns
128 MB	64 Mbit	1	16	x4	4	4096	1024
128 MB	64 Mbit	2	8	x8	4	4096	512
128 MB	128 Mbit	1	8	x8	4	4096	1024
128 MB	256 Mbit	1	4	x16	4	8192	512

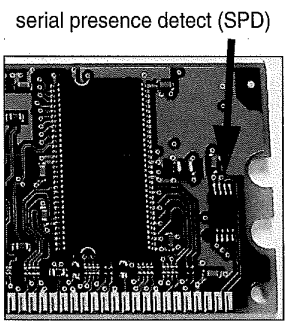


FIGURE 10.12: The SPD device stores memory module configuration information.

TABLE 10.2 Sample parameter values stored in SPD

Configuration	Value (interpreted)
DRAM type	DDR SDRAM
No. of row addresses	16384
No. of column addresses	1024
No. of banks	4
Data rate	400
Module type	ECC
CAS latency	3

10.4 Memory System Topology

In Figure 10.13, a memory system where 16 DRAM devices are connected to a single DRAM controller is shown. In Figure 10.13, the 16 DRAM devices are organized into 4 separate *ranks* of memory. Although all 16 DRAM devices are connected to the same DRAM controller, different numbers of DRAM devices are connected to different networks for the unidirectional address and command bus, the bidirectional data bus, and the unidirectional chip-select lines. In this topology, when a command is issued, electrical signals on the address and command busses are sent to all 16 DRAM devices in the memory system, but the separate chip-select signal selects a set of 4 DRAM devices in a single rank to provide the data for a read command or receive the data for a write command. In this topology, each DRAM device in a given rank of memory is also connected to a subset of the width of the data bus along with three other DRAM devices in different ranks of memory.

Memory system topology determines the signal path lengths and electrical loading characteristics in the memory system. As a result, designers of modern high-performance DRAM memory systems must pay close attention to the topology and organizations of the DRAM memory system. However, due to the evolutionary nature of the memory system, the classic system topology described above has remained essentially unchanged for Fast Page Mode DRAM (FPM), Synchronous DRAM (SDRAM), and Dual Data Rate SDRAM (DDR) memory systems. Furthermore, variants of the classical topology with fewer ranks are expected to be used for DDR2 and DDR3 memory systems.

10.4.1 Direct RDRAM System Topology

One memory system with a topology dramatically different from the classical topology is the Direct RDRAM memory system. In Figure 10.14, four Direct RDRAM devices are shown connected to a single Direct RDRAM memory controller. Figure 10.14 shows that in a Direct RDRAM memory system, the DRAM devices are connected to a well-matched network of interconnects where the clocking network, the data bus, and the command busses are all path-length matched by design. The benefit of the well-matched interconnection network is that signal skew is minimal by design, and electrical signaling rates in the Direct RDRAM memory system can be increased to higher frequencies than a memory system with the classic memory system topology. Modern DRAM systems with conventional multi-rank topology can also match the raw signaling rates of a Direct RDRAM memory system. However, the drawback is that idle cycles must be designed into the access protocol and devoted to system-level synchronization. As a result,

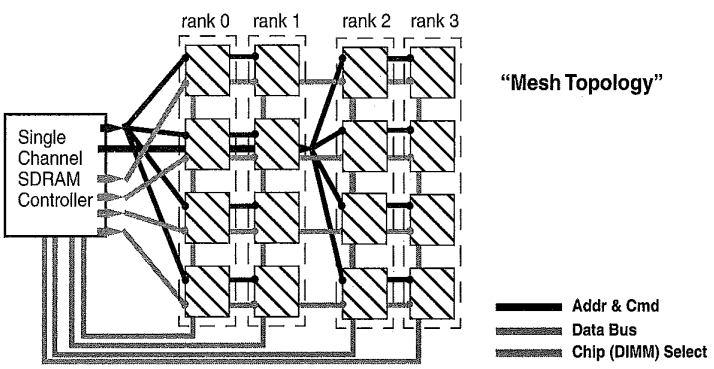


FIGURE 10.13: Topology of a generic DRAM memory system.

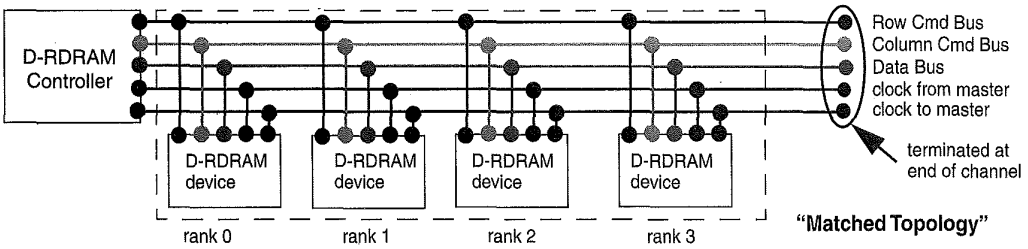


FIGURE 10.14: Topology of a generic Direct RDRAM memory system.

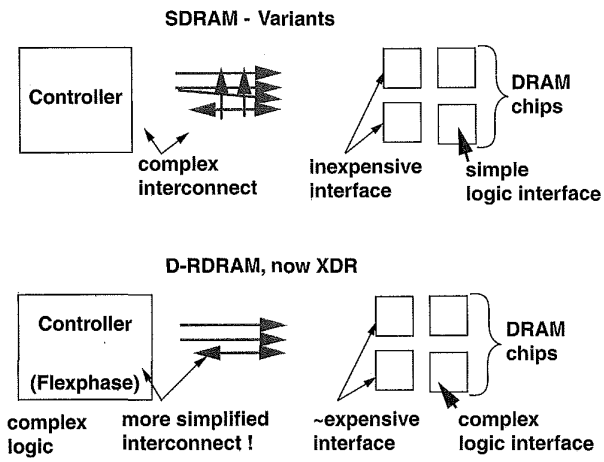


FIGURE 10.15: Philosophy differences.

even when pushed to comparable data rates, multi-rank DRAM memory systems with classical system topologies are somewhat less efficient in terms of data transported per cycle per pin.

The Direct RDRAM memory system achieves higher efficiency in terms of data transport per cycle per pin through the use of a novel system topology. However, in order to take advantage of the system topology and enjoy the benefits of higher pin data rates as well as higher data transport efficiency, Direct RDRAM memory devices are by design more complex than comparable DRAM memory devices that use the classic memory system topology. In DRAM devices, complexity translates directly to increased costs. As a result, the higher data transport efficiency of Direct RDRAM memory systems has to be traded off against relatively higher DRAM device costs.

10.5 Summary

Figure 10.15 shows the difference in philosophy of commodity SDRAM variant devices such as DDR SDRAM and high data rate DRAM memory devices

such as Direct RDRAM and XDR DRAM. Similar to SDRAM variant memory systems, Direct RDRAM and XDR DRAM memory systems are engineered to allow tens of DRAM devices to be connected to a single DRAM controller. However, to achieve high signaling data rates, Direct RDRAM and XDR DRAM memory systems rely on the re-engineering of the interconnection interface between the memory controller and the DRAM devices. In these high data rate DRAM devices, far more circuitry is placed on the DRAM devices in terms of pin interface impedance control and signal drive current strength.

CHAPTER 11

Basic DRAM Memory-Access Protocol

The basic structures of modern DRAM devices and memory system organizations have been described in previous chapters. This chapter continues the examination of the DRAM memory system with a discussion on a basic DRAM memory-access protocol. A DRAM memory-access protocol defines commands and timing constraints that a DRAM memory controller uses to manage the movement of data between itself and DRAM devices. The basic DRAM memory-access protocol described in this chapter is generic in nature, and it can be broadly applied to modern memory systems such as SDRAM, DDR SDRAM, DDR2 SDRAM, and DDR3 SDRAM memory systems. The examination of the generic DRAM memory-access protocol begins by focusing on basic DRAM commands and the sequence of events that occurs in a DRAM device in the execution of the basic DRAM commands. Different DRAM memory systems, in particular, high-performance-oriented and low-power-oriented DRAM memory systems, have slightly differing access protocols. Specialized or high-performance DRAM memory systems such as Direct RDRAM, GDDRx, and FCRAM have slightly varying sets of DRAM commands and different command timings and interactions as those described in this chapter. However, the fundamental command interactions in all DRAM memory systems are substantially similar to each other, and understanding of the basic DRAM memory-access protocol can aid in the understanding of more complex memory-access protocols in more specialized DRAM memory systems.

11.1 Basic DRAM Commands

A detailed examination of any DRAM memory-access protocol is a difficult and complex task. The complexity of the task arises from the number of combinations of commands in modern DRAM memory systems. Fortunately, a basic memory-access protocol can be modelled by accounting for a limited number of basic DRAM commands.[1] In this section, five basic DRAM commands are described. The descriptions of the basic commands form the foundation of the DRAM memory-access protocol examined in this chapter. The interactions of the basic DRAM commands are then used to determine the latency response and sustainable bandwidth characteristics of DRAM memory systems in this book.

[1]Modern DRAM devices such as Direct RDRAM and DDR2 SDRAM devices support larger sets of commands. However, most are used to manage the electrical characteristics of the DRAM devices, and only indirectly impacts access latency and sustainable bandwidth characteristics of a DRAM memory system at a given operating frequency. Commands such as those used to manage power-down states and self-refresh modes are not examined in this chapter for the sake of simplification.

Throughout this chapter, the SDRAM device illustrated in Figure 11.1 is used as the generic DRAM device for the purpose of defining the basic memory-access protocol. The generic DRAM memory-access protocol described in this chapter is based on a resource usage model. That is, the generic DRAM memory-access protocol assumes that two different commands can be fully pipelined on a given DRAM device as long as they do not require the use of a shared resource at a given time.[2] Figure 11.1 illustrates that in the movement of data in modern DRAM devices, a given command would progress through different phases of operation to facilitate the movement of data, and in each phase, a given command would require the use of certain resources that often cannot be shared with a different command. Figure 11.1 illustrates four overlapped phases of operation for an abstract DRAM command. In phase 1, the command is transported through the address and command busses and decoded by the DRAM device. In phase 2, data is moved within a bank, either from the cells to the sense amplifiers or from the sense amplifiers

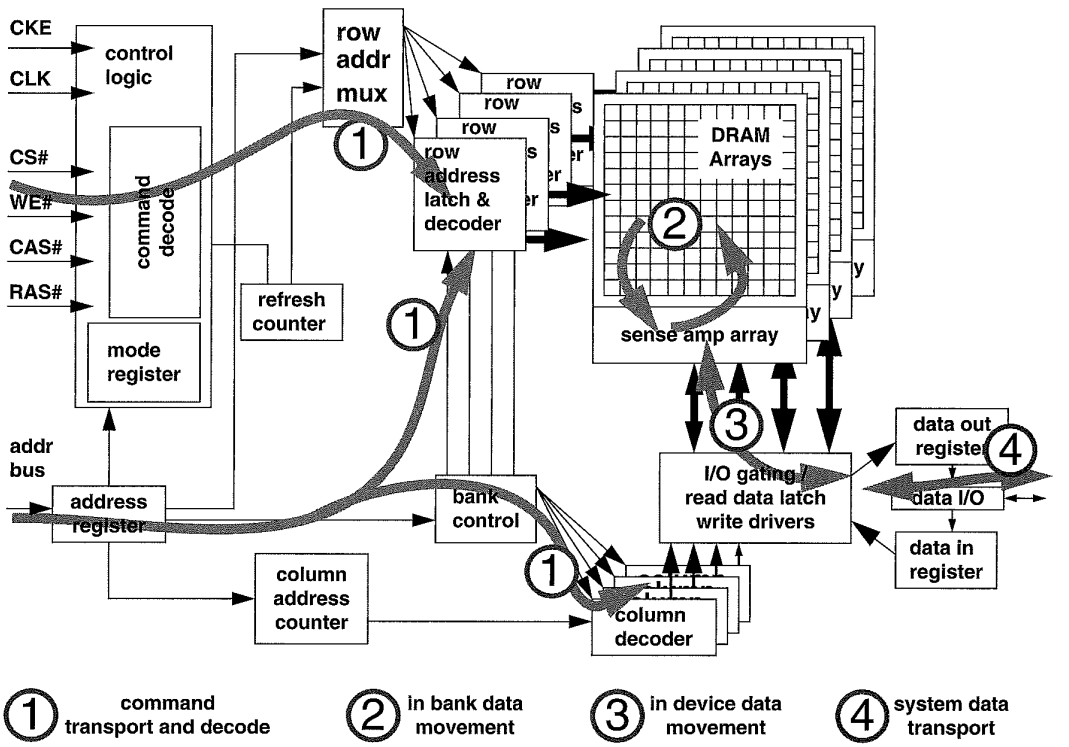


FIGURE 11.1: Command and data movement on a generic SDRAM device.

[2]Additional timing parameter constraints on the scheduling of DRAM commands such as t_{RRD} and t_{FAW} exist in addition to the resource-sharing based timing parameter constraints. These timing parameters are used to limit the maximum current draw of DRAM devices, so while the resource on the DRAM device is not used, these timing parameters limit the rate of resource utilization to limit peak power consumption characteristics on a given DRAM device.

back into the DRAM arrays. In phase 3, the data is moved through the shared I/O gating circuit and then through the read latches or write drivers, as appropriate in each case. In phase 4, read data is placed onto the data bus by the DRAM device in the case of a column-read command or by the memory controller in the case of a column-write command. Since the data bus may be connected to multiple ranks of memory, no two commands to different ranks of memory can use the shared data bus at the same instance in time.

A DRAM-access protocol also defines the timing constraints between combinations of consecutive DRAM commands. In this chapter, the description of the DRAM memory-access protocol begins with the examination of individual DRAM commands and progresses with the examination of combinations of DRAM commands. The impact of power-limitation-based constraints such as the row-to-row activation delay and the four-bank activation window are then described in detail in Sections 11.3.2 and 11.3.3, respectively.

11.1.1 Generic DRAM Command Format

Figure 11.2 illustrates the progression of a generic DRAM command. In Figure 11.2, the time period that it takes to transport the command from the DRAM controller to the DRAM device is illustrated and labelled as t_{CMD}. Figure 11.2 also illustrates $t_{parameter1}$, a generic timing parameter that measures the duration of "phase 2." In Figure 11.2, $t_{parameter1}$ defines the amount of time that the described command spends in the use of the selected bank, and $t_{parameter2}$ defines the amount of time that the described command spends in the use of resources common to multiple banks of DRAM arrays in the same DRAM device. In this manner, $t_{parameter1}$ also denotes the minimum amount of time that must pass between the scheduling of two commands whose relative timing is limited by the sharing of resources within a given bank of DRAM arrays, and $t_{parameter2}$ also denotes the minimum amount of time that must pass between the start of two commands whose relative timing is limited by the sharing of resources by multiple banks of DRAM arrays within the same DRAM device.

DRAM commands are abstractly defined in this chapter, and the abstraction separates the actions of each command from the timing-specific nature of each action in specific DRAM-access protocols. That is, the abstraction enables the same set of DRAM command interactions to be applied to different DRAM memory systems with different timing parameter values. By abstracting out protocol-specific timing characteristics, DRAM commands can be described in abstract terms. The generic DRAM memory-access protocol, in turn, enables abstract performance analysis of DRAM memory systems that can then be broadly applied to different memory systems and retain relevance in the cross comparison.

11.1.2 Summary of Timing Parameters

The basic timing parameters used in the examination of the basic DRAM-access protocol are summarized in Table 11.1. The timing parameters summarized in Table 11.1 are far from a complete set of timing parameters used in the description of every modern DRAM memory-access protocol. Nevertheless, the limited set of timing parameters described in Table 11.1 is sufficient to characterize and illustrate

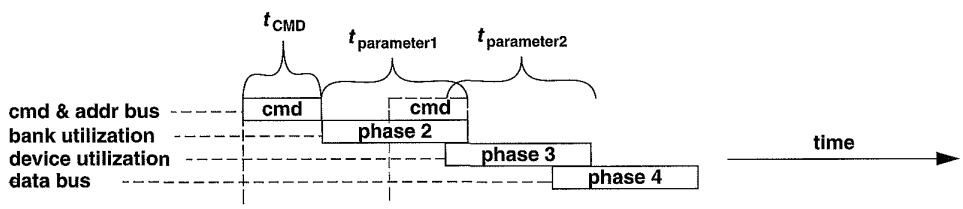


FIGURE 11.2: Different phase of an abstract DRAM command in a generic DRAM device.

the operations of a basic DRAM memory-access protocol of a modern DRAM memory system.

11.1.3 Row Access Command

Figure 11.3 abstractly illustrates the progression of a row access command. The row access command is also known as the row activation command. The purpose of a row access command is to move data from the cells in the DRAM arrays to the sense amplifiers and then restore the data back into the cells in the DRAM arrays as part of the same command. Two timing parameters are associated with a row access command: t_{RCD} and t_{RAS}. The time it takes for the row access command to move data from the DRAM cell arrays to the sense amplifiers is known as the *Row-Column (Command) Delay*, t_{RCD}. After t_{RCD} time from the assertion of the row access command, the entire row of activated data is held in the sense amplifiers, and subsequent column read or column-write commands can then move data between the sense amplifiers and the memory controller through the data bus.

TABLE 11.1 Summary of timing parameters used in a generic DRAM-access protocol

Parameter	Description	Illustration
t_{AL}	Added Latency to column accesses, used in DDRx SDRAM devices for posted CAS commands.	Figure 11.11
t_{BURST}	Data burst duration. The time period that data burst occupies on the data bus. Typically 4 or 8 beats of data. In DDR SDRAM, 4 beats of data occupy 2 full clock cycles.	Figure 11.4
t_{CAS}	Column Access Strobe latency. The time interval between column access command and the start of data return by the DRAM device(s). Also known as t_{CL}.	Figure 11.4
t_{CCD}	Column-to-Column Delay. The minimum column command timing, determined by internal burst (prefetch) length. Multiple internal bursts are used to form longer burst for column reads. t_{CCD} is 2 beats (1 cycle) for DDR SDRAM, and 4 beats (2 cycles) for DDR2 SDRAM.	Figure 11.4
t_{CMD}	Command transport duration. The time period that a command occupies on the command bus as it is transported from the DRAM controller to the DRAM devices.	Figure 11.2
t_{CWD}	Column Write Delay. The time interval between issuance of the column-write command and placement of data on the data bus by the DRAM controller.	Figure 11.5
t_{FAW}	Four (row) bank Activation Window. A rolling time-frame in which a maximum of four-bank activation can be engaged. Limits peak current profile in DDR2 and DDR3 devices with more than 4 banks.	Figure 11.32
t_{OST}	ODT Switching Time. The time interval to switching ODT control from rank to rank.	Figure 11.19
t_{RAS}	Row Access Strobe. The time interval between row access command and data restoration in a DRAM array. A DRAM bank cannot be precharged until at least t_{RAS} time after the previous bank activation.	Figure 11.3
t_{RC}	Row Cycle. The time interval between accesses to different rows in a bank. $t_{RC} = t_{RAS} + t_{RP}$.	Figure 11.6
t_{RCD}	Row to Column command Delay. The time interval between row access and data ready at sense amplifiers.	Figure 11.3
t_{RFC}	Refresh Cycle time. The time interval between Refresh and Activation commands.	Figure 11.7
t_{RP}	Row Precharge. The time interval that it takes for a DRAM array to be precharged for another row access.	Figure 11.6

(*continued*)

TABLE 11.1 (*continued*)

Parameter	Description	Illustration
t_{RRD}	Row activation to Row activation Delay. The minimum time interval between two row activation commands to the same DRAM device. Limits peak current profile.	Figure 11.32
t_{RTP}	Read to Precharge. The time interval between a read and a precharge command.	Figure 11.13
t_{RTRS}	Rank-to-rank switching time. Used in DDR and DDR2 SDRAM memory systems; not used in SDRAM or Direct RDRAM memory systems. One full cycle in DDR SDRAM.	Figure 11.18
t_{WR}	Write Recovery time. The minimum time interval between the end of a write data burst and the start of a precharge command. Allows sense amplifiers to restore data to cells.	Figure 11.5
t_{WTR}	Write To Read delay time. The minimum time interval between the end of a write data burst and the start of a column-read command. Allows I/O gating to overdrive sense amplifiers before read command starts.	Figure 11.5

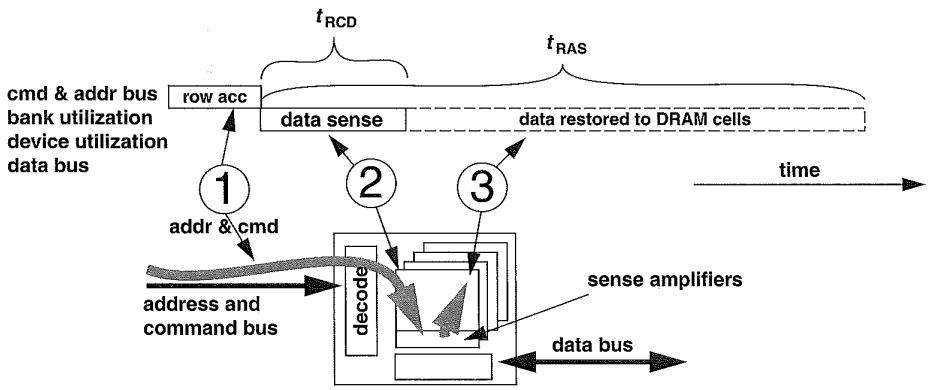


FIGURE 11.3: Row access command and timing.

After t_{RCD} time from the assertion of the row access command, data is available at the sense amplifiers, but not yet fully restored to the DRAM cells. The time it takes for a row access command to discharge and restore data from the row of DRAM cells is known as the *Row Access Strobe latency* or t_{RAS}. After t_{RAS} time from the assertion of the row access command, the sense amplifiers are assumed to have completed data restoration to the DRAM arrays, and the sense amplifiers can then be precharged for another row access to a different row in the same bank of DRAM arrays.

11.1.4 Column-Read Command

Figure 11.4 illustrates the progression of a column-read command. A column-read command moves data from the array of sense amplifiers of a given bank of DRAM arrays through the data bus back to the memory controller. Three basic timing parameters are associated with a column-read command: t_{CAS}, t_{CCD}, and t_{BURST}. The *Column Access Strobe Latency* (t_{CAS}, or t_{CL}) is the time it takes for the DRAM device to place the requested data onto the data bus after issuance of the column-read command. Modern DRAM devices

move data internally in short and continuous bursts. Figure 11.4 illustrates the case where the DRAM device internally moves the data in two short burst durations, but data is placed onto the data bus in a longer, continuous burst. The internal burst length of the DRAM device is labelled as t_{CCD} in Figure 11.4, and the duration of the data burst on the data bus for a single column-read command is labelled as t_{BURST}. The timing parameter t_{CCD} represents the timing of minimum burst duration, or minimum column-to-column command timing. The minimum burst duration is determined by the prefetch length of the DRAM device. For example, the prefetch length of the DDR SDRAM device is 2 beats of data, so t_{CCD} is one full clock cycle in DDR SDRAM devices; the prefetch length of the DDR2 SDRAM device is 4 beats of data, so t_{CCD} is two full clock cycles in DDR2 SDRAM devices; and so on.

The difference in the prefetch length of column access commands has limited impact on the generic memory-access protocol as long as t_{CCD} is shorter than t_{BURST}. In the case where t_{CCD} is longer than t_{BURST}, the intra-rank column access commands are limited by t_{CCD} rather than t_{BURST}. Otherwise, the only case where the differences in the prefetch lengths impact the memory-access protocol occurs when the column-read command is followed immediately by a precharge command. The read-to-precharge timing is examined separately in Section 11.2.2.

11.1.5 Column-Write Command

Figure 11.5 illustrates the progression of a column-write command. A column-write command moves data from the memory controller to the sense amplifiers of the targeted bank. The column-write command goes through a similar set of overlapped phases as the column-read command, but the direction of data movement differs between a column-read command and a column-write command. As a result, the ordering of the phases is reversed between the column-read and the column-write commands.

One timing parameter associated with a column-write command is t_{CWD}, *column write delay*. The column-write delay specifies the timing between assertion of the column-write command on the command bus and the placement of the write data onto the data bus by the memory controller. Different memory-access protocols have different settings for t_{CWD}. Figure 11.5 shows that in SDRAM devices, write data is placed onto the data bus at the same time as the column-write command, and t_{CWD} is zero. In DDR SDRAM devices, t_{CWD} is specified as one clock cycle in the memory system. In DDR2 SDRAM memory-access protocol, t_{CWD} is specified as one cycle less than t_{CAS}, and t_{CWD} has a range of programmability in the DDR3 SDRAM memory-access protocol between five and eight cycles. Finally, Figure 11.5 also illustrates t_{WR}, the write recovery time, and

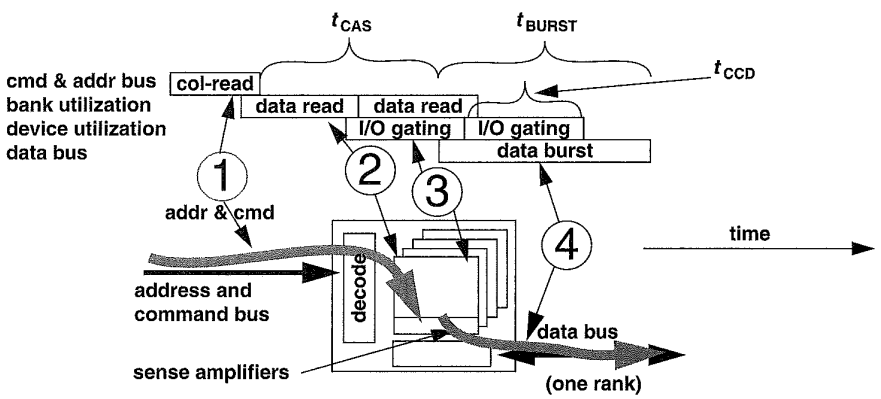


FIGURE 11.4: Column-read command and timing.

t_{WTR}, the write-to-read turnaround time. The write recovery time, t_{WR}, is the time it takes for the write data to propagate into the DRAM arrays, and it must be respected in the case of a precharge command that follows the write command. The write-to-read time, t_{WTR}, accounts for the time that the I/O gating resources are released by the write command, and it must be respected in the case of a read command that follows the write command.

11.1.6 Precharge Command

Data access in a typical DRAM device is composed of a two-step process. First, a row access command moves data from the array of DRAM cells to the array of sense amplifiers. Then, after an entire row of data is moved into the sense amplifiers by the row access command, that data is cached by the sense amplifiers for subsequent column access commands to move data between the DRAM device and the DRAM controller. The precharge command completes the row access sequence as it resets the sense amplifiers and the bitlines and prepares them for another row access command to the same array of DRAM cells. Figure 11.6 illustrates the progression of a precharge command. The timing parameter associated with the (row) precharge command is t_{RP}. That is, t_{RP} time after the assertion of the precharge command, the bitlines and sense amplifiers of the selected bank are properly precharged, and a subsequent row access command can be sent to the just-precharged bank of DRAM cells.

The two row-access-related timing parameters, t_{RP} and t_{RAS}, can be combined to form t_{RC}, the row cycle time. The row cycle time of a given DRAM device denotes the minimum amount of time that a DRAM device needs to bring data from the DRAM cell arrays into the sense amplifiers, restore the data to the DRAM cells, and precharge the bitlines to the reference voltage level for another row access command. The row cycle time is the fundamental limitation to the speed at which data can be retrieved from different rows within the same DRAM bank. As a result, t_{RC} is also commonly referred to as the random row-cycle time of a DRAM device.

11.1.7 Refresh Command

The word "DRAM" is an acronym for Dynamic Random-Access Memory. The nature of the non-persistent charge storage in the DRAM cells means that the electrical charge stored in the storage capacitors will gradually leak out through the access transistors. Consequently, to maintain data integrity, data values stored in DRAM cells must be periodically read out and restored to their respective, full voltage level before the

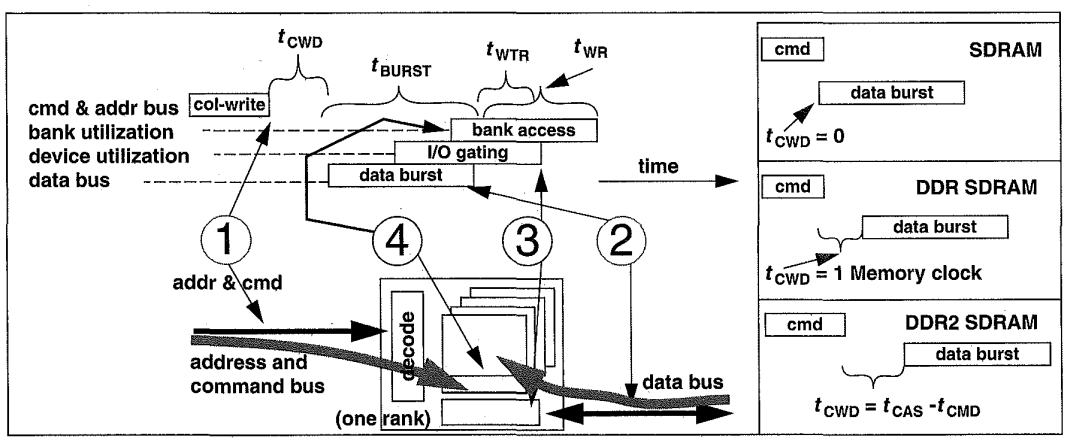


FIGURE 11.5: Column-write command and timing for DDR SDRAM and DDR2 SDRAM devices.

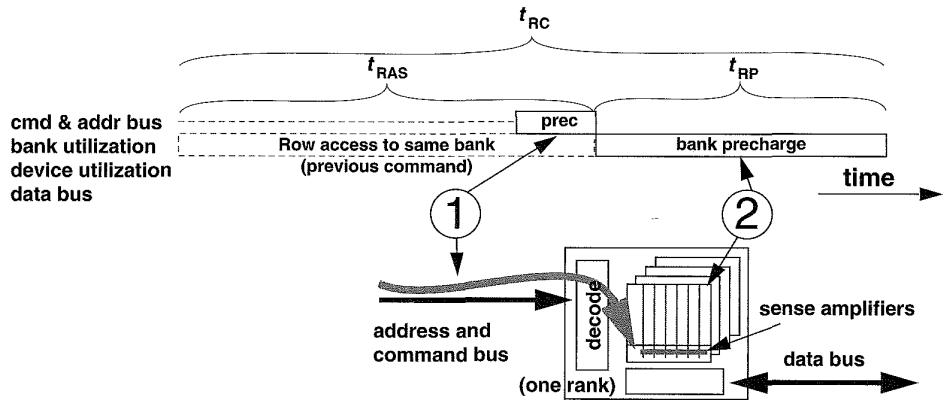


FIGURE 11.6: Row precharge command and timing.

stored electrical charges decay to indistinguishable levels. The refresh command accomplishes the task of data read-out and restoration in DRAM devices, and as long as the time interval between refresh commands made to a given row of a DRAM array is shorter than the worst-case data decay time, DRAM refresh commands can be used to ensure data integrity. The drawback to the refresh mechanism is that the refresh action consumes available bandwidth and power. Consequently, different refresh mechanisms are used in different systems; some are designed to minimize controller complexity, and some are designed to minimize bandwidth impact, while still others are designed to minimize power consumption.

To simplify the control complexity associated with the refresh command, most DRAM devices use a refresh row address register to keep track of the address of the last refreshed row. Typically, the memory controller sends a single refresh command to the DRAM device, and the DRAM device increments the address in the refresh row address register and goes through a row cycle for all rows with that row address in all of the banks in the DRAM device. Figure 11.7 illustrates a basic all-banks-concurrent refresh command that modern DRAM memory controllers use to send a single refresh command to refresh one row of DRAM cells in all banks. When this all-banks-concurrent basic refresh command is issued, the DRAM device takes the row address from the refresh address register and then sends the same row address to all banks to be refreshed concurrently. As illustrated in Figure 11.7, the single refresh command to all banks takes one refresh cycle time t_{RFC} to complete.

Table 11.2 shows the general trend of refresh cycle times in DDR and DDR2 SDRAM devices. With increasing DRAM device density, an increasing number of DRAM cells must be refreshed. The choice that DRAM manufacturers have apparently made, for larger DDR2 devices, is to keep the number of refresh commands per 64-ms period constant despite the doubling of the number of rows in successive generations of higher capacity DRAM devices. Consequently, regardless of the capacity of the DRAM device in the system, the memory controller will send 8192 refresh commands to the DRAM device every 64 ms. However, in high-capacity DRAM devices, there are more than 8192 rows, and each refresh command must refresh 2, 4, or 8 rows in these DRAM devices. In the case of the 4-Gbit DDR2 SDRAM device, there are eight times the number of rows than the number of refresh commands. Consequently, a given 4-Gbit DDR2 SDRAM device will cycle through and refresh 8 rows with each refresh command—a sequence of events that takes a long time to complete. Table 11.2 shows that each refresh command in the 4-Gbit DDR2 SDRAM device takes 327.5 ns to complete.

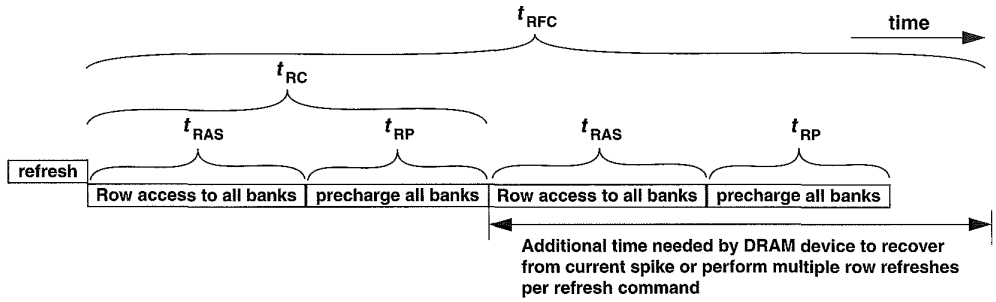


FIGURE 11.7: All-banks-concurrent row refresh timing.

TABLE 11.2 Refresh cycle times for DDR and DDR2 SDRAM devices

DRAM Device Family	Voltage	DRAM Device Capacity	Number of Banks	Number of Rows	Row Size	Refresh Count	t_{RC}	t_{RFC}
DDR	2.5V	256 Mb	4	8192	1 kB	8192	60 ns	67 ns
		512 Mb	4	8192	2 kB	8192	55 ns	70 ns
DDR2	1.8V	256 Mb	4	8192	1 kB	8192	55 ns	75 ns
		512 Mb	4	16384	1 kB	8192	55 ns	105 ns
		1024 Mb	8	16384	1 kB	8192	54 ns	127.5 ns
		2048 Mb	8	32768	1 kB	8192	~	197.5 ns
		4096 Mb	8	65536	1 kB	8192	~	327.5 ns

DRAM device design engineers and DRAM memory system design engineers are actively exploring alternatives to the bank-concurrent refresh command. Some advanced memory systems are designed in such a manner that the controller manually injects row cycle reads to individual banks. The per-bank refresh scheme can decrease the bandwidth impact of refresh commands at the cost of increased complexity in the memory controller.

11.1.8 A Read Cycle

Figure 11.8 illustrates a read cycle in generic DRAM memory systems such as SDRAM and DDRx SDRAM memory systems. In a typical, modern DRAM device, each row access command brings thousands of bits of data to the array of sense amplifiers in a given bank. A subsequent column-read command then brings tens or hundreds of those bits of data through the data bus into the memory controller. For applications that access data by streaming through memory, keeping thousands of bits of a given row of data active at the sense amplifiers ensures that subsequent memory reads from the same row do not incur the latency or energy cost of another row access. In contrast, applications that are not likely to access data in adjacent locations favor memory systems that immediately precharge the DRAM arrays after each row access to prepare the DRAM bank for a subsequent access to a different row within the same bank. Figure 11.8 illustrates a sequence of commands issued in rapid succession to access one bank of DRAM cells. Data is brought in from the DRAM cells to the sense amplifiers by the row access command. After t_{RCD} time,

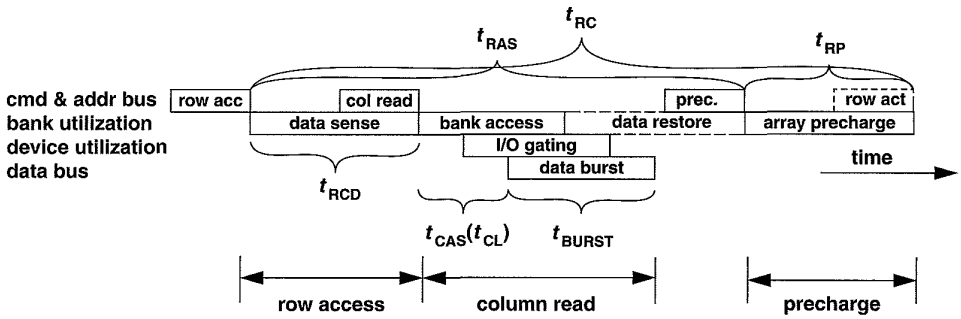


FIGURE 11.8: A read cycle.

data from the requested row is resolved by the sense amplifiers, and the memory controller can then issue column read or write commands to the DRAM device. Concurrent with the issuance of column access commands, the memory device actively restores data from the sense amplifiers to the DRAM cells. Then, after t_{RAS} time from the initial issuance of the row access command, the DRAM cells are ready for a precharge command to reset the bitlines and the sense amplifiers. Collectively, memory systems that immediately precharge a bank to prepare it for another access to a different row are known as *close-page* memory systems. Memory systems that keep rows active at the sense amplifiers are known as *open-page* memory systems.

11.1.9 A Write Cycle

Similar to the illustration of a read cycle in Figure 11.8, Figure 11.9 illustrates a write cycle in a generic DRAM memory system. In modern DRAM devices, the row cycle time is limited by the duration of the write cycle. That is, a row cycle time is defined as the minimum time period that a DRAM device needs to provide access to any data to any row in a given bank of DRAM cells. Implicitly, the access in the definition of a row access can be a read access or a write access. In the case of a write access, data must be provided by the memory controller, driven through the data bus, passed through the I/O gating multiplexors, overdrive the sense amplifiers, and finally stored into the DRAM cells. This complex series of actions must be completed before the precharge command that completes the sequence can proceed. As a result, the row cycle time must be defined so that it can account for the row access time, the column write delay, the data transport time, the write data restore time, as well as the precharge time. That is, to account for the timing of the more complex write access time, t_{RAS} must be long enough to account for t_{RCD}, t_{CWD}, t_{CCD},[3] and t_{WR}. As a result, the timing parameter t_{RAS} must be set so that it is at least equal to $t_{RCD} + t_{CWD} + t_{CCD} + t_{WR}$ in SDRAM, DDR SDRAM, and DDR2 SDRAM devices, subject to the clock cycle granularity of the respective latency values, thus demonstrating the write cycle time constraint of the row cycle time in these commodity DRAM devices.

11.1.10 Compound Commands

In the previous discussion about the read cycle, Figure 11.8 illustrates a read cycle in a generic DRAM memory system by issuing a sequence of three separate commands. As part of the evolution of DRAM

[3]DRAM devices such as DDR SDRAM and DDR2 SDRAM use multiple internal data bursts to form longer burst durations. The t_{RAS} definition needs only account for the shortest t_{BURST} duration possible, t_{CCD}.

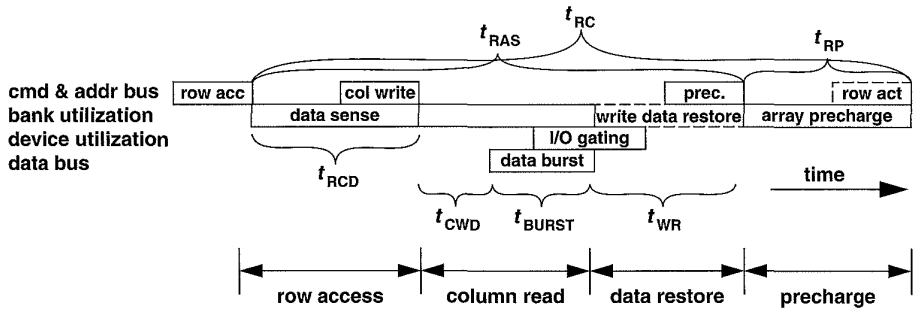


FIGURE 11.9: A write cycle.

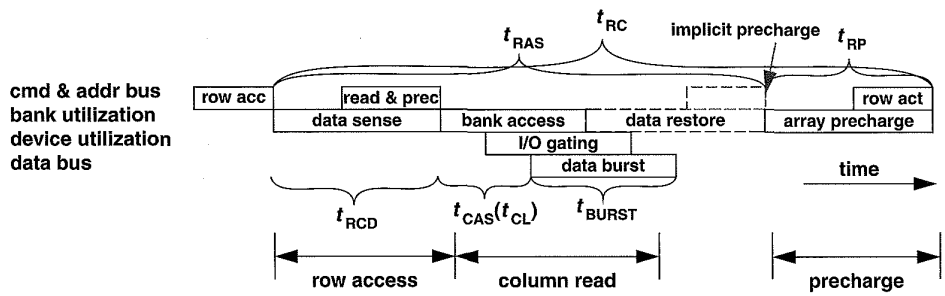


FIGURE 11.10: A read cycle with a row access command and a column-read-and-precharge command.

devices and architecture, some DRAM devices have been designed to support compound commands that perform more complex series of actions. Figure 11.10 shows the same sequence of DRAM commands as presented in Figure 11.8. However, the simple column-read command in Figure 11.8 is replaced with a compound *column-read-and-precharge* command in Figure 11.10. As the name implies, the column-read-and-precharge command combines a column-read command and a precharge command into a single command. The advantage of a column-read-and-precharge command is that for close-page memory systems that precharge the DRAM bank immediately after a read command, the column-read-and-precharge command reduces the bandwidth requirement on the command and address bus. The implicit precharge command means that the DRAM memory controller can now place a different command on the address and command bus that a separate precharge command would have otherwise occupied.

Figure 11.10 shows a column-read-and-precharge command as issued by the memory controller to the DRAM device in the earliest time slot possible after the row access command while still respecting the t_{RCD} timing requirement, but the implicit precharge command is delayed so that it does not violate the t_{RAS} timing requirement. Modern DRAM devices such as DDR2 SDRAM devices have implemented a feature that is referred to as t_{RAS} lockout to ensure that the auto-precharge component of the column read-and-precharge command will not violate the t_{RAS} timing requirement. That is, in the case where a column-read-and-precharge command is issued into the DRAM device before the DRAM device has

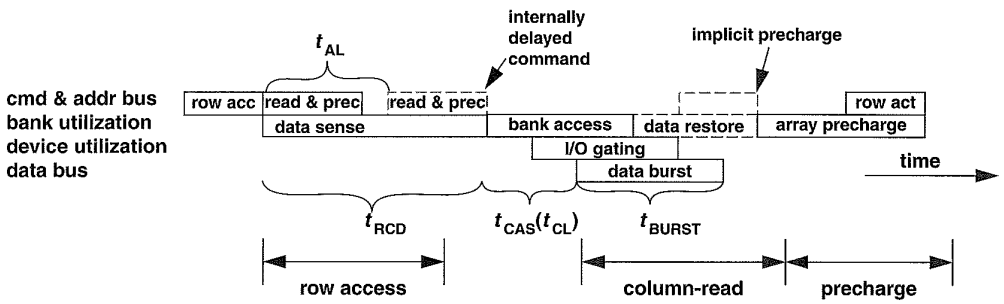


FIGURE 11.11: Posted CAS defers CAS commands in DRAM devices by a preset delay value, t_{AL}.

completed the data restoration phase of the row access command, as illustrated in Figure 11.10, the DRAM device will delay the implicit precharge command until the t_{RAS} timing requirement for the row access has been met. In this manner, close-page memory systems can issue the column-read-and-precharge command with best possible timing to retrieve data from the DRAM device without worrying about the precharge aspect of the random row access.

A second type of complex commands supported by some memory systems such as the DDR2 SDRAM memory system is the posted column access (posted CAS) command. The posted CAS command is simply a column access command whose action is delayed (or posted) by a fixed number of cycles in the DRAM device. In DRAM devices that support the posted CAS command, the device internally delays the actions of the CAS command by a preset value, labelled as t_{AL} in Figure 11.11. The number of delay cycles for the posted CAS command is preprogrammed into the DDR2 DRAM device, and the DRAM device cannot dynamically or intelligently defer the column access command. Some DRAM devices, such as the XDR DRAM device, allow the command to be optionally encoded with a delay value so that the controller can flexibly schedule a DRAM command that the DRAM device will then execute or act upon after the number of specified delay cycles.

Figure 11.11 illustrates a posted column-read-and-precharge command in a read cycle. The column-read-and-precharge command is also commonly referred to as the column-read command with auto-precharge. The sequence of commands illustrated in Figure 11.11 is the same as the sequence of DRAM commands illustrated in Figure 11.10. The difference between the command sequences illustrated in Figures 11.10 and 11.11 is that the column-read-and-precharge command is a posted CAS command, and the posted column-read command is issued immediately after the row access command in Figure 11.11. In Figure 11.11, the preset value of t_{AL} defers the action of the column-read-and-precharge command to ensure that the column read aspect of the command does not violate the t_{RCD} timing requirement.

The advantage of the posted CAS command is that it allows a memory controller to issue the column access command immediately after the row access command and greatly simplifies controller design for close-page memory systems. However, the posted CAS command is not a panacea that reduces controller complexity for all types of memory controllers, since the posted command must still respect the timing of the column access commands and normal protocol timing requirements.

11.2 DRAM Command Interactions

In the previous section, basic DRAM commands are described in some detail. In this section, the interactions between the basic DRAM commands are examined in further detail. In this chapter,

a resource usage model is used as the primary model to describe DRAM command interactions and the need for specification of the various timing parameters. In the resource usage model, DRAM commands can be scheduled consecutively subject to availability of shared on-chip resources such as sense amplifiers, I/O gating multiplexors, and the availability of off-chip resources such as the command, address, and data busses. However, even with the availability of shared resources, considerations such as device current limitations can prohibit commands from being scheduled consecutively.[4]

This section examines read and write commands in a memory system with simplistic open- and close-page row-buffer-management policies. In a memory system that implements the open-page row-buffer-management policy, once a row is opened for access, the array of sense amplifiers continues to hold the data for subsequent read and write accesses to the same row until another access to a different row within the same bank forces the controller to precharge the sense amplifiers and prepare for access to the different row. Open-page memory systems rely on workloads that access memory with some amount of spatial locality so that multiple column accesses can be issued to the same row without the need for multiple DRAM row cycles. In an open-page memory system, the DRAM command sequence for a given request depends on the state of the memory system, and the dynamic nature of DRAM command sequences in open-page memory systems means that there are larger numbers of possible DRAM command interactions and memory system state combinations in an open-page memory system. The large number of command interactions leads to a higher degree of difficulty in scheduling command sequences. In the following sections, a large number of possible DRAM command interactions for open-page memory systems are examined in detail. The detailed examination of DRAM command combinations enables the creation of a table that summarizes the minimum scheduling distances between DRAM commands. The summary of minimum scheduling distances, in turn, enables performance analysis of DRAM memory systems in this chapter.

11.2.1 Consecutive Reads and Writes to Same Rank

In modern DRAM memory systems such as SDRAM, DDR SDRAM, and Direct RDRAM memory systems, read commands to the same open row of memory in the same bank, rank, and channel can be pipelined and scheduled consecutively subject to the burst durations of data on the data bus and the internal prefetch length of the DRAM device. Figure 11.12 shows two column-read commands, labelled as read 0 and read 1, pipelined consecutively. As illustrated in Figure 11.4, t_{CAS} time after a column-read command is placed onto the command and address busses, the DRAM device begins to return data on the data bus. Since column-read commands to any open banks of the same rank can be pipelined consecutively, consecutive column-read commands to the same open row of the same bank of memory can be ideally scheduled every t_{BURST} time period. One caveat to the scheduling of consecutive column-read commands to an open row of a bank is that t_{BURST} has to be greater than or equal to t_{CCD}. Trivially, t_{BURST} is greater than t_{CCD} in all RDRAM, SDRAM, DDR SDRAM, and DDR2 SDRAM memory systems. The one exception to the rule is that t_{CCD} is 4 cycles in DDR3 SDRAM devices, since the prefetch length of DDR3 SDRAM devices is 8 data beats. Consequently, the best-case timing for consecutive column-read commands to the same or different banks of the same rank of DRAM devices is MAX(t_{BURST}, t_{CCD}).

Finally, similar to the case of consecutive column-read commands to the same bank of a given rank of memory, consecutive column-write commands can be scheduled to different open banks within the same rank of memory once every MAX(t_{BURST}, t_{CCD}) time period.

[4]That is t_{RRD} and t_{FAW}.

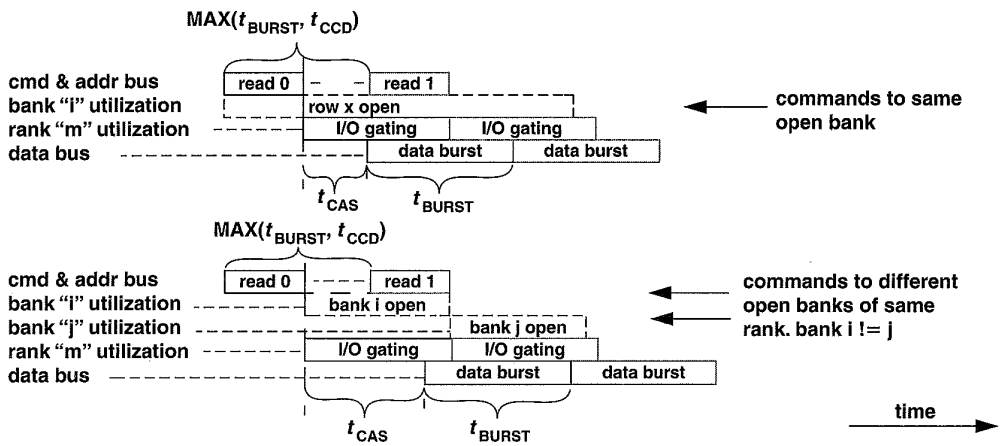


FIGURE 11.12: Consecutive column-read commands to the same bank, rank, and channel.

11.2.2 Read to Precharge Timing

Figure 11.13 illustrates the minimum command timing for a precharge command that immediately follows a column-read command. Figure 11.13 illustrates the formula for minimum command timing as $t_{BURST} + t_{RTP} - t_{CCD}$.[5] In the case where the internal burst length of the DRAM device t_{CCD} is equivalent to the column burst duration t_{BURST}, the minimum command timing between a column-read command and a precharge command is simply t_{RTP}. However, in some DRAM devices, the burst duration can be composed of multiple internal data bursts. For example, as previously illustrated in Figure 11.4, in a DDR2 SDRAM memory system where column access commands are programmed to move data in burst durations of 8 beats, the internal burst duration of a DDR2 SDRAM device is only 4 beats. As a result, the column read to precharge timing in the aforementioned DDR2 SDRAM memory system can be simply rewritten as $t_{RTP} + t_{CCD}$.

Essentially, the timing parameter t_{RTP} itself specifies the minimum amount of time that is needed between a column-read command and a precharge command. However, in DRAM devices such as the DDR2 SDRAM device, multiple shorter bursts are used to construct one continuous burst for a column-read command. In such a case, the DRAM device must keep the sense amplifiers open to drive multiple short bursts through the I/O gating multiplexors, and the timing parameter t_{RTP} must be modified to account to the extended time that the sense amplifiers are kept open. More generally, the formula for column read to precharge timing can be written as $t_{RTP} + (N - 1) * t_{CCD}$, where N is the number of internal bursts required to form one extended burst for a single column-read command.

11.2.3 Consecutive Reads to Different Rows of Same Bank

In most modern DRAM devices, multiple column-read commands to the same row of the same open bank can be issued and pipelined consecutively as illustrated in Figure 11.12. However, column-read

[5]Implicit in this formula is that the column-read command does not use posted CAS timing. Otherwise, t_{AL} must be added, and the formula rewritten as $t_{AL} + t_{BURST} + t_{RTP} - t_{CCD}$.

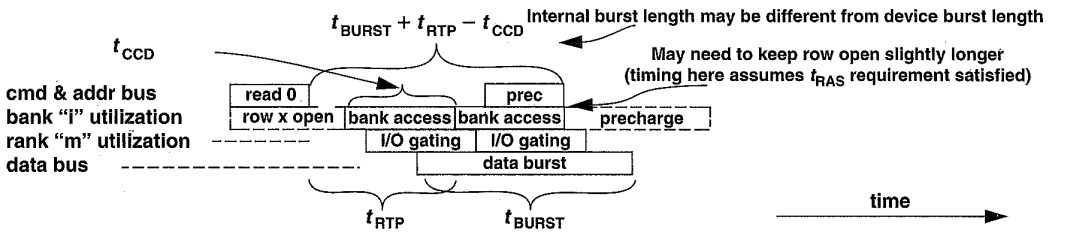


FIGURE 11.13: Read to precharge command timing.

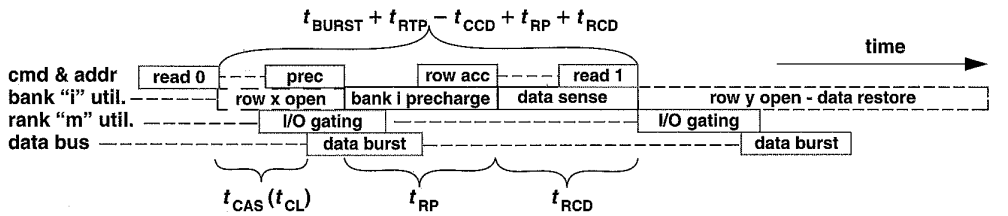


FIGURE 11.14: Consecutive column-read commands to different rows of the same bank: best-case scenario.

commands to different rows within the same bank would incur the cost of an entire row cycle time as the current DRAM array must be precharged and a different row activated by the array of sense amplifiers.

Best-Case Scenario

Figure 11.14 illustrates the timing and command sequence of two consecutive read requests to different rows within the same bank of memory array. In this sequence, as the first read command, labelled as read 0, is issued, the array of sense amplifiers must be precharged before a different row to the same bank can be opened for access. After time period t_{RP} from the assertion of the precharge command, a different row access command can then be issued, and time period t_{RCD} after the row access command, the second read command labelled as read 1 can then proceed. Figure 11.14 illustrates that consecutive column read accesses to different rows within the same bank can be scheduled with the best-case timing of $t_{BURST} + t_{RTP} - t_{CCD} + t_{RP} + t_{RCD}$ as long as the row restoration time t_{RAS} had been satisfied.

Worst-Case Scenario

Figure 11.14 illustrates the best-case timing of two consecutive read commands to different rows of the same bank. However, the timing illustrated in Figure 11.14 assumes that at least t_{RAS} time has passed since the previous row access, and data had been restored to the DRAM cells. In the case where

data from the current row has not yet been restored to the DRAM cells, a precharge command cannot be issued until t_{RAS} time period after the previous row access command to the same bank. In contrast to the best-case timing shown in Figure 11.14, Figure 11.15 shows the worst-case timing for two consecutive read commands to different rows of the same bank where the first column command was issued immediately after a row access command. In this case, the precharge command cannot be issued immediately after the first column-read command, but must wait until t_{RAS} time period after the previous row access command has elapsed. Then, t_{RP} time period after the precharge command, the second row access command can be issued, and t_{RCD} time period after that row access command, the second column-read command completes this sequence of commands.

Figure 11.14 illustrates the best-case timing of two consecutive read commands to different rows of the same bank, and Figure 11.15 illustrates the worst-case timing between two column-read commands to different rows of the same bank. The difference between the two different scenarios means that a DRAM memory controller must keep track of the timing of a row access command and delay any row precharge command until the row restoration requirement has been satisfied.

11.2.4 Consecutive Reads to Different Banks: Bank Conflict

The case of consecutive read commands to different rows of the same bank has been examined in the previous section. This section examines the case of consecutive read requests to different banks with the second request hitting a bank conflict against an active row in that bank. The consecutive read request scenario with the second read request hitting a bank conflict to a different bank has several different combinations of possible minimum scheduling distances that depend on the state of the bank as well as the capability of the DRAM controller to reorder commands between different transaction requests.

Without Command Reordering

Figure 11.16 illustrates the timing and command sequence of two consecutive read requests to different banks of the same rank, and the second read request is made to a row that is different than the active row in the array of sense amplifiers of that bank. Figure 11.16 makes three implicit assumptions. The first assumption made is that both banks i and j are open, where bank i is different from bank j. The second read request is made to bank j, but to a different row than the row of data presently held in the array of sense amplifiers of bank j. In this case, the precharge command to bank j can proceed concurrently with the column read access to a bank i. The second assumption made is that the t_{RAS} requirement has been satisfied in bank j, and bank j can be immediately precharged. The third and final assumption made is that the DRAM controller does not support command or transaction reordering between different transaction requests. That is, all of the DRAM commands associated with the first request must be scheduled before any DRAM commands associated with the second request can be scheduled.

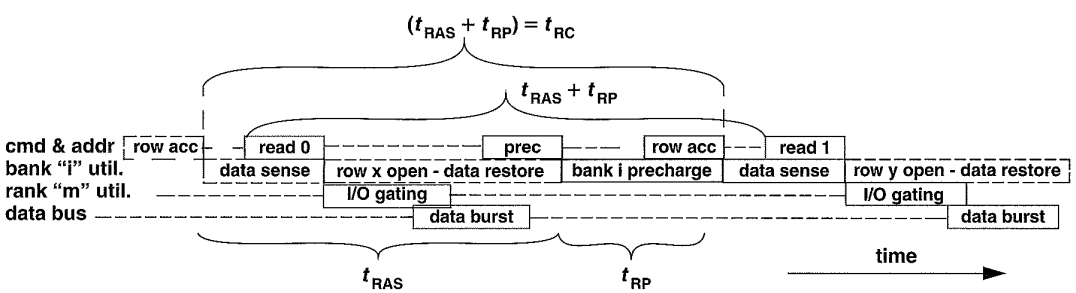


FIGURE 11.15: Consecutive column-read commands to different rows of same bank: worst-case scenario.

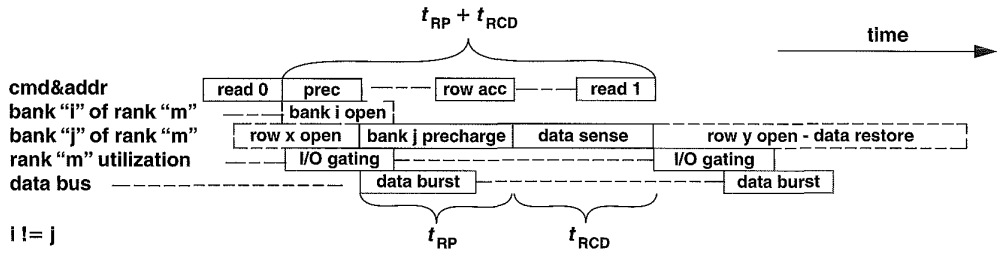


FIGURE 11.16: Consecutive DRAM read commands to different banks; bank conflict; no command reordering.

Figure 11.16 shows that due to the bank conflict, the read request to bank j is translated into a sequence of three DRAM commands. The first command in the sequence precharges the sense amplifiers to bank j, the second command brings the selected row to the sense amplifiers, and the last command in the sequence performs the actual read request and returns data from the DRAM devices to the DRAM controller. Figure 11.16 illustrates a case where consecutive read requests are made to different rows, with the second read request made to a different row of the same bank. In the case where the DRAM command sequence is not dynamically reordered by the memory controller, then the two requests can, at best, be scheduled with minimum timing distance of $t_{RP} + t_{RCD}$.

With Command Reordering

Figure 11.16 illustrates the timing of two requests to different banks with the second request hitting a bank conflict and the DRAM controller not supporting command or transaction reordering. In contrast, Figure 11.17 shows that the DRAM memory system can obtain bandwidth utilization if the DRAM controller can interleave or reorder DRAM commands from different transactions requests. Figure 11.17 shows the case where the DRAM controller allows the precharge command for bank j to proceed ahead of the column-read command for the transaction request to bank i. In this case, the column-read command to bank i can proceed in parallel with the precharge command to bank j, since these two commands utilize different resources in different banks. To obtain better utilization of the DRAM memory system, the DRAM controller must be designed with the capability to reorder and interleave commands from different transaction requests. Figure 11.17 shows that in the case where the DRAM memory system can interleave and reorder DRAM commands from different transaction requests, the two column-read commands can be scheduled with the timing of $t_{RP} + t_{RCD} - t_{CMD}$. Figure 11.17 thus illustrates one way that a DRAM memory system can obtain better bandwidth utilization with advanced DRAM controller designs.

11.2.5 Consecutive Read Requests to Different Ranks

Figure 11.12 illustrates that consecutive read commands to open banks of the same rank of DRAM device can be issued and pipelined consecutively.[6] However, consecutive read commands to different

[6]That is, assuming that the burst duration is equal to or longer than the minimum column-to-column delay time, t_{CCD}. This assumption is inherently true for SDRAM, DDR SDRAM, and DDR2 SDRAM devices, but may not be true for DDR3 and future generation devices where t_{BURST} may be shorter than t_{CCD}.

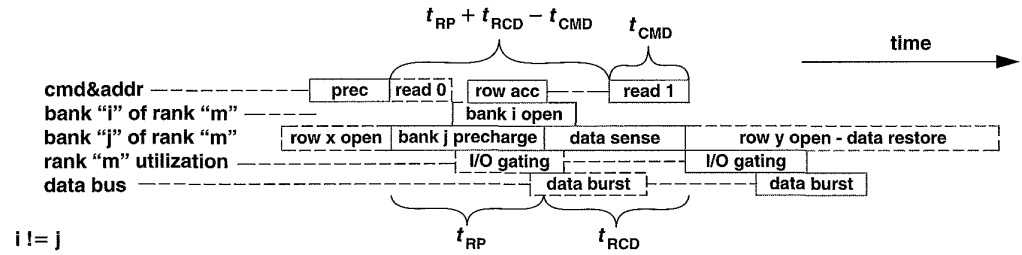


FIGURE 11.17: Consecutive DRAM read commands to different banks, bank conflict, with command reordering.

ranks of memory may not be issued and pipelined back to back depending on the system-level synchronization mechanism and the operating data rate of the memory system. In some memory systems, consecutive read commands to different ranks of memory rely on system-level synchronization mechanisms that are non-trivial for multi-rank, high data rate memory systems. In these systems, the data bus must idle for some period of time between data bursts from different ranks on the shared data bus. Figure 11.18 illustrates the timing and command sequence of two consecutive read commands to different ranks. In Figure 11.18, the read-write (DQS) data strobe resynchronization time is labelled as t_{RTRS}. For relatively low-frequency SDRAM memory systems, data synchronization strobes are not used, and t_{RTRS} is zero. For Direct RDRAM memory systems, the use of the topology-matched source-synchronous clocking scheme obviates the need for a separate strobe signal, and t_{RTRS} is also zero. However, for DDR SDRAM and DDR2 SDRAM memory systems, the use of a system-level data strobe signal shared by all of the ranks means that the t_{RTRS} data strobe resynchronization penalty is non-zero.

11.2.6 Consecutive Write Requests: Open Banks

Differing from the case of consecutive column-read commands to different ranks of DRAM devices, consecutive column-write commands to different ranks of DRAM devices may be pipelined consecutively in modern DRAM memory systems, depending on the bus termination strategy deployed. The difference between consecutive column-write commands to different ranks of DRAM devices and consecutive column-read commands to different ranks of DRAM devices is that in case of consecutive column-read commands to different ranks of DRAM devices, one rank of DRAM devices must first send data on the shared data bus and give up control of the shared data bus, then the other rank of DRAM devices must gain control of the shared data bus and send its data to the DRAM controller. In the case of the consecutive column-write commands to different ranks of memory, the DRAM memory controller can send data to different ranks of DRAM devices without needing to give up control of the shared data bus to another bus master. Consequently, write bursts to different ranks of DRAM devices can be pipelined consecutively in SDRAM and DDR SDRAM memory systems. However, as signaling on a multi-drop bus becomes more challenging with increasing data rates, DRAM device manufacturers and system design engineers have been forced to deploy more sophisticated mechanisms to improve system-level signal integrity. One mechanism deployed in DDR2 SDRAM memory systems to improve signal integrity is the use of active, ***On-Die Termination*** (ODT) on DDR2 SDRAM devices. However, one unfortunate side effect of ODT as designed in the DDR2 SDRAM memory system is that it takes 2 cycles to turn on ODT in a DDR2 SDRAM device and 2 1/2 cycles to turn off ODT, and the difference in

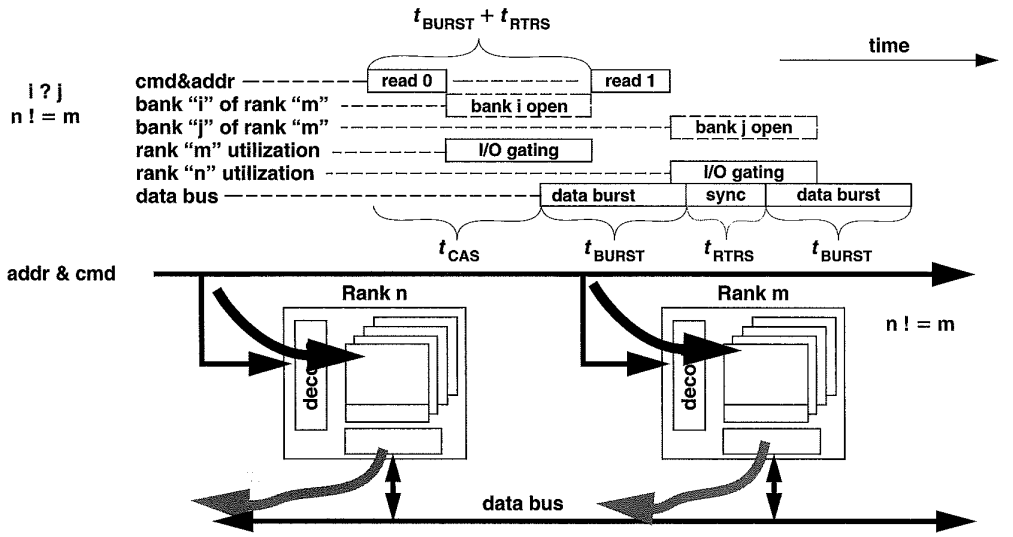


FIGURE 11.18: Consecutive column-read commands to different ranks.

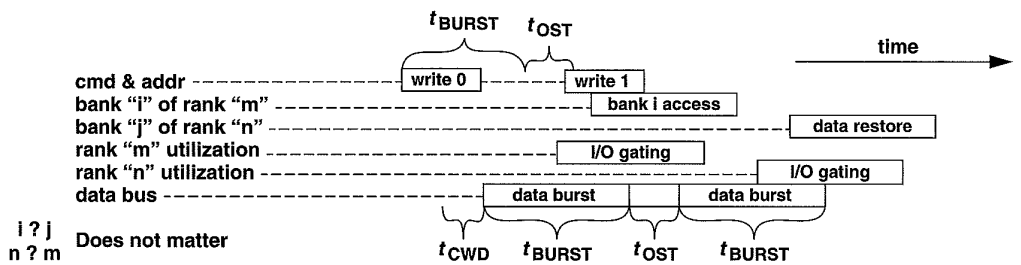


FIGURE 11.19: Consecutive column-write commands to different ranks.

turn-on and turn-off times necessitates an additional bubble between write bursts.[7] Figure 11.19 shows two write commands to different ranks, labelled as write 0 and write 1, pipelined consecutively, but with an ODT switching time penalty, labelled as t_{OST}, between different ranks. Figure 11.19 shows that consecutive column-write commands to open banks of memory can occur every $t_{BURST} + t_{OST}$ cycle. In the case of SDRAM and DDR SDRAM memory systems, write data bursts to different ranks can occur without needing any idle time on the data bus. In the case of DDR2 and DDR3 memory systems, the need to accurately control data bus signaling characteristics means that an additional cycle is needed to switch the location of the termination element in the memory channel. In reality, t_{OST} is needed in the case of a rank-to-rank read as well.

[7]The difference is needed to ensure a properly terminated system topology for the duration of data transfers.

However, since t_{RTRS} is typically greater than or equal to t_{OST}, MAX(t_{RTRS}, t_{OST}) may be simplified as t_{RTRS}.

11.2.7 Consecutive Write Requests: Bank Conflicts

Similar to the case of the consecutive read requests to different rows of the same bank, consecutive write requests to different rows of the same bank must also respect the timing requirements of t_{RAS} and t_{RP}. Additionally, column-write commands must also respect the timing requirements of the write recovery time t_{WR}. In the case of write commands to different rows of the same bank, the write recovery time means that the precharge cannot begin until the write recovery time has allowed data to move from the interface of the DRAM devices through the sense amplifiers into the DRAM cells. Figure 11.20 shows two of the best-case timing of two consecutive write requests made to different rows in the same bank. The minimum scheduling distance between two write commands to different rows of the same bank is $t_{CWD} + t_{BURST} + t_{WR} + t_{RP} + t_{RCD}$.

Figure 11.20 also shows the case where consecutive write requests are issued to different ranks of DRAM devices, with the second write request resulting in a bank conflict. In this case, the first write command proceeds, and assuming that bank j for rank n has previously satisfied the t_{RAS} timing requirement, the precharge command for a different bank or different rank can be issued immediately. Similar to the case of the consecutive read requests with bank conflicts to different banks, bank conflicts to different banks and different ranks for consecutive write requests can also benefit from command reordering.

11.2.8 Write Request Following Read Request: Open Banks

Similar to consecutive read commands and consecutive write commands, the combination of a column-write command that immediately follows a column-read command can be scheduled consecutively subject to the timing of the respective data bursts on the shared data bus. Figure 11.21 illustrates a column-write command that follows a column-read

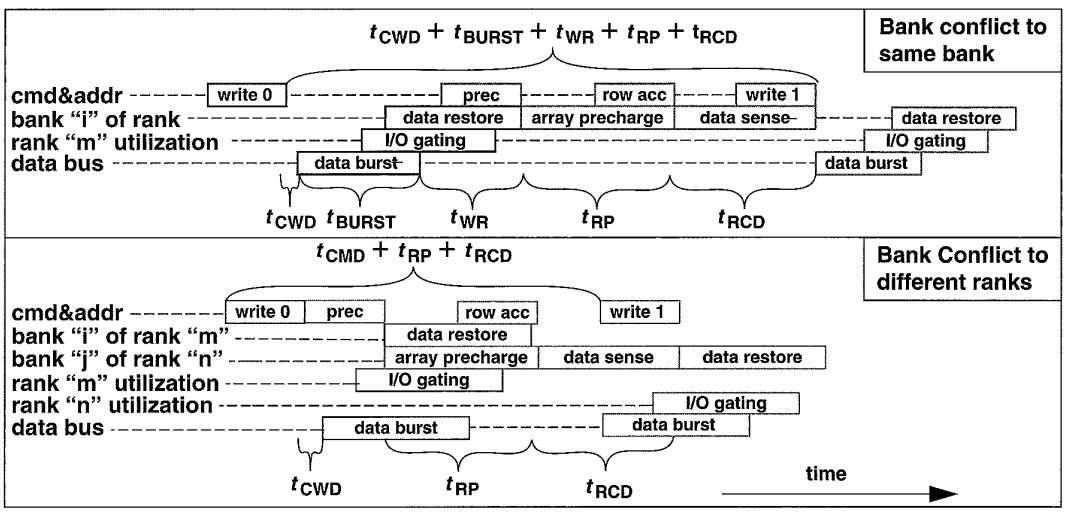


FIGURE 11.20: Consecutive write commands, bank conflict, best cases.

command and shows that the internal data movement of the column-write command does not conflict with the internal data movement of the column-read command. As a result, a column-write command can be issued into the DRAM memory system immediately after a column-read command as long as the timing of data burst returned by the DRAM device for the column-read command does not conflict with the timing of the data burst sent by the DRAM controller to the DRAM device for the column-write command. Figure 11.21 shows that the minimum scheduling distance between a column-write command that follows a column-read command is simply $t_{CAS} + t_{BURST} + t_{RTRS} - t_{CWD}$.

The minimum timing distance between a column-write command that follows a column-read command is different for different memory-access protocols. However, the minimum timing expression of $t_{CAS} + t_{BURST} + t_{RTRS} - t_{CWD}$ is valid for an SDRAM memory system as well as various DDRx SDRAM memory systems. For example, in an SDRAM memory system, t_{CWD} and t_{RTRS} are both zero, so the minimum timing distance between a write request that follows a read request in an SDRAM memory system is simply $t_{CAS} + t_{BURST}$. Comparatively, in a DDR SDRAM memory system, t_{CWD} and t_{RTRS} both require 2 beats (1 full memory clock cycle), and the minimum timing distance between a write request that follows a read request in a DDR SDRAM memory system is also $t_{CAS} + t_{BURST}$. In both cases, the timing is the same, but arrives at the same equation through different means.

11.2.9 Write Request Following Read Request to Different Banks, Bank Conflict, Best Case, No Reordering

Figure 11.22 illustrates the case where a write request follows a read request to different banks. In the figure, the column-read command is issued

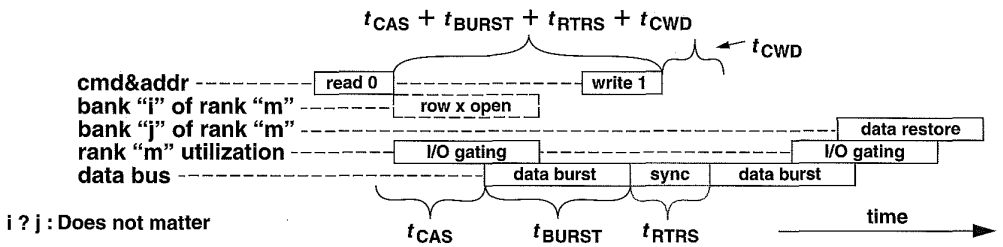


FIGURE 11.21: Write command following read command to open banks.

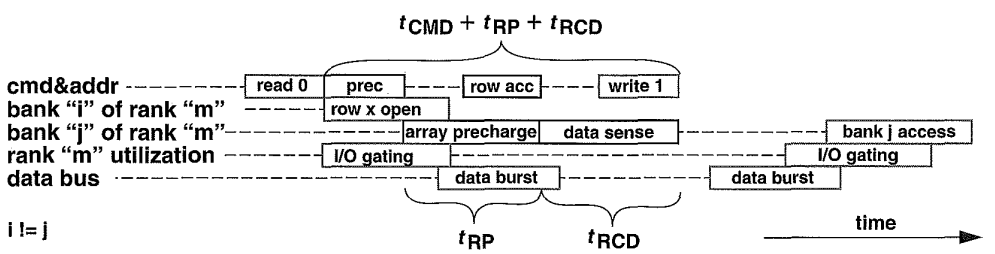


FIGURE 11.22: Write command following read command to different banks: bank conflict, best case.

to bank i, the column-write command is issued to bank j, and i is different from j. In the common case, the two commands can be pipelined consecutively with the minimum scheduling distance shown in Figure 11.21. However, the assumption given in Figure 11.22 is that the write command is a write command to a different row than the row currently held in bank j. As a result, the DRAM controller must first precharge bank j and issue a new row access command to bank j before the column-write command can be issued. In the best-case scenario presented, the row accessed by the write command in bank j had already been restored to the DRAM cells, and more than t_{RAS} time period had elapsed since the row was initially accessed. Figure 11.22 shows that under this condition, the read command and the write command that follows it to a different bank can be scheduled with the minimum scheduling distance of $t_{CMD} + t_{RP} + t_{RCD}$.

Figure 11.22 shows the case where the ordering between DRAM commands from different requests is strictly observed. In this case, the precharge command sent to bank j is not constrained by the column-read command to bank i. In a memory system with DRAM controllers that support command reordering and interleaving DRAM commands from different transaction requests, the efficiency of the DRAM memory system in scheduling a write request with a bank conflict that follows a read request can be increased in the same manner as illustrated for consecutive read requests in Figures 11.16 and 11.17.

11.2.10 Read Following Write to Same Rank, Open Banks

Figure 11.23 shows the case for a column-read command that follows a column-write command to open banks in the same rank of DRAM devices. The difference between a read command and a write command is that the direction of data flow within the selected DRAM devices is reversed with respect to each other. The importance in the direction of data flow can be observed when a read command is scheduled after a write command to the same rank of DRAM devices. Figure 11.23 shows that the difference in the direction of data flow limits the minimum scheduling distance between the column-write command and the column-read command that follows to the same rank of devices. Figure 11.23 shows that after the DRAM controller places the data onto the data bus, the DRAM device must make use of the shared I/O gating resource in the DRAM device to move the write data through the buffers into the proper columns of the selected bank. Since the I/O gating resource is shared between all banks within a rank of DRAM devices, the sharing of the I/O gating device means that a read command that follows a write command to the same rank of DRAM devices must wait until the write command has been completed before the read command can make use of the shared I/O gating resources regardless of the target or destination bank IDs of the respective column access commands. Figure 11.23 shows that the minimum scheduling distance between a write command and

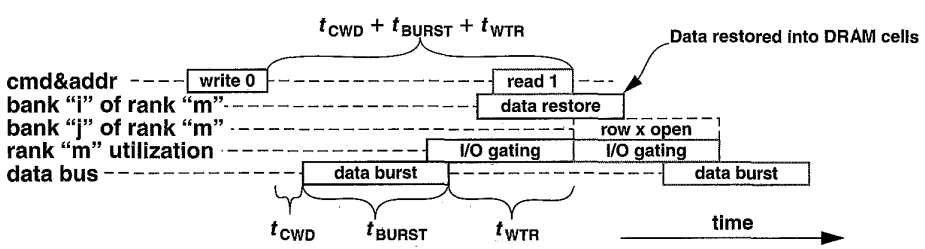


FIGURE 11.23: Read following write to the same rank of DRAM devices.

a subsequent read command to the same rank of memory is $t_{CWD} + t_{BURST} + t_{WTR}$.

In order to alleviate the write-to-read turnaround time illustrated in Figure 11.23, some high-performance DRAM devices such as Rambus Direct RDRAM have been designed with write buffers so that as soon as data has been written into the write buffers, the I/O gating resource can be used by another command such as a column-read command.

11.2.11 Write to Precharge Timing

Figure 11.24 shows the subtle difference between a column write to column read timing and a column write to precharge timing. Essentially, t_{WTR} is used to denote the time that is needed for the multiplexors in the interface of DRAM devices—the I/O gating phase illustrated in Figure 11.24—to drive the data into the array of sense amplifiers. After t_{WTR} time from the assertion of the column-write command, a column-read command is able to use the I/O gating resource to move data from an array of active sense amplifiers out of the DRAM device. The column write to precharge command combination differs from the column write to read combination in that the precharge command releases data stored in the sense amplifiers and precharges the sense amplifiers, while, as a result, a precharge command that follows a write command cannot be initiated until the write command has moved the data through the I/O gating resources to the sense amplifiers and driven the new data values directly into the DRAM cells. Figures 11.23 and 11.24 illustrate that in DRAM devices where t_{WTR} is separately specified from t_{WR}, t_{WTR} is typically shorter in duration than t_{WR}.

11.2.12 Read Following Write to Different Ranks, Open Banks

Figure 11.25 shows a slightly different case for a column-read command that follows a column-write command than the case illustrated in Figure 11.23. The combination of a column-read command

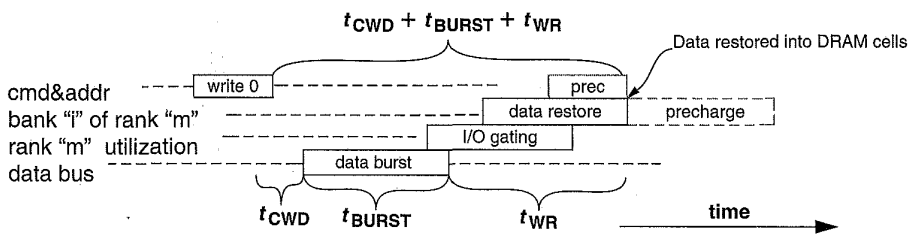


FIGURE 11.24: Write to precharge command timing.

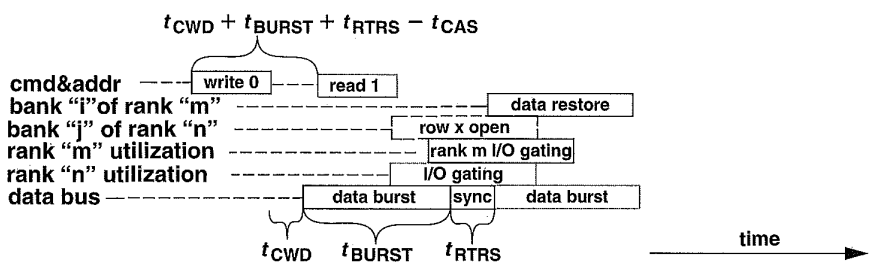


FIGURE 11.25: Read following write to different ranks of DRAM devices.

issued after a column-write command illustrated in Figure 11.25 differs from the combination of a column-read command issued after a column-write command illustrated in Figure 11.23 in that the column-write command and the column-read command are issued to different ranks of memory. Since the data movements occur in different ranks of memory, the conflict in the directions of data movement inside of each rank of memory is irrelevant. The timing constraint between the issuance of a read command after a write command to different ranks is then reduced to the data bus synchronization overhead of t_{RTRS}, the burst duration t_{BURST}, and the relative timing differences between read and write command latencies. The minimum time period between a write command and a read command to different ranks of memory is thus $t_{CWD} + t_{BURST} + t_{RTRS} - t_{CAS}$.

In a DDR SDRAM memory system, t_{CWD} is one cycle, t_{RTRS} is on clock cycle, and the minimum scheduling distance between a column-write command and a column-read command that follows it to a different rank of memory is MAX (t_{CMD}, $t_{BURST} - t_{CAS} - 2$). In contrast, in a DDR2 SDRAM memory system, t_{CWD} is one full cycle less than t_{CAS}, and if t_{RTRS} can be minimized to one full cycle, $t_{CWD} + t_{RTRS} - t_{CAS}$ would cancel to zero, and the minimum scheduling distance between a read command that follows a write command to a different rank in the DDR2 SDRAM memory system is simply t_{BURST}.

11.2.13 Read Following Write to Same Bank, Bank Conflict

Figure 11.26 illustrates the case where a read request follows a write request to different rows of the same bank. In the best-case scenario, the row accessed by the write request had already been restored to the DRAM cells, and more than t_{RAS} time period had elapsed since the previous row was initially accessed. Figure 11.26 shows that under this condition, the precharge command cannot be issued until the data from the column-write command has been written into the DRAM cells. That is, the write recovery time t_{WR} must be respected before the precharge command can proceed to precharge the DRAM array. Figure 11.26 shows that the best-case minimum scheduling distance between a read request that follows a write request to different rows of the same bank is $t_{CWD} + t_{BURST} + t_{WR} + t_{RP} + t_{RCD}$.

Figure 11.26 shows the command interaction of a read request that follows a write request to different rows of the same bank on a DRAM device that does not have a write buffer. In DRAM devices with write buffers, the data for the column-write command is temporarily stored in the write buffer. In case a read request arrives after a write request to retrieve data from a different row of the same bank, a separate commit data command may have to be issued by the DRAM controller to the DRAM devices to force the write buffer to commit the data stored in the write buffer into the DRAM cells before the array can be precharged for another row access.

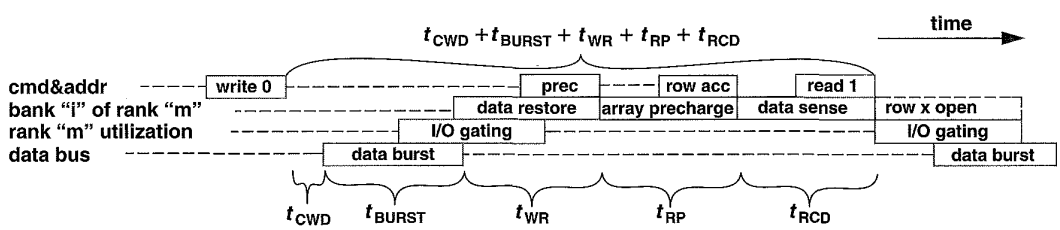


FIGURE 11.26: Read following write to different rows of the same bank.

11.2.14 Read Following Write to Different Banks of Same Rank, Bank Conflict, Best Case, No Reordering

Finally, Figure 11.27 illustrates the case where a read request follows a write request to different banks of the same rank of DRAM devices. However, the read request is sent to bank j, and a different row is active in bank j than the row needed by the read request. Figure 11.27 assumes that the t_{RAS} timing requirement has already been satisfied for bank j, and the DRAM memory system does not support DRAM command reordering between different memory transactions. Figure 11.27 shows that in this case, the precharge command for the read request command can be issued as soon as the write command is issued. Figure 11.27 thus shows that the minimum scheduling distance in this case is $t_{CMD} + t_{RP} + t_{RCD}$.

Figure 11.27 also reveals several points of note. One obvious point is that the DRAM command sequence illustrated in Figure 11.27 likely benefits from command reordering between different memory transactions. A second, less obvious point illustrated in Figure 11.27 is that the computed minimum scheduling distance depends on the relative duration of the various timing parameters. That is, Figure 11.27 assumes that the precharge command can be issued immediately after the write command and that $t_{CMD} + t_{RP} + t_{RCD}$ is greater than $t_{CWD} + t_{BURST} + t_{WR}$. In case $t_{CMD} + t_{RP} + t_{RCD}$ is, in fact, less than $t_{CWD} + t_{BURST} + t_{WR}$, the use of the shared I/O gating resource becomes the bottleneck, and the column-read command must wait until the write recovery phase of the column-write command has completed before the column-read command can proceed. That is, the minimum scheduling distance between a write request and a read request to a different bank with a bank conflict is in fact the larger of $t_{CMD} + t_{RP} + t_{RCD}$ and $t_{CWD} + t_{BURST} + t_{WR}$.

11.2.15 Column-Read-and-Precharge Command Timing

Figure 11.13 illustrates the minimum command timing for a precharge command that immediately follows a column-read command, and the minimum command timing was computed as $t_{BURST} + t_{RTP} - t_{CCD}$. Additionally, in the case where the column-read command is a posted column-read command, the additive latency parameter t_{AL} must be added to the overall command timing, resulting in the minimum command timing for a precharge command that follows a column-read command as $t_{AL} + t_{BURST} + t_{RTP} - t_{CCD}$. The column-read-and-precharge command would, in essence adopt the same timing specification and atomically perform an implicit precharge command $t_{AL} + t_{BURST} + t_{RTP} - t_{CCD}$ time after the column-read command is issued. However, the precharge component of the must still respect the row data restoration time t_{RAS} from the previous row activation command. Consequently, modern DRAM devices such as DDR2 SDRAMs have additional hardware that internally delays the precharge component of the unified column-read-and-precharge command to ensure that t_{RAS} timing to the row access is

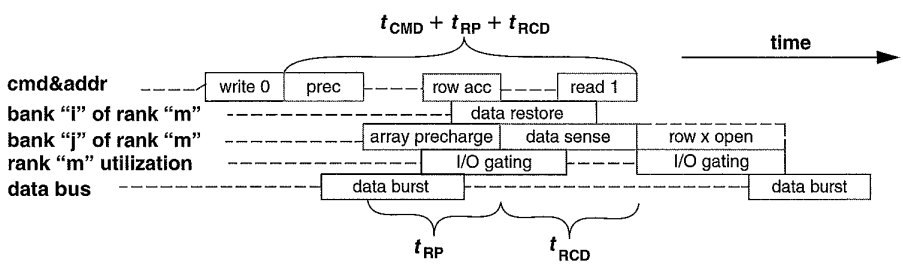


FIGURE 11.27: Read following write to different banks, bank conflict, best case.

respected before the integrated precharge component of the command is engaged. Figure 11.28 illustrates that the minimum timing of a column-read-and-precharge command that follows a row access command is simply $t_{RCD} - t_{AL}$.

11.2.16 Column-Write-and-Precharge Timing

The timing for a column-write-and-precharge command that immediately follows a row access command must ensure that both the column write component and the precharge component of the integrated column-write-and-precharge command respects the timing constraints imposed by the row access command. Fortunately, the timing parameter t_{RAS} has been defined to include a column-write command, and a column-write-and-precharge command can be freely issued after a row access command without violating precharge timing for the integrated precharge component of the command. Figure 11.29 shows the minimum timing required between a column-write command and a precharge command that follows it. Similar to the column-read-and-precharge command, the column-write-and-precharge command adopts the same timing specification and atomically performs an implicit precharge command.

Finally, DRAM devices are typically write-cycle limited, and t_{RAS} is often defined with enough timing margin for a single column-write command in SDRAM, and DDRx SDRAM memory systems. However, other memory systems and future memory systems may define t_{RAS} differently, and the timing of a column-write-and-precharge command may have additional constraints in such memory systems.

11.3 Additional Constraints

In previous sections, the resource contention model is used to justify various specified minimum timing constraints between different DRAM command combinations. However, additional constraints exist in modern DRAM memory systems and limit bandwidth utilization of modern DRAM memory systems. One such constraint is related to the power consumption of DRAM devices, and a second constraint is related to the distribution of commands and addresses to DRAM devices in the memory system. In the following sections, these additional constraints in the operations of the memory system are respectively examined.

11.3.1 Device Power Limit

With continuing emphasis placed on memory system performance, DRAM manufacturers are pushing for ever-higher data transfer rates in each successive generation of DRAM devices. However, just as increasing operating frequencies leads to higher activity rates and higher power consumption in modern processors, increasing data rates for DRAM devices also increases the potential for higher activity rates and higher power consumption on

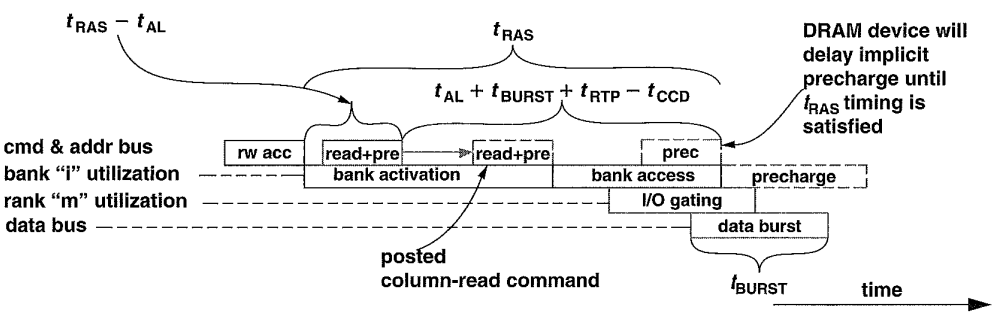


FIGURE 11.28: Row access to column-read-and-precharge command timing.

DRAM devices. One solution used by DRAM device design engineers to limit the power consumption of DRAM devices is to constrain the activity rate of DRAM devices. Constraints on the activity rate of DRAM devices, in turn, limit the capability of DRAM devices to move data and limit the performance of DRAM memory systems.

In modern DRAM devices, each time a row is activated, thousands of data bits are discharged, sensed, and then restored in parallel. As a result, the row activation command is a relatively energy intensive operation. Figure 11.30 shows the abstract current profile of a DRAM read cycle. In the figure, an active DRAM device draws a relatively low and constant quiescent current level as a result of active I/O buffers and DLLs. The DRAM device then draws additional current for each activity on the DRAM device. The total current draw of the DRAM device is simply the summation of the quiescent current draw and the current draw of each activity on the DRAM device. The current profiles shown in Figures 11.30 and 11.31 are described in terms of abstract units rather than concrete values. The reason that the current profiles are shown in abstract units in Figures 11.30 and 11.31 is that the magnitude of the current draw for the row activation command depends on the number of bits in a row that are activated in parallel, and the magnitude of the current draw for the data burst on the data bus depends on the data bus width of the DRAM device. As a result, the current profile of each command on each respective device depends not only on the type of the command, but also on the configuration of the DRAM device.

Modern DRAM devices contain multiple banks of DRAM arrays that can be pipelined to achieve high

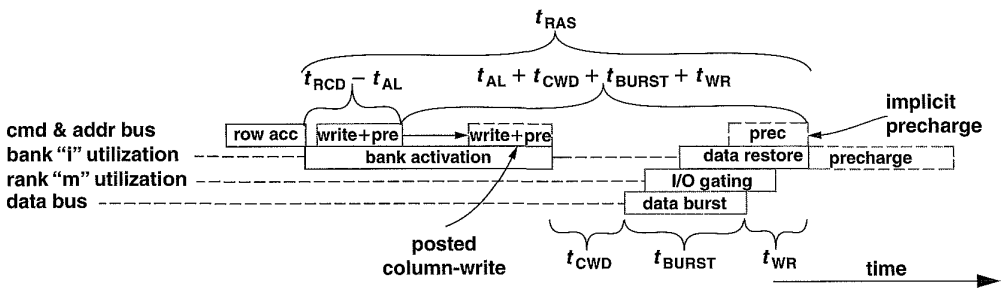


FIGURE 11.29: Row access to column-write-and-precharge command timing.

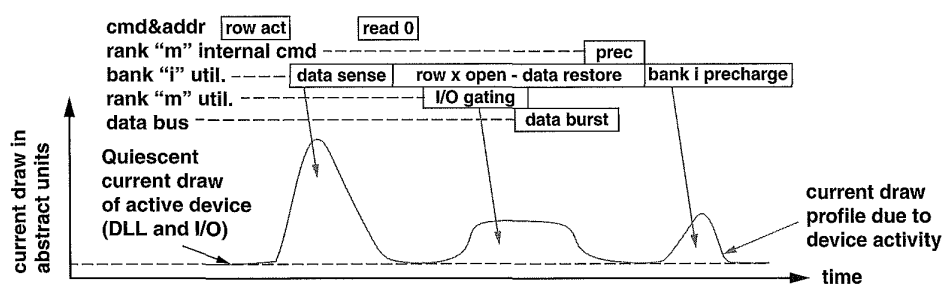


FIGURE 11.30: Current profile of a DRAM read cycle.

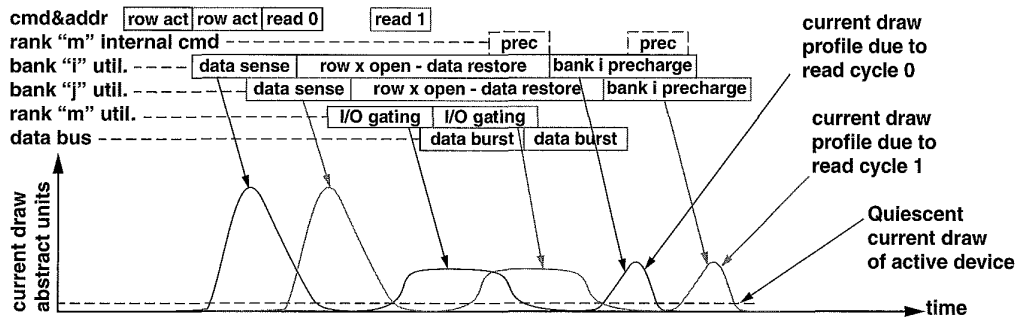


FIGURE 11.31: Current profile of two pipelined DRAM read cycles.

performance. Unfortunately, since the current profile of a DRAM device is proportional to its activity rate, a high-performance, highly pipelined DRAM device can also draw a large amount of active current. Figure 11.31 shows the individual contributions to the current profile of two pipelined DRAM read cycles on the same device. The total current profile of the pipelined DRAM device is not shown in Figure 11.31, but can be computed by the summation of the quiescent current profile and the current profiles of the two respective read cycles. The problem of power consumption for a high-performance DRAM device is that instead of only two pipelined read or write cycles, multiple read or write cycles can be pipelined, and as many as t_{RC}/t_{BURST} number of read or write cycles can be theoretically pipelined and in different phases in a single DRAM device. To limit the maximum current draw of a given DRAM device and avoid the addition of heat removal mechanisms such as head spreaders and heat sinks, new timing parameters have been defined in DDR2 and DDR3 devices to limit the activity rate and power consumption of DRAM devices.

11.3.2 t_{RRD}: Row-to-Row (Activation) Delay

In DDR2 SDRAM devices, the timing parameter t_{RRD} has been defined to specify the minimum time period between row activations on the same DRAM device. In the present context, the acronym RRD stands for **r**ow-to-**r**ow activation **d**elay. The timing parameter t_{RRD} is specified in terms of nanoseconds, and Table 11.3 shows that by specifying t_{RDD} in terms of nanoseconds instead of number of cycles, a minimum spacing between row activation is maintained regardless of operating data rates. For memory systems that implement the close-page row-buffer-management policy, t_{RRD} effectively limits the maximum sustainable bandwidth of a memory system with a single rank of memory. In memory systems with two or more ranks of memory, consecutive row activation commands can be directed to different ranks to avoid the t_{RRD} constraint.

Table 11.3 shows t_{RRD} for different configurations of a 1-Gbit DDR2 SDRAM device from Micron. The 1-Gbit DDR2 SDRAM device with the 16-bit-wide data bus is internally arranged as 8 banks of 8192 rows per bank and 16384 bits per row. Comparatively, the 512-Mbit ×4 and 256-Mbit ×8 configurations of the 1-Gbit DDR2 SDRAM device are arranged internally as 8 banks of 16384 rows per bank and 8192 bits per row. Table 11.3 thus demonstrates that with larger row size, each row activation on the 128-Mbit ×16, 1-Gbit DDR2 SDRAM device draws more current than a row activation on the 256-Mbit ×8 or 512-Mbit ×4, 1-Gbit DDR2 SDRAM devices. As a result, row activations in the 128-Mbit ×16, 1-Gbit DDR2

TABLE 11.3 t_{RRD} and t_{FAW} for a 1-Gbit DDR2 SDRAM device

Device Configuration	512 Mbit x 4	256 Mbit x 8	128 Mbit x 16
Data bus width	4	8	16
Number of banks	8	8	8
Number of rows	16384	16384	8192
Number of columns	2048	1024	1024
Row size (bits)	8192	8192	16384
t_{RRD} (ns)	7.5	7.5	10
t_{FAW} (ns)	37.5	37.5	50

SDRAM device must be spaced farther apart in time, as specified in Table 11.3.

11.3.3 t_{FAW}: Four-Bank Activation Window

The topic of DRAM device configuration was covered in some detail in a previous chapter. The coverage provided was lacking discussions in regard to the reasoning behind the trade-offs of bank count, row count, and row size. That is, the difficulties of determining the optimal bank count, row count, and column count (row size) were not discussed. However, with the description of t_{RRD} and t_{FAW}, the discussion can now proceed with some context. Basically, as DRAM devices grow in density, DRAM device configuration becomes a tug of war between some rather unpalatable choices. DRAM device manufacturers are reluctant to increase the number of banks, since bank control logic overhead increases die size. However, larger rows increase the current draw of the row activation command, necessitating longer t_{RRD} and t_{FAW} parameter specifications. Increasing the number of rows per bank means that there are more cells on a given bitline, contributing possibly to longer t_{RCD} and t_{RAS} parameters, not to mention that with more rows, more row refresh requests are needed per unit time.

To manage the ever-increasing capacity of DRAM devices, 1-Gbit and larger DDR2 SDRAM devices have been designed with 8 banks of DRAM arrays. However, the larger number of banks means that a 1-Gbit DDR2 SDRAM device can have more than 4 banks of DRAM arrays that are activated in rapid succession, and the current draw of the 8-bank device can exceed the current draw of a comparable device with only 4 banks of DRAM arrays. To limit the increase in current draw in the larger DDR2 SDRAM devices, the timing parameter t_{FAW} has been defined to specify a rolling time-frame in which a maximum of four row activations on the same DRAM device can be engaged concurrently. The acronym FAW stands for ***Four-bank Activation Window***. Figure 11.32 shows a sequence of row activation requests to different banks on the same DDR2 SDRAM device that respects both t_{RRD} as well as t_{FAW}. Figure 11.32 shows that the row activation requests are spaced at least t_{RRD} apart from each other and that the fifth row activation to a different bank is deferred until at least t_{FAW} time period has passed since the first row activation was initiated. For memory systems that implement the close-page row-buffer-management policy, t_{FAW} places great constraints on the maximum sustainable bandwidth of a memory system with a single rank of memory regardless of operating data rates. Finally, the timing parameters t_{RRD} and t_{FAW} have been defined for DDR2 SDRAM devices, and these timing parameters will carry over to DDR3 and future DDRx devices. Furthermore, these timing parameters are expected to increase in importance as future DRAM devices are introduced with even larger row sizes.

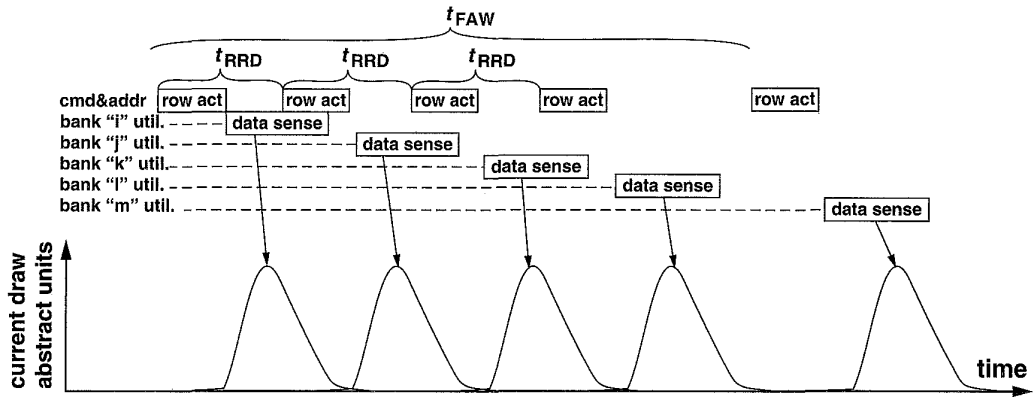


FIGURE 11.32: Maximum of four row activations in any t_{FAW} time frame.

11.3.4 2T Command Timing in Unbuffered Memory Systems

In high data rate, unbuffered memory systems, the address and command must be distributed to a large number of DRAM devices with the constraint of limited cycle times. To ensure functional correctness in the delivery of addresses and commands to the large number of DRAM devices in the memory system, the memory controller may be designed to hold the address and command for two full clock cycles. In this mode of operation, the memory controller is described as having 2T command timing. The net effect is that the command and address bandwidth of the memory system is reduced by one-half in these memory systems.

Figure 11.33 shows two memory systems populated with 4 ranks of DRAM devices. The memory system on the left has a single set of address and command busses connected to all four ranks of memory in the system. In contrast, the memory system on the right has two sets of address and command busses, with one set of busses devoted to each half of the memory system. Figure 11.33 shows that as a consequence of the number of loads on the address and command busses on the single set of address and command busses, the memory system on the left may be forced to use 2T command timing, while the memory system on the right may be able to retain the use of 1T command timing.

11.4 Command Timing Summary

In previous sections, the minimum timing between different combinations of DRAM commands was examined in detail. Table 11.4 summarizes the minimum timing equations for basic DRAM command interactions between the row access, column read, column write, column-read-and-precharge, column-write-and-precharge, precharge, and refresh commands. Table 11.4 is organized by each DRAM command, the possible commands that can precede each command, and the respective minimum timing constraints that must be met in between each command combination before the second command can be issued.

11.5 Summary

In this chapter, a basic DRAM-access protocol is described in detail. The basic DRAM-access protocol

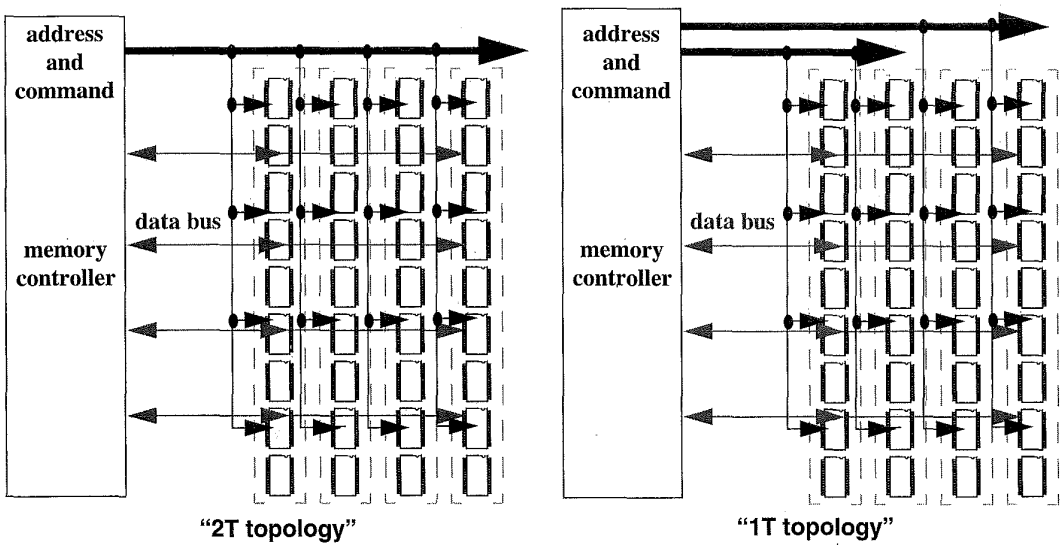


FIGURE 11.33: Unbuffered memory systems with 4 ranks of DRAM devices with one set of address and command busses and two sets of address and command busses.

is illustrated through the use of a resource utilization model. That is, two DRAM commands can be pipelined consecutively if they do not require the use of a shared resource at the same instance in time. The basic resource usage model is then qualified by stating that additional constraints such as power consumption limitation can further limit the issue rate of DRAM commands.

One popular question that is often asked, but not addressed by the description of the generic DRAM-access protocol, is in regard to what happens if the timing parameters are not fully respected. For example, what happens when a precharge command is issued before the t_{RAS} data restoration timing parameter has been fully satisfied? Does the DRAM device contain enough intelligence to delay the precharge command until t_{RAS} has been satisfied?

The answer to questions such as these is that, in general, DRAM devices contain very little intelligence in terms of logic circuitry to ensure functional correctness. The DRAM device manufacturers provide data sheets to specify the minimum timing constraints for individual DRAM commands. To ensure that the DRAM devices operate correctly, the DRAM controller must respect the minimum and maximum timing parameters as defined in the data sheet. In the specific case of a precharge command issued to a DRAM device before t_{RAS} has been satisfied, the DRAM device may still operate correctly, since the DRAM cell *may* have already been largely restored by the time the precharge command was engaged. The issue of early command issuance is thus analogous to that of the practice of processor overclocking. That is, a processor manufacturer can

TABLE 11.4 Summary of minimum DRAM command timing equations

Prev	Next	Rank	Bank	Minimum Timing	Illustration	Notes
A	A	s	s	t_{RC}	Figure 11.8	
A	A	s	d	t_{RRD}	Figure 11.32	Plus t_{FAW} for 5th RAS to same rank
P	A	s	d	t_{RP}	Figure 11.6	
F	A	s	s	t_{RFC}	Figure 11.7	
A	R	s	s	$t_{RCD} - t_{AL}$	Figure 11.11	$t_{AL} = 0$ without posted-CAS command
R	R	s	a	MAX(t_{BURST}, t_{CCD})	Figure 11.12	t_{BURST} is burst of prev. CAS to same rank
R	R	d	a	$t_{BURST} + t_{RTRS}$	Figure 11.18	t_{BURST} is burst of prev. CAS to diff rank
W	R	s	a	$t_{CWD} + t_{BURST} + t_{WTR}$	Figure 11.23	t_{BURST} is burst of prev. CASW to same rank
W	R	d	a	$t_{CWD} + t_{BURST} + t_{RTRS} - t_{CAS}$	Figure 11.25	t_{BURST} is burst of prev. CASW to diff rank
A	W	s	s	$t_{RCD} - t_{AL}$	Figure 11.11	
R	W	a	a	$t_{CAS} + t_{BURST} + t_{RTRS} - t_{CWD}$	Figure 11.21	t_{BURST} is burst of prev. CAS to any rank
W	W	s	a	MAX(t_{BURST}, t_{CCD})	Figure 11.19	t_{BURST} is burst of prev. CASW to same rank
W	W	d	a	$t_{BURST} + t_{OST}$	Figure 11.21	t_{BURST} is burst of prev. CASW to diff. rank
A	P	s	s	t_{RAS}	Figure 11.8	
R	P	s	s	$t_{AL} + t_{BURST} + t_{RTP} - t_{CCD}$	Figure 11.13	t_{BURST} is burst of prev. CAS to same rank
W	P	s	s	$t_{AL} + t_{CWD} + t_{BURST} + t_{WR}$	Figure 11.24	t_{BURST} is burst of prev. CASW to same rank
F	F	s	a	t_{RFC}	Figure 11.7	
P	F	s	a	t_{RP}	Figure 11.6	

A = row Access; R = column Read; W = column Write; P = Precharge; F = reFresh; s = same; d = different; a = any.

specify that a processor will operate correctly within a given frequency and supply voltage range. An end-user may increase the supply voltage and operating frequency of the processor in hopes of obtaining better performance from the processor. In such a case, the processor may well operate correctly. However, the parameters of operation are then outside of the bounds specified by the processor manufacturer, and the correctness of operation is no longer guaranteed by the processor manufacturer. Similarly, in the case where a DRAM command is issued at a timing that is more aggressive than that specified by the DRAM device data sheet, the DRAM device may still operate correctly, but the correctness of operation is no longer guaranteed by the DRAM device or module manufacturer.

CHAPTER 12

Evolutionary Developments of DRAM Device Architecture

The first Dynamic Random-Access Memory (DRAM) device, based on the operation of Field Effect Transistors (FET), was invented by Robert Dennard of IBM in 1966. Then in 1970, Intel Corp. produced the first commercial DRAM device, the 1103. In the decades since the introduction of Intel's 1103 DRAM device, DRAM device architecture has undergone continuous and gradual evolution. However, up until the mid-1990s, the evolutionary path of DRAM device architecture had been dominated by a single path devoted to high-capacity, low-cost commodity devices.

In recent years, the differing requirements placed on modern DRAM memory systems have strained the evolutionary development of modern DRAM devices. Consequently, the largely monolithic path of DRAM device architecture evolution has fractured into multiple evolutionary paths in recent years to respectively satisfy the varying system requirements of low cost, high bandwidth, low latency, and low power.

In previous chapters, basic structures of DRAM devices, common system configurations, abstract timing parameters, and abstract access protocols were described in detail. These chapters provide the foundation that enables the discussion in the rationale and capability of different DRAM devices. This chapter is devoted to the description and architectural evolution of different families of stand-alone DRAM devices.

12.1 DRAM Device Families

In the decades since the invention of the basic charge-storage DRAM circuit and the introduction of the first DRAM device, a myriad of DRAM devices, each with slightly different device architectures, has been developed to meet specific system requirements of low cost, low latency, high bandwidth, low power, or a reasonable combination thereof. In this chapter, the role of different DRAM devices is examined in context by classifying the different paths of the DRAM family tree illustrated in Figure 12.1.

12.1.1 Cost (Capacity), Latency, Bandwidth, and Power

The utility and cost effectiveness of commodity DRAM devices have allowed them to proliferate and become critical design components in large varieties of computer systems and computing devices. However, the specific requirements of different computer systems and computing devices have, in turn, forced DRAM device architectures to evolve and meet the wide spectrum of system requirements. Consequently, large numbers of DRAM devices that vary widely in their respective architecture, feature set, and performance characteristics now exist and occupy different market niches. To begin the difficult task of describing and comparing different DRAM

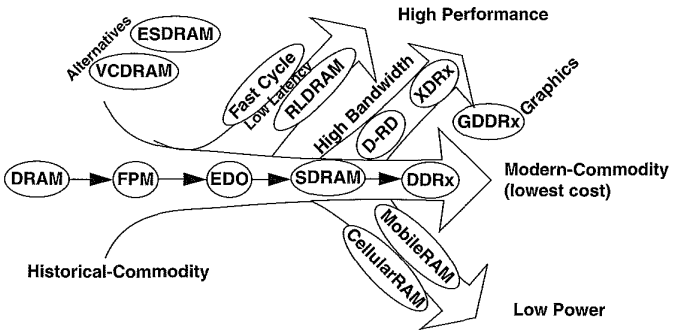


FIGURE 12.1: A broad classification of the DRAM family tree.

devices and different device architectures, four simple metrics are used in this chapter to broadly evaluate all DRAM devices and classify them into one of the four general paths of the family of DRAM devices described in this section. The four metrics are cost (capacity), latency, bandwidth, and power.

As described previously, different DRAM devices have been designed and manufactured to target low cost, low latency, high bandwidth, low power, or a reasonable compromise thereof. However, the single overriding consideration that has greatly impacted and constrained the evolutionary developments of DRAM device architectures is the issue of cost. In the general sense, cost-focused DRAM devices strive for the lowest cost per bit, and in this chapter, the focus on the lowest cost per bit is equated to the focus on capacity. In the decades since the invention of the DRAM circuit, the vast majority of DRAM devices manufactured and sold are commodity DRAM devices focused on cost, and these DRAM devices are classified in this chapter as belonging in the *Commodity* path of the DRAM device family tree. The classification of the DRAM family tree in this chapter follows the evolutionary family tree illustrated in Figure 12.1. Figure 12.1 separates and classifies the myriad of DRAM devices into four evolutionary paths in the DRAM family tree: the *Commodity* path, the *Low-Latency* path, the *High-Bandwidth* path, and the *Low-Power* path. To facilitate the examination of the numerous DRAM devices in the *Commodity* path, the *Commodity* path is further separated into *Historical-Commodity* and *Modern-Commodity* subpaths. Alternatively, DRAM devices that focus on providing fast random accesses are classified as belonging to the *Low-Latency* path of the DRAM device family tree. DRAM devices that focus on delivering the highest bandwidth are classified as belonging to the *High-Bandwidth* path. Finally, the near-commodity *Graphics-Oriented* GDDR devices and *Alternative* DRAM devices that had appeared along the way in the evolution of the DRAM family tree—yet do not readily belong in the primary paths of the DRAM family tree—are also examined in Sections 12.5 and 12.6.

12.2 Historical-Commodity DRAM Devices

As stated previously, the overriding factor that has driven and continues to drive the development of DRAM devices is the cost per storage bit. As a consequence of the singular focus on cost, nearly all legacy DRAM devices that existed prior to the emergence of *Synchronous* DRAM (SDRAM) devices are classified in this chapter as belonging in the *Historical-Commodity* path of the DRAM family tree. Specifically, the commodity DRAM device evolved from *asynchronous* to *fast page mode* (FPM) to *extended data-out* (EDO) to *burst-mode EDO* (BEDO) and then finally into *synchronous* (SDRAM). In this chapter, SDRAM and

its direct descendants such as DDR SDRAM, DDR2 SDRAM, and DDR3 SDRAM devices are classified as belonging in the *Modern-Commodity* path of the DRAM family tree, and these devices are examined in Section 12.3.

In this section, DRAM devices that belong to the *Historical-Commodity* path of the DRAM family tree are selectively examined, and the selected devices are the Intel 1103 DRAM device, the asynchronous DRAM device, the FPM DRAM device, the EDO DRAM device, and the BEDO DRAM device. These selectively examined devices represent important milestones in the evolutionary path of historical-commodity DRAM devices. However, a large number of different DRAM devices were produced by various DRAM manufacturers in the time period between the appearance of Intel's 1103 DRAM device and the BEDO DRAM device. Consequently, it is important to note that while the DRAM devices selectively examined in this section illustrate well the gradual evolution of commodity DRAM devices from the early 1970s to the late 1990s, the few illustrated devices do not represent all of the DRAM devices and evolutionary attempts in that time period.

In the evolution of DRAM device architecture from Intel's 1103 DRAM device to the BEDO DRAM device, the changes to the DRAM device have largely been structural in nature and relatively minor in terms of implementation cost; yet device functionality and throughput have increased significantly in each successive generation of DRAM devices. In the following section, the general description of the *Historical-Commodity* path of the DRAM family tree begins with a cursory examination of Intel's 1103 DRAM device.

12.2.1 The Intel 1103

The first commercially successful DRAM device, the 1103, was introduced by Intel in 1971. Due to its relatively low cost, the 1103 gained wide acceptance and rapidly replaced magnetic core memory in many different applications. Figure 12.3 shows the package pin configuration of the 1103, and it shows that this device was packaged in a low-cost, 18-pin, plastic dual in-line package. The 1103 had a row cycle time of 580 ns and a random access time of 300 ns. The 1103 also had a data retention time that was guaranteed to be longer than 2 ms; as long as each cell was refreshed at least once every 2 ms, data integrity was guaranteed. Differing from DRAM devices that followed it in subsequent years, the 1103 also had some interesting features such as unidirectional data input and output. The unidirectional input and output gave the early DRAM devices such as the 1103 (up to FPM devices) an interesting capability—an atomic read-modify-write cycle. That is, as the DRAM device drives data out of the output, the memory controller can hold the address on the address bus. Then, the controller can read, process, and immediately write data back through the input to the same address—something not practical in later DRAM devices with bidirectional data busses. Also, the 1103 used dedicated pins for row and column addresses, while later devices typically multiplexed row and column addresses over a single set of address pins. Moreover, the 1103 was a rudimentary device in terms of power supply requirements. Unlike the more modern DRAM devices, the 1103 required separate power supplies to properly bias the transistors. Finally, one feature that differentiated the 1103 from later DRAM devices was that it utilized the 3 transistor, 1 capacitor (3T1C) structure as the basic storage cell. Due to the 3T1C cell structure in the 1103 illustrated in Figure 12.2, data readout from the 1103 is nondestructive. Consequently, data does not need to be restored into the DRAM cell immediately after it is read out.

In comparison to the myriad of DRAM devices that followed it, Intel's 1103 DRAM device was a

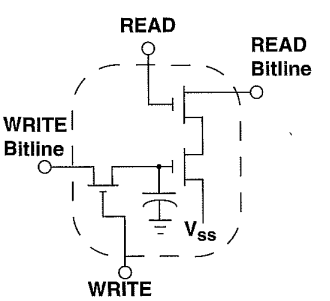


FIGURE 12.2: One (3T1C) bit cell of Intel's 1103 DRAM device.

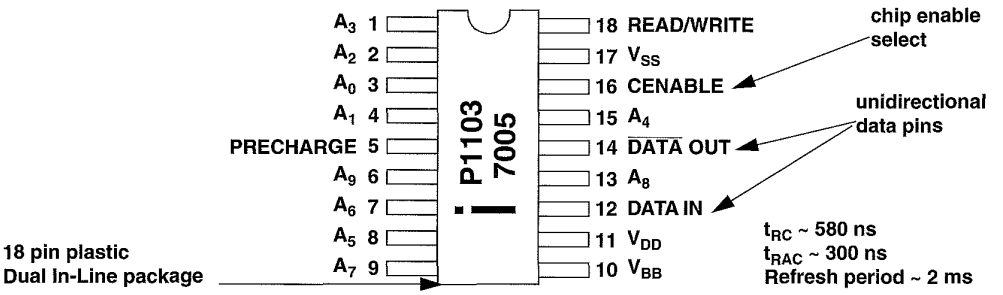


FIGURE 12.3: Pin configuration of Intel's 1103 DRAM device.

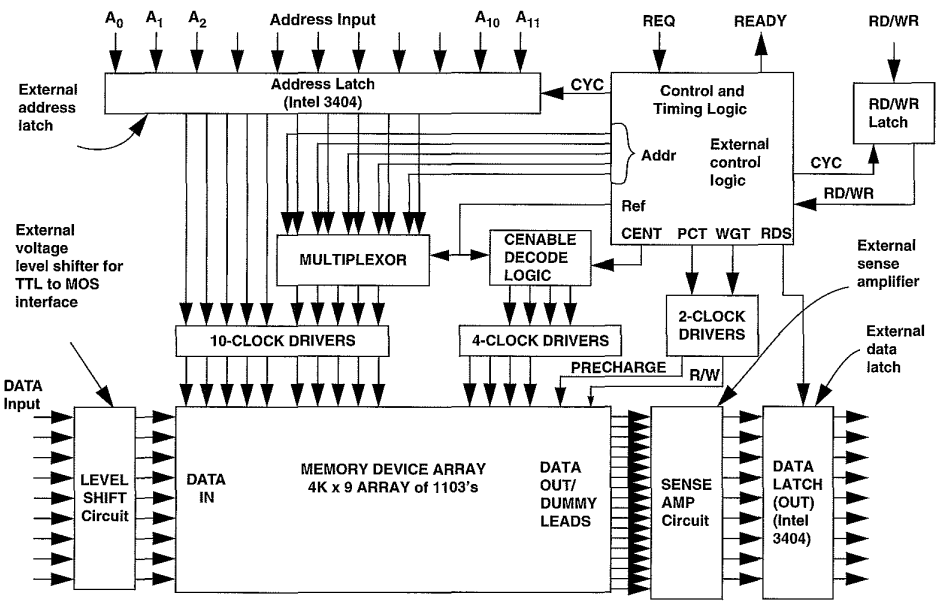


FIGURE 12.4: Structural block diagram of an 1103 memory system, consisting of 9 1103 devices.

rudimentary device that contained little more than address decoders and an array of 1024 DRAM cells. Common structures that were found in later devices, such as integrated address latches and sense amplifiers, were not present in the 1103. Consequently, a memory system that utilized the 1103 required more external circuitry than later devices. Figure 12.4 illustrates the structural block diagram of a memory system that utilizes the 1103.[1] The figure shows that several important external circuits are needed to

[1]Memory system block diagram taken from 1103 data sheet.

create a memory system based on the 1103. First, external voltage level shifters were needed to interface TTL devices to the MOS signal levels of the 1103. Second, the 1103 memory system required a significant amount of external logic to perform a simple read or write. Finally, the memory system used external sense amplifiers to resolve the stored data value from an 1103 DRAM device.

12.2.2 Asynchronous DRAM Devices

In the 1970s, subsequent to the introduction of Intel's 1103 DRAM device, many other DRAM devices were produced by Intel and other DRAM manufacturers. These DRAM devices were not standardized, and each DRAM device was unique in some way, containing subtle improvements from the previous generation of DRAM devices. For example, subsequent to the 1103, Intel introduced the 2104 DRAM device that multiplexed the row address and the column address over the same address bus, the 2116 DRAM device that was Intel's first 16-kbit DRAM device, and the 2118 DRAM device that was the first DRAM device to use a single 5-V power supply.

Other early DRAMs were sometimes clocked, where the DRAM commands were driven by a periodic clock signal. However, by the mid-1970s, DRAM device interfaces moved away from the clock-driven interface and moved to an *Asynchronous* command–data timing interface. These *Asynchronous* DRAM devices, like the clocked versions before them, require that every single access go through separate row activation and column access steps. Even if the microprocessor wants to request data contained in the same row that it previously requested, the entire row activation and column access process must be repeated. Figure 12.5 illustrates the timing for an asynchronous DRAM device. The figure shows that in the early asynchronous DRAM device, a row-activation command and a column-access command are needed to read data from the DRAM device, and two entire row cycles are needed to move data to or from any two addresses.

12.2.3 Page Mode and Fast Page Mode DRAM (FPM DRAM)

Since the introduction of Intel's 1103 DRAM device, each successive DRAM device has introduced new features to improve upon previous DRAM devices. One important feature that greatly improved DRAM device performance and was found in all later DRAM devices is *page mode* operation. In page mode operation, an entire row (page) of data is held by an array of sense amplifiers integrated into the DRAM device, and multiple column accesses to data in the same row can occur without having to suffer the latency of another row access. Page mode operation in DRAM devices improves system performance by taking advantage of the spatial locality present in the memory-access patterns of typical application

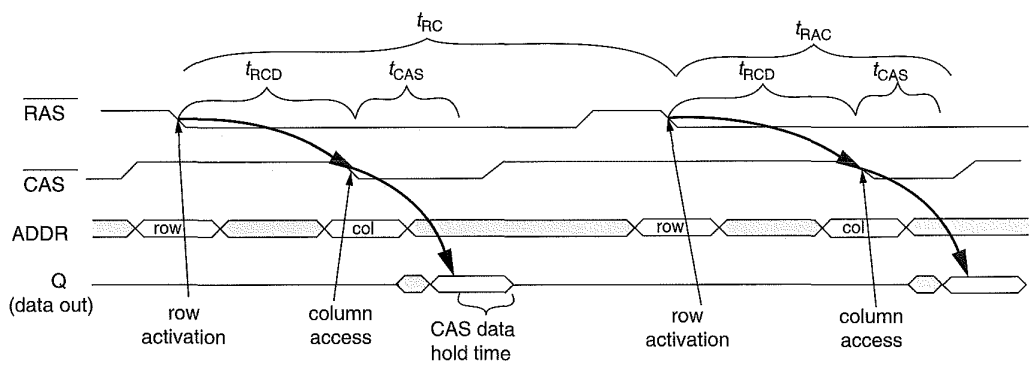


FIGURE 12.5: Read timing for a conventional asynchronous DRAM device.

programs. Page mode operation was not possible in the earliest DRAM devices due to the fact that the sense amplifiers were located off-chip. However, this mode of operation was enabled by the integration of sense amplifiers into the DRAM device, and data for an entire row could be sensed and buffered by the array of integrated sense amplifiers in parallel. Page mode permitted fast column accesses, and FPM improved the timing of the column access by allowing the column address buffers to remain open and active for as long as the row address strobe signal was held low. In this manner, the column addresses could be sent through the column address buffer and decoded before the column address strobe signal was asserted by the memory controller. Consequently, FPM was introduced into many 64-kbit DRAM devices in the early 1980s. FPM DRAM remained as the mainstream commodity memory well into the 1990s until it was replaced, in part, by EDO DRAM and finally entirely by *Synchronous DRAM* (SDRAM) in the late 1990s.

Figure 12.6 illustrates the organization and structure of a 64-Mbit FPM DRAM device with a 16-bit-wide bidirectional data bus. The FPM DRAM device illustrated in Figure 12.6 is a relatively modern DRAM device with a bidirectional data bus, while early generations of FPM DRAM devices typically had separate and unidirectional data input and output pins. Internally, the DRAM storage cells in the FPM DRAM device in Figure 12.6 are organized as 4096 rows, 1024 columns per row, and 16 bits of data per column. In this device, each time a row access occurs, the row address is placed on the address bus, and the *row address strobe* (RAS) is asserted by an external memory controller. Inside the DRAM device, the address on the address bus is buffered by the row address buffer and then sent to the row decoder. The row address decoder then accepts the row address and selects 1 of 4096 rows of storage cells. The data values contained in the selected row of storage cells are then sensed and kept active by the array of sense amplifiers. Each row of DRAM cells in the illustrated FPM DRAM device consists of 1024 columns, and each column is 16 bits wide. That is, a 16-bit-wide column is the basic addressable unit of memory in this device, and each column access that follows the row access would read or write 16 bits of data from the same

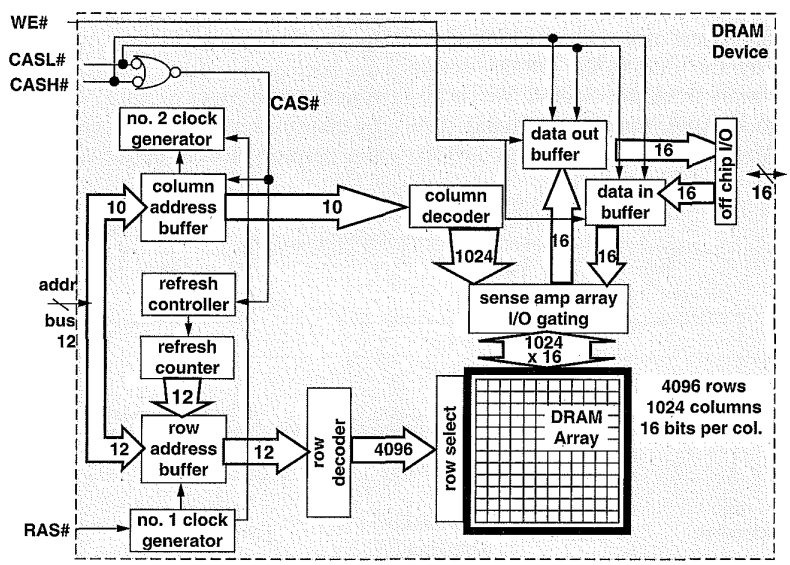


FIGURE 12.6: Internal organization of a 64-Mbit past page mode DRAM device (4096 x 1024 x 16).

row of DRAM. The FPM DRAM device illustrated in Figure 12.6 does allow each 8-bit half of the 16-bit column to be accessed independently through the use of separate *column access strobe high* (CASH) and *column access strobe low* (CASL) signals. In FPM DRAM devices, column access commands are handled in a similar manner as the row access commands. For a column access command, the memory controller places the column address on the address bus and then asserts the appropriate CAS signals. Internally, the DRAM chip takes the column address, decodes it, and selects 1 column out of 1024 columns. The data for that column is then placed onto the data bus by the DRAM device in the case of an ordinary column read command or overwritten with data from the memory controller depending on the write enable (WE) signal.

Figure 12.7 illustrates the timing for page mode read commands in an FPM device. The figure shows that once a row is activated by holding the $\overline{RAS}$ signal low, multiple column accesses can be used to retrieve spatially adjacent data from the same DRAM row with the timing of t_{PC}, the page mode cycle time. Figure 12.7 also shows that the output data valid time period is implicitly controlled by the timing of the $\overline{CAS}$ signal. That is, the DRAM device will hold the value of the read data out for some period of time after $\overline{CAS}$ goes high. The DRAM device then prepares for a subsequent column-access command without an intervening precharge command or row-access command.

In FPM DRAM devices, the page mode cycle time can be as short as one-third of the row cycle time. Consequently, (fast) page mode operation increased DRAM device bandwidth for spatially adjacent data by as much as three times that of a comparable generation asynchronous DRAM device without page mode operation.

12.2.4 Extended Data-Out (EDO) and Burst Extended Data-Out (BEDO) Devices

FPM DRAM devices were highly popular and proliferated into many different applications in the 1980s and the first half of the 1990s. Then, in the mid-1990s, EDO, a new type of DRAM device, was introduced into the market and achieved some degree of success in replacing FPM DRAM devices in mainstream personal computers. The EDO DRAM device added a new output enable ($\overline{OE}$) signal and allowed the DRAM device to hand over control of the output buffer from the $\overline{CAS}$ signal to the $\overline{OE}$ signal. Consequently, read data on the output of the DRAM device can remain valid for a longer time after the $\overline{CAS}$ signal is driven high, thus the name "extended data-out."

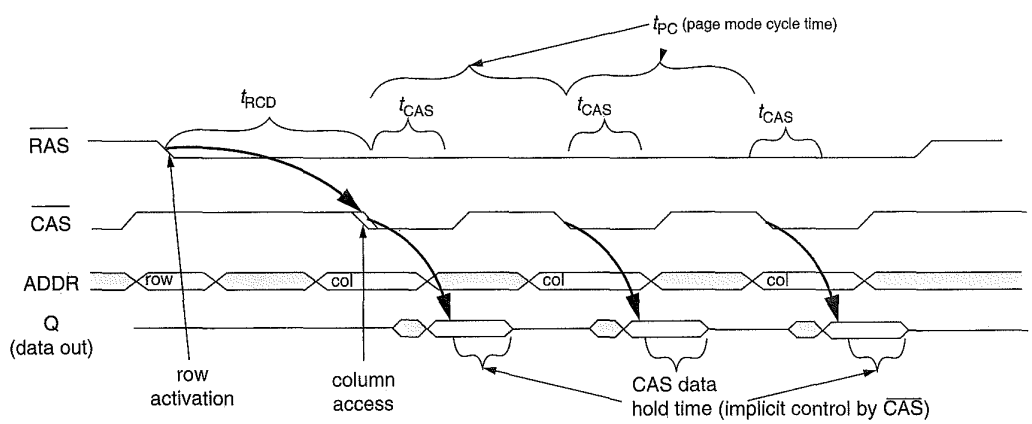


FIGURE 12.7: Multiple column read command timing for an FPM DRAM device.

Figure 12.8 gives the timing for three column read commands in an EDO DRAM device. The figure shows that the explicit control of the output data means that the $\overline{CAS}$ signal can be cycled around faster in an EDO DRAM device as compared to a FPM DRAM device, without impacting CAS data hold time. Consequently, EDO devices can achieve better (shorter) page mode cycle time, leading to higher device bandwidth when compared to FPM DRAM devices of the same process generation.

The EDO DRAM device achieved some success in replacing FPM DRAM devices in some markets, particularly the personal computer market. The higher bandwidth offered by the EDO DRAM device was welcomed by system design engineers and architects, as faster processors drove the need for higher memory bandwidth. However, the EDO DRAM device was unable to fully satisfy the demand for higher memory bandwidth, and DRAM engineers began to seek an evolutionary path beyond the EDO DRAM device.

The BEDO DRAM device builds on EDO DRAM by adding the concept of "bursting" contiguous blocks of data from an activated row each time a new column address is sent to the DRAM device. Figure 12.9 illustrates the timing of several column access commands for a BEDO DRAM device. The figure shows that the row activation command is given to the BEDO device by driving the $\overline{RAS}$ signal low. The column access command in the BEDO device is then initiated by driving the $\overline{CAS}$ signal low, and the address of the column access command is latched in at that time. Figure 12.9 shows that unlike the DRAM devices that preceded it, the BEDO DRAM device then internally generates four consecutive column addresses and places the data for those columns onto the data bus with each falling edge on the $\overline{CAS}$ signal. By eliminating the need to send successive column addresses over the bus to drive a burst of data in response to each microprocessor request, the BEDO device eliminates a significant amount of timing uncertainty between successive addresses, thereby further decreasing the page mode cycle time that correlates to increasing DRAM device bandwidth.

12.3 Modern-Commodity DRAM Devices

The nature of the commodity DRAM business means that any proposed change to the DRAM device must incur relatively low circuit overhead

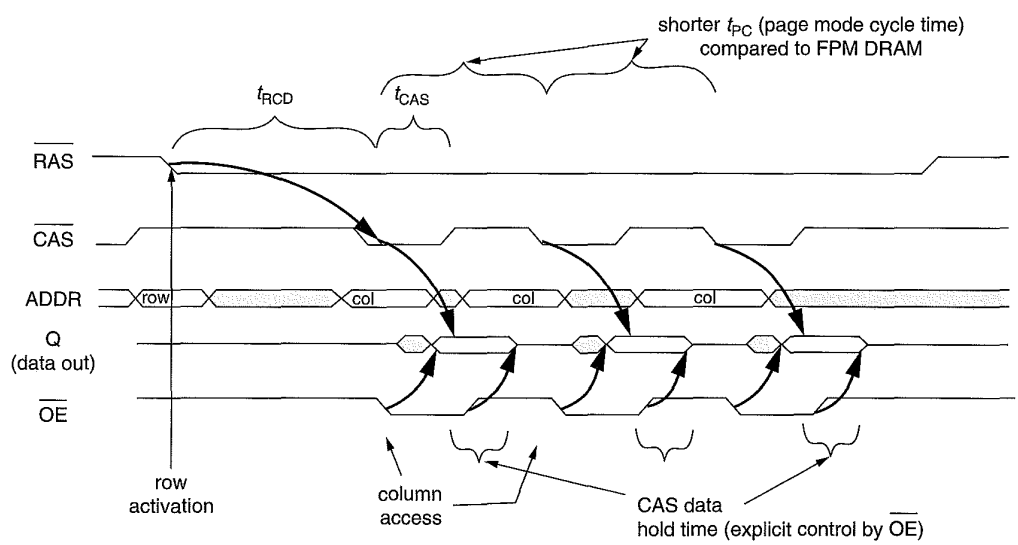


FIGURE 12.8: Multiple read command timing for an EDO DRAM device.

for the change to be acceptable to DRAM device manufacturers. Consequently, changes to DRAM devices are often evolutionary (low die cost) rather than revolutionary (high die cost). For example, the circuit overhead for an EDO DRAM device over that of an FPM DRAM device was minimal, and the evolutionary step required only a subtle change in protocol. However, owing to the convergence of factors such as the push to attain higher device bandwidth and more functionality from the DRAM device, DRAM manufacturers and systems manufacturers gathered through the *Joint Electron Device Engineering Council* (JEDEC) and collectively defined a new DRAM device, the SDRAM device. Compared to the EDO DRAM device that preceded it, the SDRAM device contained many new features that separated it distinctly from DRAM devices that preceded it. In this manner, the SDRAM device represents a break in the commodity path of the DRAM family tree and separates the *Modern-Commodity* DRAM devices from the *Historical-Commodity* DRAM devices.

The SDRAM device proved to be a highly successful DRAM device that proliferated into many different applications. The SDRAM device also became the basis for many different DRAM devices that followed it. The device architecture of commodity DRAM devices such as the Dual Data Rate (DDR) SDRAM, the Dual Data Rate II (DDR2) SDRAM, and Dual Data Rate III (DDR3) SDRAM is the direct descendant of the SDRAM device architecture. The venerable SDRAM device can also claim as its descendant, directly or indirectly, DRAM devices such as the *Graphics DDR* (GDDR) SDRAM, *Graphics DDR2* (GDDR2) SDRAM, *Graphics DDR3* (GDDR3) SDRAM, *Enhanced SDRAM* (ESDRAM), and *Virtual Channel SDRAM* (VCDRAM), as well as *Mobile SDRAM* and *Mobile DDR SDRAM*. In this section, the evolutionary development for modern-commodity DRAM devices from SDRAM to DDR3 SDRAM devices are described and examined in detail.

12.3.1 Synchronous DRAM (SDRAM)

The SDRAM device is the first device in a line of modern-commodity DRAM devices. The SDRAM device represented a significant departure from the DRAM devices that preceded it. In particular, SDRAM devices differ from previous generations of FPM and EDO DRAM devices in three significant ways: the SDRAM device has a synchronous device interface, all SDRAM devices contain multiple internal banks, and the SDRAM device is programmable.

DRAM devices are controlled by memory controllers optimized for specific systems. In the case of FPM and EDO DRAM devices used in modern computer systems, the memory controller typically operated at specific, fixed frequencies to interface with other components of the system. The asynchronous nature

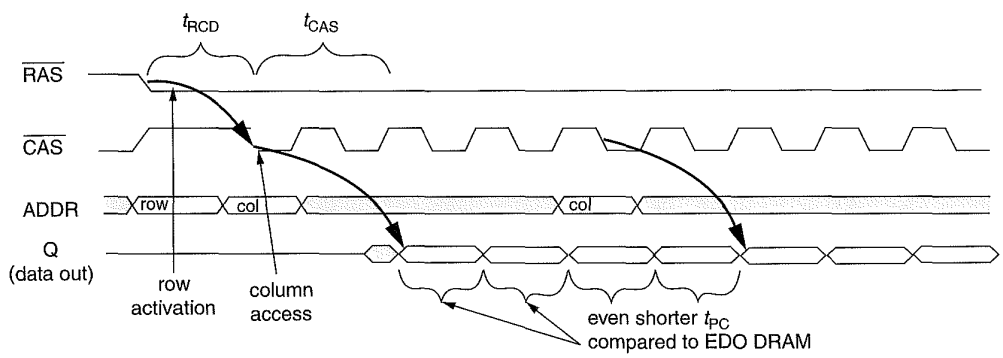


FIGURE 12.9: Multiple column read command timing for a BEDO DRAM device.

of FPM and EDO devices—that can, in theory, enable different controller and system designs to aggressively extract performance from different DRAM devices with different timing parameters—was more of a hindrance than an asset in the design of high volume and relatively high-performance memory systems. Consequently, computer manufacturers pushed to place a synchronous interface on the DRAM device that also operated at frequencies commonly found in the computer systems of that era, resulting in the SDRAM device.

In FPM and EDO DRAM devices, the $\overline{RAS}$ and $\overline{CAS}$ signals from the memory controller directly control latches internal to the DRAM device, and these signals can arrive at the DRAM device's pins at any time. The DRAM devices then respond to the $\overline{RAS}$ and $\overline{CAS}$ signals at the best possible speed that they are inherently capable of. In SDRAM devices, the $\overline{RAS}$ and $\overline{CAS}$ signal names were retained for signals on the command bus that transmits commands, but these specific signals no longer control latches that are internal to the DRAM device. Rather, the signals deliver commands on the command bus of the SDRAM device, and the commands are only acted upon by the control logic of the SDRAM device at the falling edge of the clock signal. In this manner, the operation of the state machine in the DRAM device moved from the memory controller into the DRAM device, enabling features such as programmability and multi-bank operation.

The second feature that significantly differentiated the SDRAM device from FPM and EDO devices that preceded it was that the SDRAM device contained multiple banks internally. The presence of multiple, independent banks in each SDRAM device means that while one bank is busy with a row activation command or a precharge command, the memory controller can send a row access command to initiate row activation in a different bank or can send a column access command to a different open bank. The multi-bank architecture of the SDRAM device means that multiple memory requests can be pipelined to different banks of a single SDRAM device. The first-generation 16-Mbit SDRAM device contained only 2 banks of independent DRAM arrays, but higher capacity SDRAM devices contained 4 independent banks in each device.

The SDRAM device also contains additional functionalities such as a column-access-and-precharge command. These functionalities along with the programmable state machine and multiple banks of sense amplifiers mean that the SDRAM device contains significant circuit overhead compared to FPM and EDO DRAM devices of the same capacity and process generation. However, since the circuit overhead was relatively constant and independent of device capacity, the die size overhead of the additional circuitry became less of an issue with larger device capacities. Consequently, the relatively low delta in manufacturing costs in combination with demonstrable performance benefits enabled 100-MHz, 64-Mbit SDRAM devices to take over from FPM and EDO devices as the mainstream commodity DRAM device.

Figure 12.10 shows a block diagram of an SDRAM device. The figure shows that the SDRAM device, unlike the FPM DRAM device in Figure 12.6, has 4 banks of DRAM arrays internally, each with its own array of sense amplifiers. Similar to the FPM device, the SDRAM device contains separate address registers that are used to control dataflow on the SDRAM device. In case of a row access command, the address from the address register is forwarded to the row address latch and decoder, and that address is used to activate the selected wordline. Data is then discharged onto the bitlines, and the sense amplifiers array senses, amplifies, and holds the data for subsequent column accesses. In the case of a column read command, the data is sent through multiple levels of multiplexors, then through the I/O gating structure out to the data bus, and eventually driven into the memory controller. In the case of a column write command, the memory controller places data on the data bus, and the SDRAM device then latches the data in the data in register, drives the data through the I/O gating structure, and overwrites data in the sense amplifier arrays; the sense amplifiers, in turn, drive the new data values into the DRAM cells through the open access transistors.

In an SDRAM device, commands are decoded on the rising edge of the clock signal (CLK) if the chip-select line (CS#) is active. The command is asserted by the DRAM controller on the command bus, which consists of the write enable (WE#), column access (CAS#), and row access (RAS#) signal lines. Although the signal lines have function-specific names, they essentially form a command bus, allowing the SDRAM device to recognize more commands without the use of

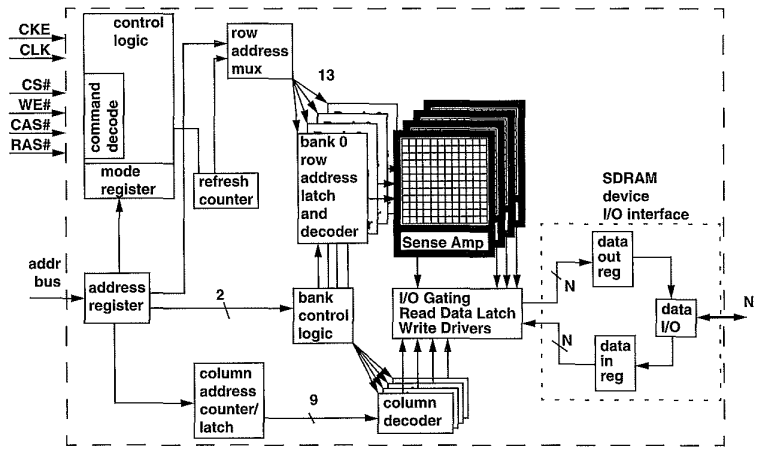


FIGURE 12.10: SDRAM device architecture with 4 banks.

additional signal lines. Table 12.1 shows the command set of the SDRAM device and the signal combinations on the command bus. The table shows that as long as CS# is not selected, the SDRAM device ignores the signals on the command bus. In the case where CS# is active on the rising edge of the clock, the SDRAM device then decodes the combination of control signals on the command bus. For example, the SDRAM device recognizes that the combination of an active low voltage value on RAS#, an interactive high voltage value on CAS#, and an inactive high voltage value on WE# as a row activation command and begins the row activation process on the selected bank, using the row address as provided on the address bus.

A different combination of signal values on the command bus allows the SDRAM device to load in new values for the mode register. That is, in the case where CS#, RAS#, CAS#, and WE# are all active on the rising edge of the clock signal, the SDRAM device decodes the load mode register command and loads the mode register from values presented on the address bus.

The third feature in an SDRAM device that differentiates it from previous DRAM devices is that the SDRAM device contains a programmable mode register, and the behavior of the DRAM device depends on the values contained in the various fields of the mode register. The presence of the mode register means that the SDRAM device can exhibit different behaviors in response to a given command. Specifically, the mode register in an SDRAM device allows it to have programmable CAS latency, programmable burst length, and programmable burst order.

Figure 12.11 shows that in an SDRAM device, the mode register contains three fields: CAS latency, burst type, and burst length. Depending on the value of the CAS latency field in the mode register, the DRAM device returns data two or three cycles after the assertion of the column read command. The value of the burst type determines the ordering of how the SDRAM device returns data, and the burst length field determines the number of columns that an SDRAM device will return to the memory controller with a single column read command. SDRAM devices can be programmed to return 1, 2, 4, or 8 columns or an entire row. Direct RDRAM devices and DDRx SDRAM devices contain more mode registers that control an ever-larger set of programmable operations, including, but not limited to, different operating modes for power conservation, electrical termination calibration modes, self-test modes, and write recovery duration.

To execute a given command, the control logic block on the SDRAM device accepts commands sent on the command bus from the memory controller. Then, depending on the type of command

TABLE 12.1 Command definition on the SDRAM device

Command	CS#	RAS#	CAS#	WE#	addr
command inhibit (nop)	H	X	X	X	X
no operation (nop)	L	H	H	H	X
active (activate row - RAS)	L	L	H	H	addr
read (start read - CAS)	L	H	L	H	addr
write (start write - CAS W)	L	H	L	L	addr
burst terminate	L	H	H	L	X
precharge	L	L	H	L	**
auto refresh	L	L	L	H	X
load mode register	L	L	L	L	code

**Bank address, or all banks with a_10 assertion.

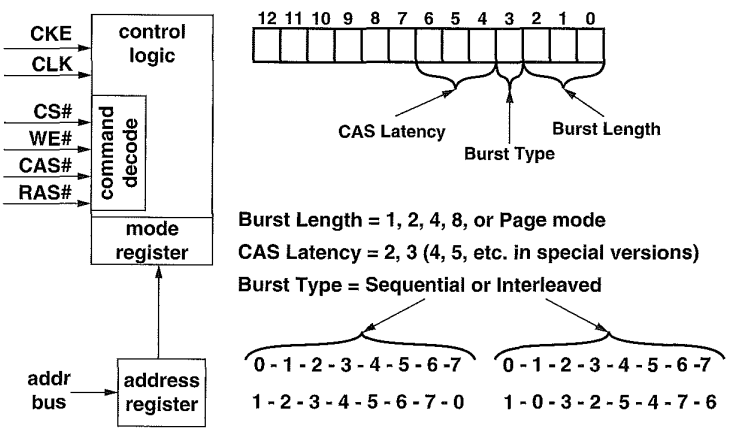


FIGURE 12.11: Programmable mode register in an SDRAM device.

and values contained in the respective fields of the mode register, the SDRAM device performs specific sequences of operations on successive clock cycles without requiring clock-by-clock control from the memory controller. For example, in the case of the column-read-and-precharge command, the SDRAM device accepts the command, and then, depending on the value programmed into the mode register, the SDRAM device begins to return data two or three clock cycles after the command was asserted on the command bus. The burst length and burst order of the single column access command also depends on the value programmed in the mode register. Then, the SDRAM device automatically precharges the DRAM bank after the column read command completes.

SDRAM-Access Protocol

Figure 12.12 illustrates some basic concepts of data access to an SDRAM memory system. The figure shows a total of three commands: a row activation command to bank i of rank n, followed by a column read access command to the same bank, followed by another column read access command to an open bank in a different rank. Figure 12.12 shows that the interface of the SDRAM memory system presents the memory system as a synchronous state machine that responds to commands issued from the memory controller. Specifically, Figure 12.12 shows that a row activation is started in an SDRAM device by the device latching the command and addresses on the rising edge of the clock signal. The SDRAM device then decodes the command and transitions the state machine on the DRAM device that is appropriate to the specific command received. In Figure 12.12, two cycles after the row activation command, the row-column delay time is satisfied, and the row is then assumed by the memory controller to be open. The memory controller then places the column read access command on the command bus, and t_{CAS} time later, the SDRAM device begins to return data from the just-opened row to the memory controller. In Figure 12.12, the SDRAM devices are programmed to return four consecutive columns of data for each column access command, so four consecutive columns of data are placed onto the data bus by the DRAM device without further interaction from the memory controller. Finally, Figure 12.12 subtly illustrates that the synchronous interface of the SDRAM memory system is only a convenient illusion. For the case of two column read commands to different ranks, the change in the bus master of the data bus on back-to-back clock cycles leads to some minor uncertainty on the timing.

Die Photo and a TSOP Package

Figure 12.13 shows the die photograph of a 256-Mbit SDRAM device from Micron. In the figure, much of the surface area of the silicon die is dominated by the regular structures of the DRAM arrays.

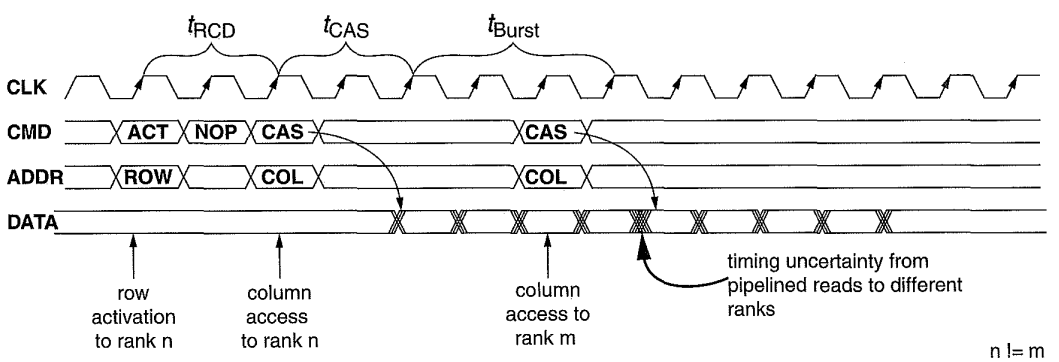


FIGURE 12.12: A row activation, followed by a column access to rank n, followed by a column access to rank m.

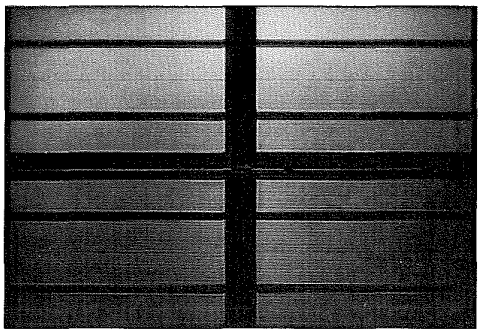


FIGURE 12.13: 256-Mbit SDRAM device from Micron. (Photo courtesy of Micron.)

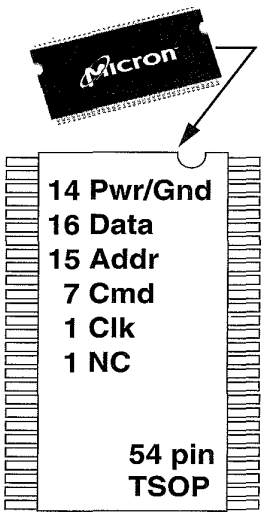


FIGURE 12.14: TSOP for an SDRAM device.

In this case, roughly 70% of the silicon surface is used by the DRAM arrays, and the rest of the area is taken up by I/O pads, sense amplifiers, decoders, and the minimal control logic. The SDRAM device shown in Figure 12.13 is manufactured on a DRAM-optimized, 0.11-μm process with three layers of metal interconnects and six layers of polysilicon. The die size of the SDRAM device is approximately 45 mm^2.

SDRAM devices are packaged in a low-cost *Thin, Small Outline Package* (TSOP). Figure 12.14 shows a 54-pin TSOP for an SDRAM device with a 16-bit-wide data bus. In a x16 SDRAM device, 14 pins on the 54-pin TSOP are used for power and ground, 16 pins are used for the data bus, 15 pins are used for the address bus, 7 pins are used for the command bus, and 1 pin is used for the clock signal.

PC100—The Proliferation of Extended, Rigorous DRAM Standardization and Qualification Processes

In April 1998, Intel Corp. introduced a new system controller that was the first controller to operate the SDRAM memory system at 100 MHz. Prior to the rollout of the 440BX AGPset, engineers at Intel discovered that memory modules manufactured by different module manufacturers, possibly utilizing different DRAM devices from different manufacturers, may not inter-operate in a seamless manner when placed into the same SDRAM memory system despite the fact that each module individually meets technical requirements for 100-MHz operation according to JEDEC standards specification. To ensure the success of the 440BX system controller chipset and the associated Pentium II processor platform, Intel Corp., in conjunction with DRAM device, DRAM module, and system manufacturers, adopted a more stringent set of standards for 100-MHz SDRAM memory. This set of stringent

requirements for SDRAM memory to operate at 100 MHz was referred to as the PC100 standard.

The PC100 SDRAM standard ensured module inter-operability by further decreasing the allowable timing margins on the chip and module interface. The decreased timing margins placed more stringent requirements on SDRAM device and SDRAM memory module manufacturers. In an effort to alleviate the demand placed on these manufacturers, Intel began to provide reference designs of SDRAM memory modules complete with bills of materials that specified prequalified SDRAM parts, schematics that illustrated connection points on a memory module, and Gerber files that specified the connections within the PCB layers of a memory module, as well as the values and placement of resistors and capacitors on a memory module. The reference design approach reduced the burden placed on memory module manufacturers and allowed the PC100 standard to proliferate. Consequently, PC100 SDRAM memory modules quickly gained popularity as end-users were assured of high performance and trouble-free memory system configuration, regardless of configuration and manufacturer of the DRAM device or modules.

Subsequent to the PC100 standardization effort, Intel attempted to shift industry memory system architecture to Direct RDRAM. Consequently, Intel did not drive the 133-MHz PC133 standardization effort. Nevertheless, the qualification path set down by Intel to ensure compatibility between different manufacturers was used to drive subsequent standard qualification efforts for faster SDRAM and DDRx memory systems. Currently, the module standardization effort to ensure trouble-free inter-operability resides in a subcomittee within JEDEC.

12.3.2 Double Data Rate SDRAM (DDR SDRAM)

The *Double Data Rate* (DDR) Synchronous DRAM device evolved from, and subsequently replaced, the SDRAM device as the mainstream commodity DRAM device. Consequently, DDR SDRAM device architecture closely resembles SDRAM device architecture. The primary difference between DDR SDRAM device architecture and the SDRAM device architecture is that the SDRAM device operates the data bus at the same data rate as the address and command busses, while the DDR SDRAM device operates the data bus at twice the data rate of the address and command busses. The reason that the data bus of the DDR SDRAM device operates at twice the data rate of the address and command busses is that signal lines on the data bus of the traditional SDRAM memory system topology are more lightly loaded than signal lines on the address and command busses. Figure 12.15 shows

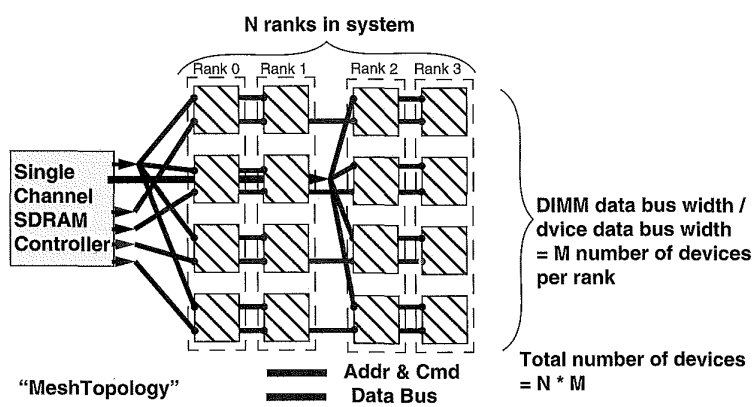


FIGURE 12.15: SDRAM memory system topology shows uneven loading on the different busses. Electrical loading on the address bus is much heavier than on the data bus.

the general topology of the SDRAM memory system where there are N ranks of DRAM devices in the memory system with M ranks of DRAM devices per rank. In the topology shown in Figure 12.15, each pin on the address and command bus may drive as many as $N * M$ loads, whereas the pins on the data bus are limited to the maximum of N loads. Consequently, the data bus can switch states at a much higher rate as compared to the address and command busses. DDR SDRAM devices are architected to take advantage of the imbalance in loading characteristics and operate the data bus at twice the data rate as the command and address busses.

DDR SDRAM-Access Protocol

Figure 12.16 illustrates basic concepts of data access to a DDR SDRAM memory system. A total of three commands are shown: a row activation command to bank i of rank n, followed by a column read access command to the same bank, followed by another column read access command to an open bank in a different rank. Figure 12.16 shows that data transfer occurs at twice the rate on the data bus as compared to the address and command bus. Figure 12.16 also shows that the DDR SDRAM memory system uses the data strobe signal (DQS), a signal not found in previous-generation SDRAM devices, to provide a source-synchronous timing reference signal between the source and the destination. In DDR SDRAM devices, the DQS signal is controlled by the device that sends the data on the data bus. In the case of a read command, the DQS signal is used by the DRAM device to indicate the timing of read data delivery to the memory controller. In the case of a write command, the DQS signal is used by the memory controller to indicate the timing of write data delivery from the memory controller to the DRAM device.

The timing diagram in Figure 12.16 shows a one-cycle bubble between data bursts from different ranks of DRAM devices. The one-cycle bubble exists because the DQS signal is a shared signal used by all data sources in the memory system. Consequently, one idle cycle is needed for one bus master to hand off control of the DQS signal line to another bus master, and the one-cycle bubble appears on the data bus as a natural result.

DDR SDRAM I/O Interface

Figure 12.17 presents a block diagram view of the DDR SDRAM device I/O interface. As Figure 12.16

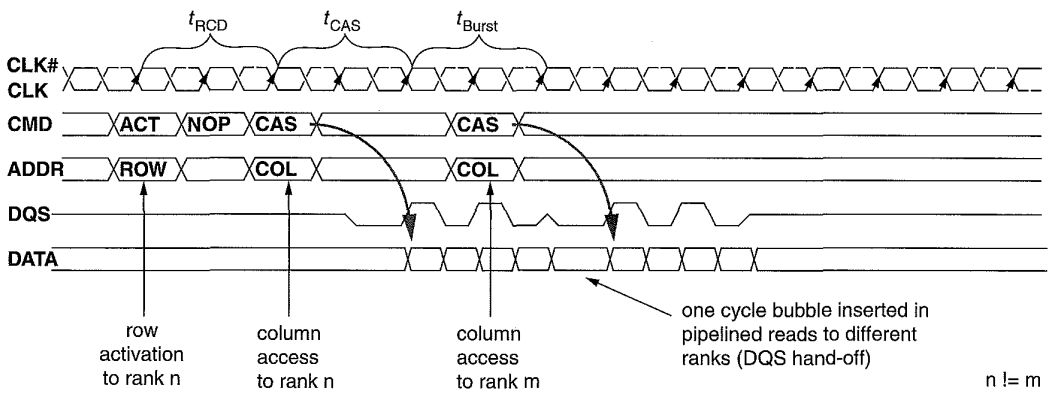


FIGURE 12.16: Accessing data in a DDR SDRAM memory system.

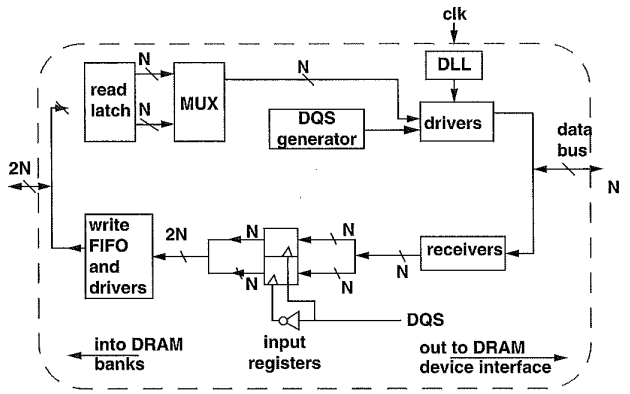


FIGURE 12.17: DDR SDRAM device I/O.

illustrates, DDR SDRAM memory systems transfer data on both edges of the DQS strobe signal. However, despite the increase in the rate of data transfer on the device interface, the rate of internal data transfer in the DDR SDRAM device is not similarly increased. Instead, in DDR SDRAM devices, the rate of data transfer is internally halved so that data movement occurs internally at twice the width, but half the rate of the device interface. DRAM device manufacturers have adopted the terminology of *M-bit prefetch* to describe the data rate multiplication architecture, where *M* represents the multiplication factor between the DRAM device's internal width of data movement and the width of the data bus on the device interface. In this nomenclature, DDR SDRAM devices are said to have a 2-bit prefetch architecture where 2 * *N* bits are moved internally at rate *Y*, but the DDR SDRAM device provides an *N*-bit-wide interface to the memory system that moves data at rate 2 * *Y*.

Aside from the difference in the I/O interface of the DRAM device, DDR SDRAM device architecture is otherwise identical to SDRAM device architecture. Consequently, some DRAM manufacturers created unified designs that can be bonded out as a DDR SDRAM device of data width X or an SDRAM device of data width 2 * *X*. These unified designs allowed these manufacturers to quickly shift wafer allocations to meet shifting market demands. However, these unified designs typically cost several percentage points of die overhead, so their use was limited to the transitional period between SDRAM and DDR SDRAM devices.

Series Stub Terminated Signaling Protocol

Aside from the changes to the I/O architecture of the DRAM device, the signaling protocol used by the DDR SDRAM device also differed from the signaling protocol used by the SDRAM device. The signaling protocol used in DDR SDRAM devices had to meet two conditions: better signal integrity to achieve the higher data rate and a signaling protocol that would still permit the DRAM device core and the DRAM device interface to share a common, yet lower voltage level. The *2.5-V Series Stub Terminated Logic* (SSTL-2) signaling protocol met the requirement for DRAM manufacturers to simultaneously achieve the higher signaling rates found in DDR SDRAM memory systems and to lower the device voltage from the 3.3 V used by SDRAM devices to 2.5 V. Consequently, SSTL-2 is used in all DDR SDRAM devices.

Figure 12.18 illustrates the idealized signal input and output characteristics for an SSTL-2 inverter. The figure shows that SSTL-2 differs from *Low-Voltage Transistor-Transistor Logic* (LVTTL) in that SSTL-2 uses a common reference voltage V_{ref} to differentiate between a logically high voltage state and a logically low voltage state. The use of the common voltage reference enables SSTL-2 devices to enjoy better voltage margins than LVTTL, despite the decrease in the overall voltage range from 3.3 to 2.5 V.

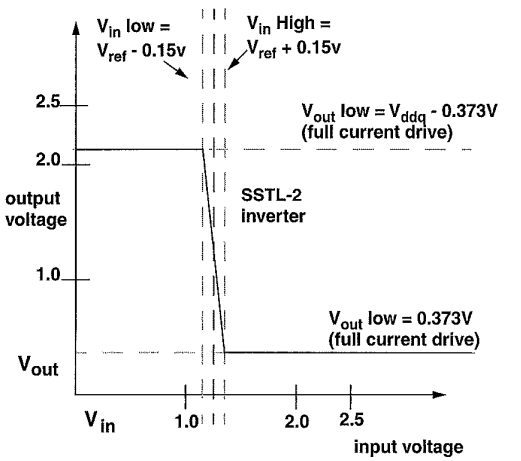


FIGURE 12.18: SSTL-2 signaling in DDR SDRAM devices.

12.3.3 DDR2 SDRAM

In the search for an evolutionary replacement to the DDR SDRAM device as the commodity DRAM device of choice, DRAM device manufacturers sought to achieve higher device data rates and lower power dissipation characteristics without substantially increasing the complexity, which translates to higher manufacturing cost, of the DRAM device. The DDR2 SDRAM device architecture was developed by a consortium of DRAM device and system manufacturers at JEDEC to meet these stringent requirements. The DDR2 SDRAM device was able to meet the requirement of a higher data transfer rate without substantially increasing the manufacturing cost of the DRAM device by further increasing the prefetch length, from 2 bits in DDR SDRAM device architecture to 4 bits. In the *M*-bit prefetch nomenclature, DDR2 SDRAM devices move 4 * *N* bits internally at rate *Y*, but provide an *N*-bit-wide interface to the memory system that moves data at a rate of 4 * *Y*. Figure 12.19 presents a block diagram view of the DDR2 SDRAM device I/O interface. The figure shows that DDR2 SDRAM devices further double the internal datapath of the DRAM device compared to that of a comparable DDR SDRAM device.

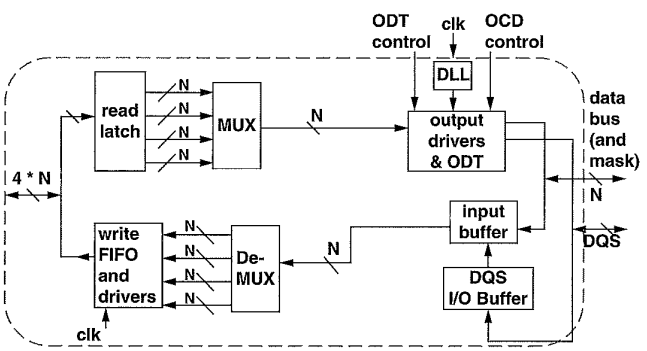


FIGURE 12.19: DDR2 SDRAM device I/O interface.

Figure 12.19 also shows another subtle difference between DDR and DDR2 SDRAM devices. The I/O interface of the DDR2 SDRAM device has an additional signal to control the termination characteristic of the input pin. The *On-Die Termination* (ODT) signal can be controlled by a DDR2 SDRAM memory controller to dynamically adjust the electrical characteristics of the memory system, depending on the configuration of the specific memory system.

12.3.4 Protocol and Architectural Differences

The evolutionary relationship between DDR SDRAM and DDR2 SDRAM device architectures means that the two devices architectures are substantially similar to each other. However, there are subtle architectural and protocol differences that separate the DDR2 SDRAM device from the DDR SDRAM device. For example, DDR2 SDRAM devices can support posted CAS commands, and DDR2 devices now mandate a delay between the column write command and data from the memory controller.

Figure 12.20 illustrates two pipelined transactions to different banks in a DDR2 SDRAM device. The figure shows that the DDR2 SDRAM device is programmable to the extent that it can hold a column access command for a certain number of cycles before it executes the command. The posted CAS command feature allows a DDR2 SDRAM memory controller to treat a row activation command and a column access command as a unitary command pair to be issued in consecutive cycles rather than two separate commands that must be controlled and timed separately. In Figure 12.20, the additional hold time for the CAS command is three cycles, and it is labelled as t_{AL}, for Additional Latency. Figure 12.20 also shows that the DDR2 SDRAM device now requires a write delay that is equivalent to $t_{CAS} - 1$ number of cycles. With the addition of t_{AL}, column read command timing can be simply referred to as Read Latency, or t_{RL}, and column write command timing can be simply referred to as Write Latency, or t_{WL}.

Differential Strobes and FBGA Packages

In addition to dynamic termination control, DDR2 SDRAM device architecture also contains other features that differentiate it from DDR SDRAM device architecture. For example, DDR2 SDRAM devices can optionally support a differential DQS signal, while DDR SDRAM devices only support a single ended DQS signal. The differential DQS signal enables the DDR2 SDRAM device to bolster the signal integrity

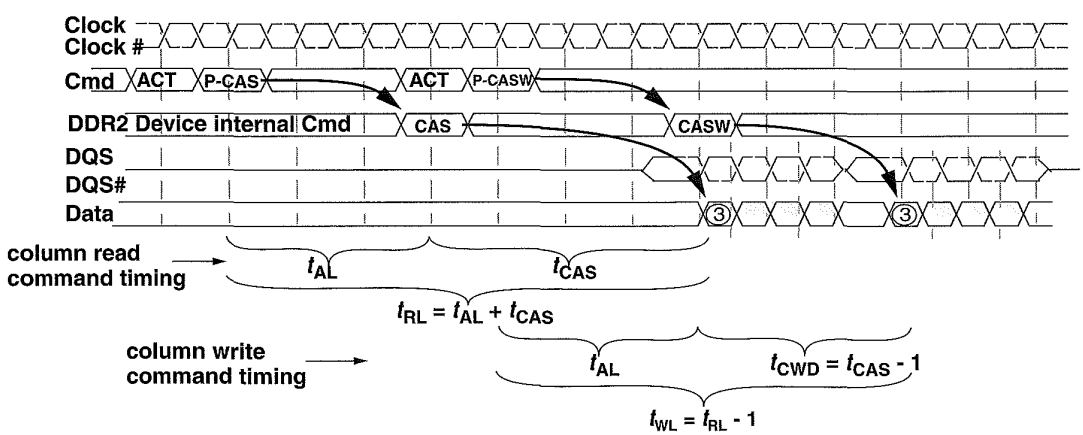


FIGURE 12.20: A posted column read command and a posted column write command in a DDR2 SDRAM system.

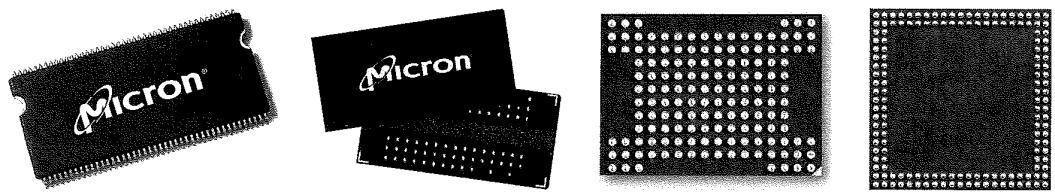


FIGURE 12.21: A DDR SDRAM device in a TSOP and a DDR2 SDRAM device in an FBGA package. (Photos courtesy of Micron.)

of the reference strobe, thus enabling it to operate at higher data rates. In addition to illustrating the progression of the posted column read and posted column write commands, Figure 12.20 also shows that data for column read commands, sent by the DRAM devices, is edge-aligned to the data strobe signal, while data for column write commands, sent by the DRAM controller, is center-aligned to the DQS and DQS# data strobe signals.[2]

Finally, Figure 12.21 shows a DDR SDRAM device in a TSOP and a DDR2 SDRAM device in a *Fine Ball Grid Array* (FGBA) package. FBGA packaging is more expensive than TSOP, but the ball grid contacts present less electrical parasitics for the signal transmission line. Consequently, FBGA packaging is required for the higher data rate DDR2 devices, while it remains optional for DDR SDRAM devices.

12.3.5 DDR3 SDRAM

Having learned many lessons in the evolutionary development of DDR and DDR2 SDRAM devices, DRAM device and systems manufacturers have continued on the path of further increasing DRAM device prefetch lengths to enable higher device data rates in the next-generation commodity DRAM device—DDR3 SDRAM. The DDR3 SDRAM device continues the technique of increasing prefetch lengths and employs a prefetch length of 8. Consequently, DDR3 SDRAM devices are expected to attain data rates that range between 800 Mbps per pin to 1.6 Gbps per pin, doubling the range of 400 to 800 Mbps per pin for DDR2 SDRAM devices.[3]

DDR3 SDRAM devices also contain additional enhancements not found in DDR2 SDRAM devices. For example, DDR3 SDRAM devices of all capacities have at least 8 banks of independent DRAM arrays, while the 8-bank architecture is limited to DDR2 devices with capacities of 1 Gbit or larger. DDR3 SDRAM devices will also have two features that may enable them to reduce refresh power consumption. One optional feature that will enable DDR3 SDRAM devices to reduce refresh power consumption is a temperature-dependent self-refresh mode. In this self-refresh mode, the rate of refresh and current of the self-refresh circuitry will be adjusted automatically by the DDR3 device, depending

[2]Read data in SDRAM devices is edge-aligned to the clock signal, while write data is center-aligned to the clock signal. The data timing described herein for DDR2 is similar for DDR, DDR2, and DDR3 SDRAM devices.
[3]At the time of the writing of this text, discussions are underway to extend the range of DDR3 data rate past 2 Gbps.

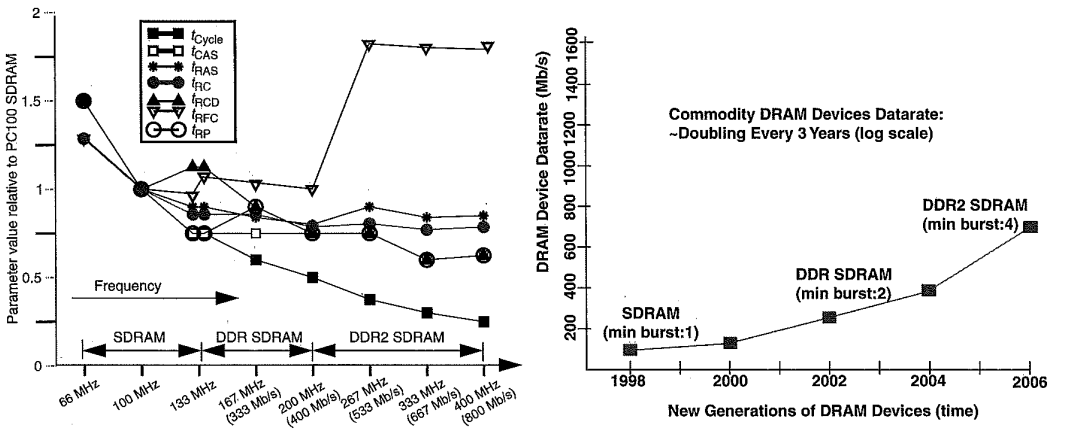


FIGURE 12.22: Commodity DRAM device timing and data rate scaling trends.

on the temperature of the device. The second feature that will enable DDR3 devices to reduce refresh power is that a DDR3 device can be programmed to refresh only a subset of the rows that contain data, rather than all of the rows once per fresh loop.[4] These features, in combination with the lower 1.5 V supply voltage, enable DDR3 devices to consume less power per unit of storage or unit of bandwidth.

Unfortunately, the high data transfer rate of DDR3 SDRAM devices requires significant trade-off in memory system configuration flexibility, a trade-off that will limit the utility of the device in unbuffered memory systems. To reach the high data rate of DDR3 SDRAM devices, DRAM device and system manufacturers have imposed the limit of two ranks of DRAM devices in a memory system, and the two ranks of DRAM devices are assumed to be located close to each other. Consequently, computer system manufacturers will not be able to design and market computers with traditional, unbuffered memory systems that still allow end-users to flexibly configure the capacity of the memory system as could be done with DDR and DDR2 SDRAM memory systems.

12.3.6 Scaling Trends of Modern-Commodity DRAM Devices

In the decade since the definition of the SDRAM standard, SDRAM devices and their descendants have continued on a general scaling trend of exponentially higher data rates and slowly decreasing timing parameter values for each generation of DRAM devices. Table 12.2 contains a summary of timing parameter values for selected SDRAM, DDR SDRAM, and DDR2 SDRAM devices. Table 12.2 shows that fundamental device operation latencies in terms of wall clock time have gradually decreased with successive generations of DRAM devices, while DRAM device data rates have increased at a much higher rate. Consequently, DRAM device operation latencies have, in general, increased in terms of cycles, despite the fact that the latency values in nanoseconds have decreased in general.

Figure 12.22 shows the scaling trends of commodity DRAM devices from 1998 to 2006. In the time period illustrated in Figure 12.22, random row cycle times in commodity DRAM devices decreased on the order of 7% per year. In contrast, the data rate of

[4]Partial array refresh is also present in DDR2 SDRAM devices.

TABLE 12.2 Timing parameter values of various x4 SDRAM, DDR, and DDR2 devices

	SDRAM						DDR SDRAM						DDR2 SDRAM					
Capacity	16 Mbit		256 Mbit				512 Mbit						1 Gbit					
Manf.	Siemens		Micron		Micron		Micron		Micron		Micron		Micron		Micron		Micron	
Part No.	64V16220GU		48LC64M4A2		48LC64M4A2		46V128M4		46V128M4		46V128M4		47H256M4		47H256M4		47H256M4	
Bin	–10		–75		–7E		–6		–5B		–5		–3		–3E		–25	
Freq.	66 MHz		100 MHz		133 MHz		133 MHz		167 MHz		200 MHz		267 MHz		333 MHz		400 MHz	
Data rate	66 Mbps		100 Mbps		133 Mbps		266 Mbps		333 Mbps		400 Mbps		533 Mbps		667 Mbps		800 Mbps	
	clks	ns	clks	ns	clks	ns	clks	ns	clks	ns	clks	ns	clks	ns	clks	ns	clks	ns
t_{Cycle}	1	15	1	10	1	7.5	1	7.5	1	6	1	5	1	3.75	1	3	1	2.5
t_{CAS}	2	30	2	20	2	15	2	15	2.5	15	3	15	4	15	4	12	5	12.5
t_{FAW}	—		—		—								10	37.5	13	39	15	37.5
t_{RAS}	5	75	5	50	6	45	6	45	7	42	8	40	11	41.25	14	42	17	42.5
t_{RC}	6	90	7	70	8	60	8	60	10	60	11	55	15	56.25	18	54	22	55
t_{RCD}	2	30	2	20	3	22.5	3	22.5	3	18	3	15	4	15	4	12	5	12.5
t_{RFC}	6	90	7	70	9	67.5	10	75	12	72	14	70	34	127.5	42	129	51	127.5
t_{RP}	2	30	2	20	2	15	2	15	3	18	3	15	4	15	4	12	5	12.5
t_{RRD}	2	30	2	20	2	15	2	15	2	12	2	10	3	11.25	3	9	4	10
t_{RTP}	1	15	1	10	1	7.5	1	7.5	1	6	1	5	2	7.5	3	9	3	7.5
t_{WR}	2	30	2	20	2	15	2	15	3	18	3	15	4	15	5	15	6	15
t_{WTR}	—		—		—		1	7.5	2	12	2	10	2	7.5	3	9	3	7.5

TABLE 12.3 Quick summary of SDRAM and DDRx SDRAM devices

		SDRAM	DDR SDRAM	DDR2 SDRAM	DDR3 SDRAM
Supply voltage		3.3 V	2.5[a] V	1.8 V	1.5 V
Signaling		LVTTL	SSTL-2	SSTL-18	SSTL-15
Bank count		4[b]	4	4[c]	8
Data rate range		66~133	200~400	400~800	800~1600
Prefetch length		1	2	4	8
Internal datapath width	×4	4	8	16	32
	×8	8	16	32	64
	×16	16	32	64	128

[a]400-Mbps DDR SDRAM standard voltage set at 2.6 V.
[b]16-Mbit density SDRAM devices only have 2 banks in each device.
[c]256- and 512-Mbit devices have 4 banks; 1-, 2-, and 4-Gbit DDR2 SDRAM devices have 8 banks in each device.

commodity DRAM devices doubled every three years. Consequently, the relatively constant row cycle times and rapidly increasing data rates mean that longer requests or a larger number of requests must be kept in flight by the DRAM memory controller to sustain high bandwidth utilization.

Figure 12.22 also shows an anomaly in that the refresh cycle time t_{RFC} has increased rather than decreased in successive generations of DRAM devices, unlike the scaling trends for other timing parameters. Although the t_{RFC} values reported in Table 12.2 are technically correct, the illustrated trend is somewhat misleading. That is, t_{RFC} increased for the DDR2 devices examined over that for the DDR SDRAM devices due to the fact that the DDR SDRAM devices examined in Table 12.2 are 512-Mbit devices, whereas the DDR2 SDRAM devices examined in Table 12.2 are 1-Gbit devices. In this case, the larger capacity means that a larger number of cells must be refreshed, and the 1-Gbit DDR2 DRAM device must take longer to perform a refresh command or draw more current to refresh twice the number of DRAM storage bits in the same amount of time as the 512-Mbit DDR SDRAM device. Table 12.2 shows that DRAM manufacturers have, in general, chosen to increase the refresh cycle time t_{RFC}, rather than significantly increase the current draw needed to perform a refresh command. Consequently, the refresh cycle time t_{RFC} has not decreased in successive generations of DRAM devices when the general trend of increasing capacity in each generation is taken into account.

Table 12.3 contains a summary of modern-commodity SDRAM devices, showing the general trends of lower supply voltages to the devices and higher operating data rates. Table 12.3 also summarizes the effect of the increasing prefetch length in SDRAM, DDR SDRAM, DDR2 SDRAM, and DDR3 SDRAM devices. Table 12.3 shows the worrying trend that the increasing data rate of commodity DRAM devices has been achieved at the expense of increasing granularity of data movement. That is, in a ×4 SDRAM device, the smallest unit of data movement is a single column of 4 bits. With increasing prefetch length, the smallest unit of data movement has also increased proportionally. The increasing granularity of data movement means that the commodity DRAM system is losing randomness of data access, and the higher device bandwidth is achieved only by streaming data from spatially adjacent address locations. The situation is being compounded with the fact that ×8 DRAM devices are far outselling ×4 DRAM devices, and ×4 DRAM devices are only being used in memory systems that require maximum capacity. Consequently, ×4 DRAM devices are now selling at a price premium over ×8 and ×16 DRAM devices.

The larger granularity of data movement has serious implications in terms of random access performance as well as memory system reliability. For example, in a DDR2 SDRAM memory system, the loss of a single device will take out 16 bits of data in a ×4 device and 32 bits of data in a ×8 device. Consequently, a minimum data bus width of 144 bits is needed, in combination with sophisticated error correction hardware circuitry to cover for the loss of 16 bits of data in a ×4 device and 32 bits of data in a ×8 device.

12.4 High Bandwidth Path

In 1990, the 80386 processor was the dominant desktop processor, and memory bandwidth was not a limiting issue as it is in more modern memory systems. However, in that same year, Rambus Corp. was founded with a focus to design high-speed chip interfaces, specifically memory system interfaces. Rambus Corp.'s focus on high-speed signaling technology led it to design high bandwidth memory interfaces and systems that differ radically from commodity DRAM memory systems in terms of signaling protocol, system topology, device architecture, and access protocol. As a result of its singular focus on high device and system bandwidth, the high bandwidth path of DRAM device architecture evolution is dominated by memory systems developed by Rambus Corp.

In the years since its founding, Rambus Corp. has had various levels of contribution and involvement in different memory systems. However, two high bandwidth memory systems are often cited when Rambus technology is brought up for discussion: the *Direct Rambus Dynamic Random-Access Memory* (Direct RDRAM) memory system and the *Extreme Data Rate* (XDR) memory system. In this section, the unique features of these two memory systems are singled out for examination.

12.4.1 Direct RDRAM

The Direct RDRAM device and system architectures are radically different from the conventional, commodity DRAM device and system architectures. In this section, we begin with the fundamental difference between the Direct RDRAM memory system and the commodity, cost-focused DRAM memory systems by starting with an examination of the high-speed signaling technology developed by Rambus, the *Rambus Signaling Level* (RSL) signaling protocol. RSL enabled Rambus Corp. to design high-speed DRAM device interfaces. However, the relatively slow DRAM circuits designed for use in low-cost commodity DRAM devices mean that the high-speed RSL signaling protocol must be coupled with suitable device and system architectures to attain high values of practical, sustainable bandwidth. In the following sections, the signaling technology, system architecture, device architecture, and access protocol of the Direct RDRAM memory system are systematically examined. The systematic examination of the Direct RDRAM memory system architecture begins with an examination of RSL.

The Rambus Signaling Level (RSL)

The RSL was Rambus' first signaling technology. RSL was designed as a high-speed, singled-ended, multidrop, bidirectional bus signaling technology. RSL is designed as a signaling technology that can support a variable number of loads on the same bus—between 1 and 32 DRAM devices can be connected to the same bus. RSL debuted with data rates of 500 Mbps and reached over 1.2 Gbps in various configurations.

Figure 12.23 shows that the RSL is defined to swing between 1.0 and 1.8 V. RSL is designed to operate as a high-speed signaling system that makes use of current mode output drivers with carefully controlled slew rates to deliver low-voltage signals across transmissions lines with carefully controlled impedance characteristics.

Memory System Architecture

The system architecture of the Direct RDRAM memory system differs dramatically from system architectures of SDRAM and DDRx SDRAM memory systems. The key elements of the Direct RDRAM memory system architecture that differentiate it from SDRAM and DDRx SDRAM memory system architectures are path-matched address, command and data bus topology, separate row and column address

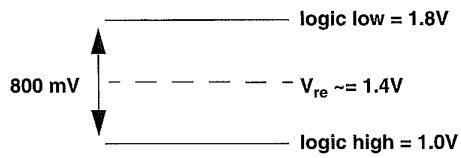


FIGURE 12.23: Direct RDRAM signaling: Rambus signaling levels.

channels with encoded command packets, separate data channel, multiple cycle, packet-based command and address assertion from memory controller to memory device, and the absence of a critical word forwarding data burst.

The system architecture of the Direct RDRAM memory system was designed to sustain high pin-bandwidth regardless of the number of DRAM devices in the memory system. The motivation for this design decision is that it allows for a high degree of performance scalability with multiple channels of memory, regardless of the number of DRAM devices per channel. In theory, an embedded system with a single channel memory system with a single DRAM device in the memory system can enjoy as much pin-bandwidth as a server system with multiple channels of memory and fully populated with 32 devices per channel. Unlike high data rate DDR2 and DDR3 memory systems that rely on multiple ranks of DRAM devices to collectively provide sufficient bandwidth to saturate the channel, a single-rank Direct RDRAM memory system can provide as much bandwidth as a multi-rank Direct RDRAM memory system.

Device Architecture

Figure 12.24 illustrates the device organization of a 288-Mbit Direct RDRAM device with 32 interleaved and dependent (32d) banks internally. The Direct RDRAM device can be architected to contain different numbers of banks. The organization illustrated in Figure 12.24 contains 32 dependent banks.[5] In the 32d Direct RDRAM device architecture, each bank is split into two halves, an upper half bank and a lower half bank, and adjacent banks share common sets of sense amplifiers. That is, the lower half of bank i shares the same set of sense amplifiers with the upper half of bank $i + 1$. Consequently, adjacent banks i and $i + 1$ cannot both be open at the same time.

One difference between Direct RDRAM memory systems versus SDRAM and DDRx SDRAM memory systems is that SDRAM and DDRx SDRAM memory systems rely on wide data busses with a non-power-of-two number of devices in parallel to provide the requisite number of data and check bits for error detection and correction. In contrast, a single Direct RDRAM device may form the entire data bus width of a Direct RDRAM memory system. Consequently, different versions of Direct RDRAM devices are used in different Direct RDRAM memory systems. Direct RDRAM memory systems that do not require error correction use Direct RDRAM devices with a 16-bit-wide data bus, and Direct RDRAM memory systems that require error correction use Direct RDRAM devices with an 18-bit-wide data bus. Figure 12.24 illustrates a Direct RDRAM device with an 18-bit-wide data bus. The figure shows that the 18-bit-wide data bus is organized as two separate 9-bit-wide data busses.

Unlike SDRAM and DDRx SDRAM devices where a given command and the associated address are sent in a single clock cycle, Direct RDRAM devices encode

[5]A 4 independent (4i) banks per device architecture was promoted as a cost reduction initiative for desktop systems that are typically configured with multiple DRAM systems on a given RIMM. However the initiative did not gain traction, and the 32d device architecture was the primary device architecture at the 256-/288-Mbit device node.

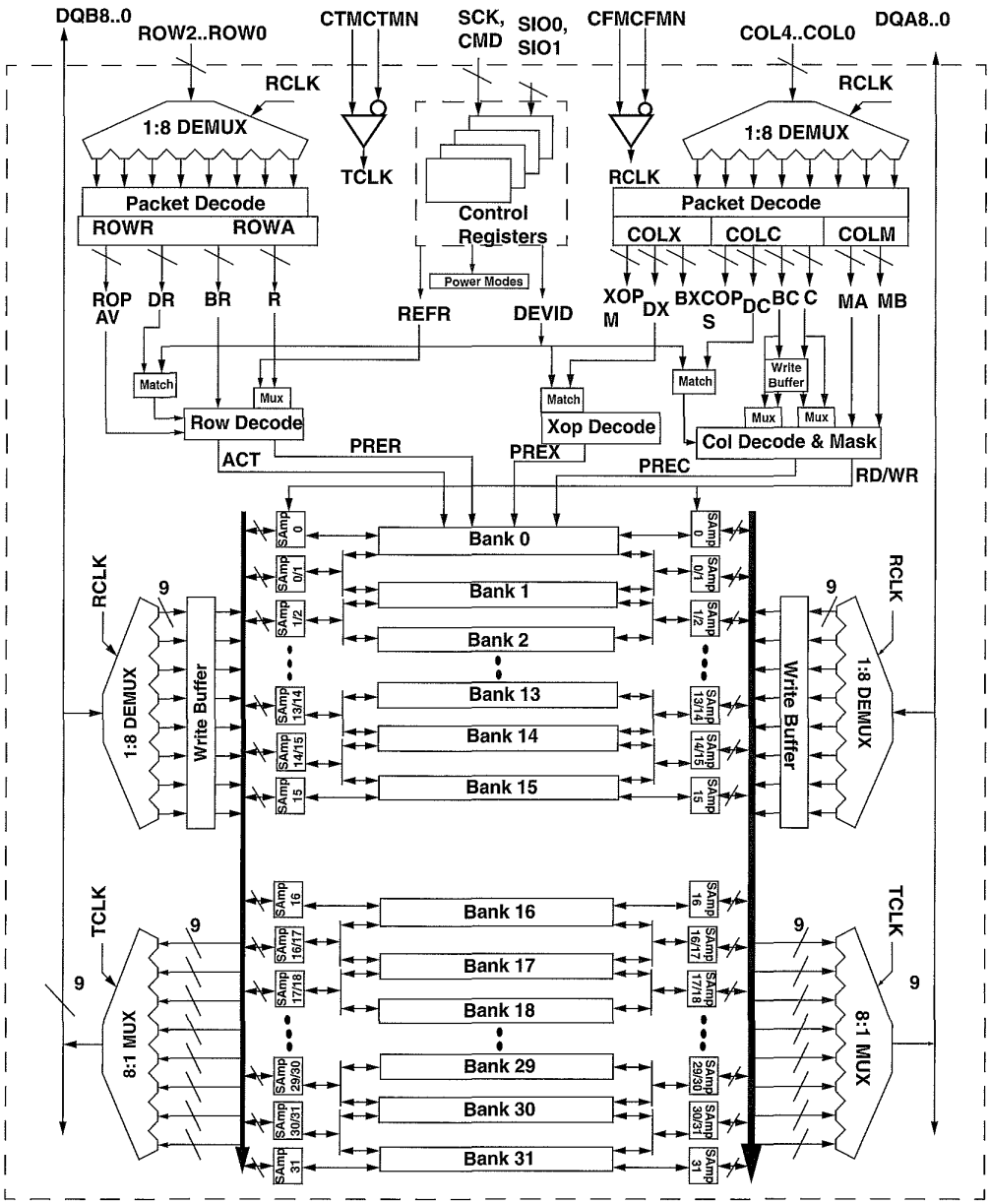


FIGURE 12.24: Split (32 interleaved) bank architecture of a Direct RDRAM device.

the command and address into a multi-cycle packet that must be buffered, de-multiplexed, and decoded by the Direct RDRAM device. In the case of a column read command, a read command packet (40 bits in size) is sent over the 5-bit-wide COL[4:0] column command bus on both the rising edge and the falling edge of the clock-from-master (CFM) clock signal. The 40-bit-wide packet is received and decoded by the Direct RDRAM device into COLX, COLC, and COLM command and address fields. The Direct RDRAM device then moves the requested data from the array of sense amplifiers through the 8:1 multiplexor onto the data bus. In this manner, the data prefetch architecture of Direct RDRAM devices is very similar to the internal device architecture of DDR3 SDRAM devices. However, unlike DDR3 SDRAM devices, Direct RDRAM devices do not support burst reordering to send the critical word first.

Topology

To take advantage of the high-speed signaling system, Rambus designed a new system topology for Direct RDRAM. Figure 12.25 shows that each Direct RDRAM device appears on a Direct DRAM channel as a single load. Figure 12.25 presents a topological view of the Direct RDRAM memory system with the memory controller connected to two memory modules, and each module contains four memory devices. The memory module for the Direct RDRAM memory system is referred to as a *Rambus In-line Memory Module* (RIMM) by Rambus. Each memory device illustrated in the Direct RDRAM memory system in Figure 12.25 presents a 16-bit-wide data interface to the data bus. In Figure 12.25, all of the Direct RDRAM devices are connected on the same channel with a 16-bit-wide data bus. In essence, each Direct RDRAM device is a single rank of memory in a Direct RDRAM memory system.

In a Direct RDRAM memory system, the clock-from-master signal, the clock-to-master signal, the column address and command bus, the row address and command bus, and the two 8-bit-wide data busses are routed in parallel. Unlike the mesh topology of the SDRAM and the DDRx SDRAM memory systems, the topology of the Direct RDRAM memory system allows all of the signal interfaces of the DRAM device to operate with minimal skew relative to the system clock signals. The Direct RDRAM memory system uses a scheme where the clock signals propagate alongside data signals to minimize timing skew between clock and data. The minimal clock-data timing skew means that the Direct RDRAM memory system can avoid the insertion of idles in the protocol to account for timing uncertainties and achieve high bandwidth efficiency. The arrangement of the memory device in the Direct RDRAM memory system also means that with more devices, the bus turnaround time increases and results in longer memory latency.

Access Protocol

The memory-access protocol of the Direct RDRAM memory system is quite different from access protocols of traditional memory systems such as SDRAM,

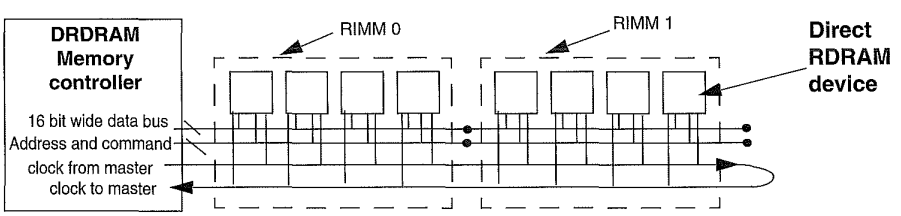


FIGURE 12.25: Direct RDRAM system topology.

DDR SDRAM, and DDR2 SDRAM memory systems. As a consequence of the matched topology of the clock, data, and command and address busses, the Direct RDRAM memory controller encodes all commands and addresses into packets[6] and transmits them in 8 half cycles,[7] a period of time called an *octcycle,* from the controller to the DRAM device. In the case of row commands, the Direct RDRAM controller encodes the row address and access commands into row access command packets that are 24 bits in length and transports the 24-bit-long packets from the controller to a Direct RDRAM device over 3 signal wires in 8 half cycles. In the case of column access commands, the Direct RDRAM controller encodes the column addresses and column access commands into packets that are 40 bits in length for transport to a Direct RDRAM device over 5 signal wires in 8 half cycles.

The Direct RDRAM memory system transports data in a manner that is similar to the command transport mechanism. In the case of a column read command, 128 bits of data are transported from a single Direct RDRAM device over 16 signal wires in 8 half cycles to the memory controller. The timing of a memory read command in a Direct RDRAM memory system is illustrated in three separate steps in Figure 12.26. The figure shows that the Direct RDRAM memory controller first packs and encodes a row activation command and then transmits the packed row activation command over the span of four clock cycles in step 1. The packed column access command is then transmitted to the Direct RDRAM device in step 2. Finally, in step 3, data is returned to the controller by a given Direct RDRAM device (device 0 in Figure 12.26) in over 8 half cycles. The 128 bits, or 16 bytes, of data are also referred to as a *dualoct* by Rambus.

One interesting detail about the Direct RDRAM memory-access protocol illustrated in Figure 12.26 is that the Direct RDRAM controller began to transmit the column read command almost immediately after the row access command, before t_{RCD} timing had been satisfied. The reason that the transmission of the column read command can begin almost immediately after the transmission of the row access command is that each command packet must be buffered, decoded, and then executed by the addressed Direct RDRAM device. Consequently, the column access command shown as step 2 in Figure 12.26 is not decoded until the transmission of the command has been completed and the entire packet has been received by the Direct RDRAM device. The optimal timing to minimize delay is then to coincide the row-activation-to-column-access delay with the end of the column access packet as shown in Figure 12.26.

The Direct RDRAM memory system is designed to sustain high bandwidth throughput, and the Direct RDRAM memory-access protocol allows a large combination of different memory-access commands to be

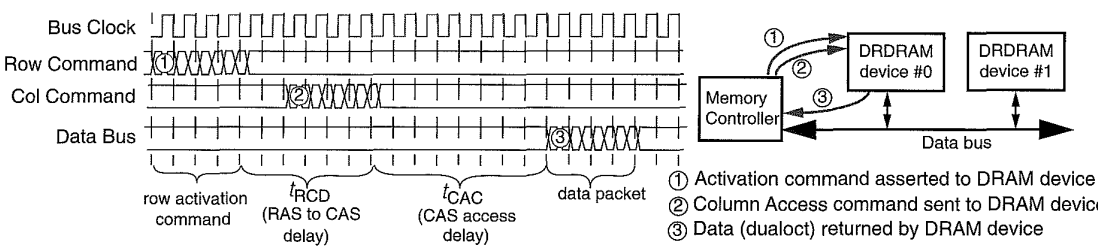


FIGURE 12.26: A row activation command followed by a column read command in a Direct RDRAM system.

[6]The command and address packets are not network packets with a header and payload. Rather, these packets are simply multi-cycle, fixed-duration, predefined format, command and address data sent by the controller to the DRAM device.
[7]The Direct RDRAM memory system sends and receives data, commands, and addresses on each edge of a clock signal. Consequently, in the Direct RDRAM memory system, the time period of 8 half cycles is equal to 4 full clock cycles.

issued by the memory controller consecutively without the need for idle cycles. Figure 12.27 illustrates the full utilization of the data bus by showing three consecutive column read commands that return data to the memory controller without the presence of any idle cycles between data packets on the data bus. Figure 12.27 shows two consecutive column read commands to device #0, followed by a third command sent to device #1. The figure then shows the return of data by the memory devices after the appropriate CAS delay. In theory, no idle cycles are needed in between column read commands, since the data transmission on the data bus is synchronized by the common *clock-to-master* signal, even in the case where the column read commands are sent to different Direct RDRAM devices in the Direct RDRAM channel.[8]

The Write-to-Read Turnaround Issue

Figure 12.28 illustrates the problematic case of a column read command that follows a column write command in commodity DRAM devices such as a

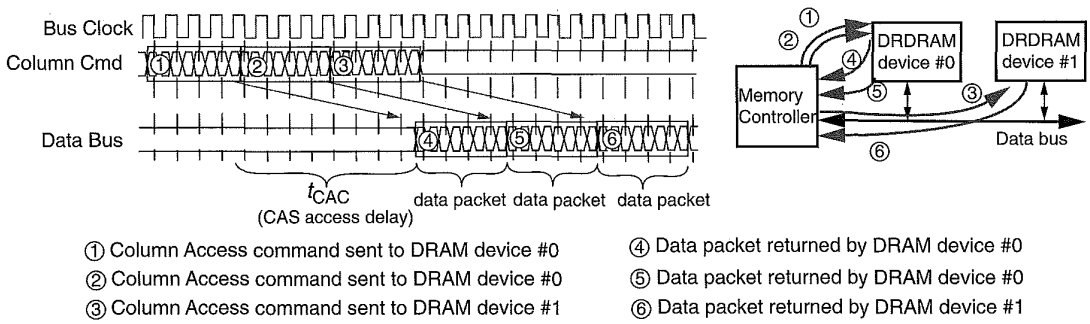


FIGURE 12.27: A row activation command followed by a column read command in a Direct RDRAM system.

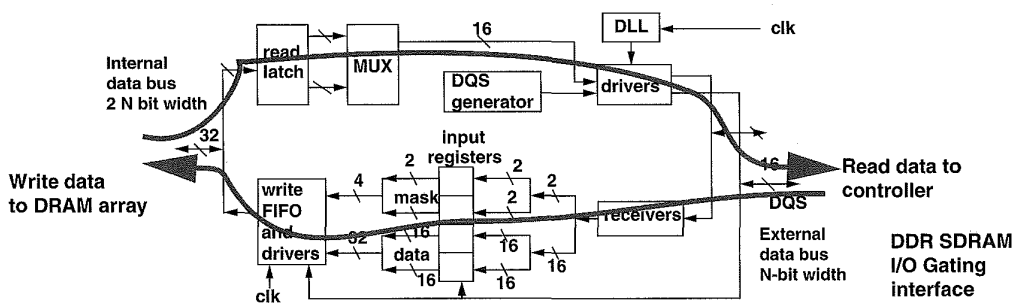


FIGURE 12.28: I/O gating sharing problem for read following write to same rank of DRAM devices—commodity DDRx SDRAM.

[8]In theory, although no idle cycles are needed in between any two column read commands to open pages in a Direct RDRAM memory system, idle cycles are sometimes inserted into Direct RDRAM memory systems between column access commands to reduce the activity rate of the Direct RDRAM device and to reduce the peak power dissipation of Direct RDRAM memory systems and Direct RDRAM devices.

DDR SDRAM device. The figure illustrates that despite the existence of separate read and write FIF buffers and drivers, both read data and write data must share the use of internal and external data busses through the I/O gating structure. The sharing of the data busses means that write data must be driven into and through the internal data busses to the selected bank of sense amplifiers before data for a subsequent read command to the same device can be placed onto the internal data bus. The bottleneck of the internal data bus leads directly to long write-to-read turnaround times in high data rate DRAM devices.

Write Buffer in Direct RDRAM Devices

To reduce the long write-read turnaround time in high data rate DRAM devices, Rambus designed Direct RDRAM devices and XDR DRAM devices with specialized structures to alleviate the write-to-read turnaround problem. The solution to the write-to-read turnaround problem implemented in the Direct RDRAM device is through the use of a write buffer. The existence of the write buffer in the Direct RDRAM device means that as soon as data is written into the write buffers, the I/O gating resource can be used by a subsequent column read command. Figure 12.29 illustrates the sequence of three column access commands to a Direct RDRAM device: a column write command followed by a column read command that is, in turn, followed by a retire command. Figure 12.29 illustrates that the write-to-read turnaround time is significantly reduced through the use of the write buffer. Essentially, actions typically performed by column write commands are separated into a write-into-write-buffer command and a retire-from-write-buffer-into-sense-amplifier command. The separation of the column write command enables a subsequent read command to be issued

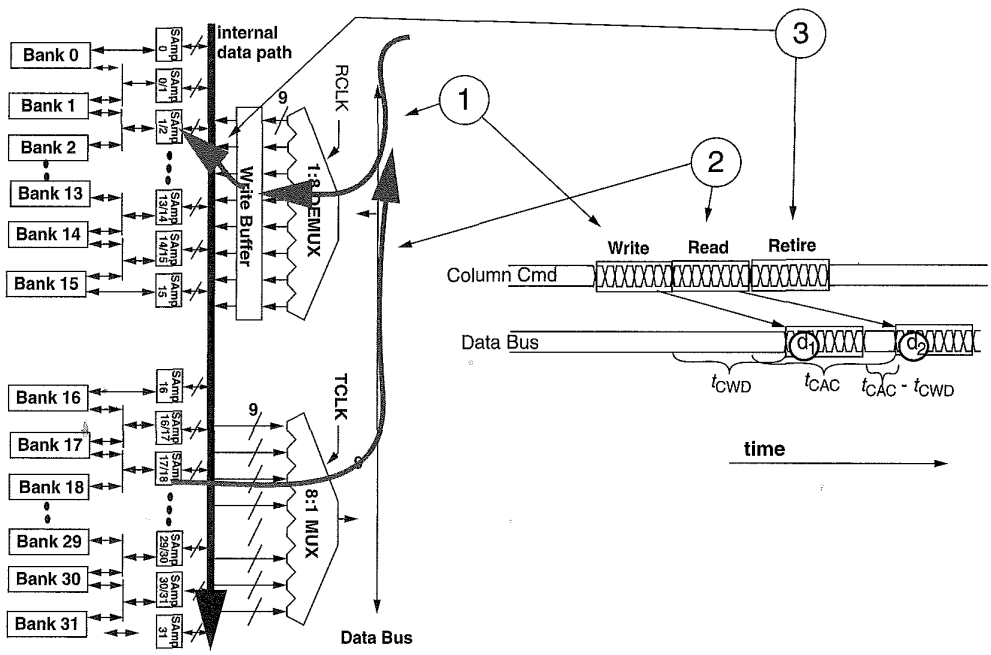


FIGURE 12.29: A write-to-read-to-retire command sequence in a Direct RDRAM device.

immediately after a column write command since the column write command does not have to immediately drive the data through the internal datapath and into the sense amplifiers.

The use of the write buffer by the Direct RDRAM device has several implications from the perspective of the memory-access protocol. First, a column write command places write data temporarily in the write buffer, but addresses of subsequent column read commands are not checked against addresses of pending data in the write buffer by the Direct RDRAM device. The interesting result is that a column read command that immediately follows a column write command to the exact same location in a Direct RDRAM device will read stale data from the sense amplifiers while the most recent data is held in the write buffers. The use of the write buffer by the Direct RDRAM device thus requires the memory controller to keep track of the addresses of column write commands that have not yet been retired from the write buffer to enforce memory consistency.

The use of write buffers to alleviate the write-to-read turnaround time directly increases the complexity of the DRAM device as well as the complexity of the DRAM memory controller. In the case of the DRAM device, die area devoted to the write buffer increases the die cost of the DRAM device. From the perspective of the memory-access protocol, the memory controller has to manage a new retire-write-data command, further increasing its complexity.

12.4.2 Technical and Pseudo-Technical Issues of Direct RDRAM

In the late 1990s, the Direct RDRAM memory system was chosen by Intel Corp. as the next-generation memory system that would replace the 100-MHz SDRAM memory system. However, due to a number of issues, Direct RDRAM memory systems failed to gain market acceptance and did not replace SDRAM as the mainstream memory system. Instead, a more moderate evolutionary path of DDRx SDRAM memory systems replaced the SDRAM memory system. The issues that prevented Direct RDRAM memory systems from gaining market acceptance consisted of a number of technical issues that were primarily engineering trade-offs and a number of non-technical issues that were related to the licensing of the Direct RDRAM memory system by Rambus Corp. to DRAM device manufacturers and system design houses. In this chapter, the focus is placed on the technical issues rather than the business decisions that impacted the failure of the Direct RDRAM memory system to achieve commodity status. Moreover, the challenges faced by Direct RDRAM memory systems in attempting a revolutionary break from the commodity SDRAM memory system are quite interesting from the perspective that future memory systems that attempt similar revolutionary breaks will have to overcome similar challenges faced by the Direct RDRAM memory system. Consequently, the engineering trade-offs that increased the cost of implementation or reduced the performance advantage of the Direct RDRAM memory systems are examined in detail.

Die Size Overhead

In practical terms, the Direct RDRAM memory system increased memory device complexity to obtain higher pin-bandwidth and increased data transport efficiency. For example, the baseline Direct RDRAM device contained 16 or 32 dependent banks, requiring separate bank control circuitry and arrays of sense amplifiers. Each Direct DRAM device also contained circuitry to carefully manage the electrical characteristics of the Direct RDRAM device. The high-speed RSL signaling interface further required separate I/O voltage rings for the Direct RDRAM device. Collectively, the sophisticated architectural features and electrical control circuitry added significantly to the die cost. Consequently, Direct RDRAM devices were approximately 20~30% larger than SDRAM devices at the 64-Mbit node. This die size overhead resulted from a combination of the required circuit overhead and the fact that these first-generation devices were designed for speed rather than die size. Fortunately, the circuit overhead of Direct RDRAM devices was relatively constant in terms of the number of transistors. Consequently, the die size overhead of Direct RDRAM devices, as a percentage of die area, decreases with increasing DRAM device capacity. At the 128-Mbit density node, Toshiba produced an SDRAM device with a die size of

91.7 mm^2, while its Direct RDRAM device on the same process had a die size of 103 mm^2, making the die size overhead 12% for Toshiba at the 128-Mbit node. The die size overhead for Direct RDRAM devices was expected to drop below 10% for the 256-Mbit density node and decrease further at higher density nodes.

Sophisticated System Engineering Requirement

The high system-level signaling rates of the Direct RDRAM memory system required careful control and understanding of the electrical characteristics of the memory controller I/O interface, the system board, the memory modules, and the DRAM device package and I/O interface. Analogously, the issues in implementing Direct RDRAM memory systems were similar to the issues that necessitated the PC100 standard, and the solutions were similar: significant amounts of engineering resources had to be devoted to design a high data rate memory system. Consequently, system manufacturers were reluctant to devote the engineering resources required to implement Direct RDRAM memory systems. In particular, some low-cost-focused system manufacturers lacked the engineering resources required to design low-cost Direct RDRAM memory systems.

One issue that illustrated the importance of engineering resources in the deployment of cost-effective Direct RDRAM memory systems is the issue of multilayer system boards. The Direct RDRAM memory system architecture required matching signal flight times for parallel signals on various command and data busses. To minimize crosstalk and ensure signal integrity, first-generation proof-of-concept system boards that implemented Direct RDRAM memory systems were designed on 8-layer PCBs, and many second-generation system boards that shipped commercially with Direct RDRAM memory systems were designed on 6-layer PCBs. Compared to commodity systems that used 4-layer PCBs, the 6-layer system board further increased cost to the early implementation of Direct RDRAM memory systems. Finally, with the passage of time and the devotion of engineering resources by Rambus and various system manufacturers, the system-level cost issue of Direct RDRAM memory systems was eventually brought to parity with commodity systems as Direct RDRAM memory systems on 4-layer PCBs were proven to be practical.

Advanced Packaging and Testing Equipment Requirement

The high signaling rate Direct RDRAM device required careful control of the electrical characteristics of the DRAM device package and I/O interface. Consequently, Direct RDRAM devices could not use similar low-cost TSOPs and SOJ packages that were used for SDRAM devices. Instead, Direct RDRAM devices were shipped with BGA packages that minimized the impedance contributions of the pin interface of the package. Figure 12.30 illustrates one type of BGA packaging used for Direct RDRAM devices. Unfortunately, the use of the BGA package further added cost to DRAM device and memory module manufacturers that were not accustomed to the new packaging. For example, memory module manufacturers could

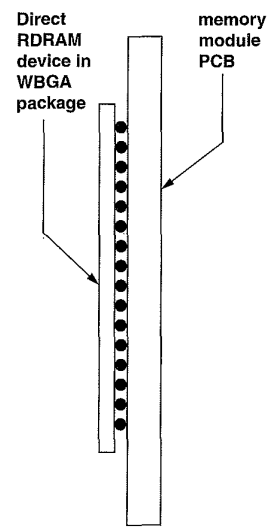


FIGURE 12.30: Edge view of a Wirebond Ball Grid Array (WBGA) Direct RDRAM device mounted on a memory module.

not visually inspect the solder connections between the BGA package and the memory module as they could with SOJ and TSOPs. Consequently, new equipment had to be purchased to handle the new packaging, further increasing the cost delta between Direct RDRAM memory systems and the then-commodity SDRAM and DDR SDRAM memory systems.

Heat Density—Heat Spreader—Enforced Idle Cycles

In the classical mesh topology of SDRAM and DDRx SDRAM memory systems, multiple DRAM devices are connected in parallel to form a given rank of memory. Moreover, high data rate DDR2 and DDR3 SDRAM devices do not contain enough banks in parallel in a single rank configuration to fully saturate the memory channel. Consequently, the on-chip and in-system data movements associated with any given command issued by the memory controller are always distributed across multiple DRAM devices in a single rank and also typically across different ranks in standard 64- or 72-bit-wide SDRAM and DDRx SDRAM memory systems. In contrast, the Direct RDRAM memory system is architected for high bandwidth throughput, and a single Direct RDRAM device provides full bandwidth for a given channel of Direct RDRAM devices. However, the ability of the single Direct RDAM device to provide full bandwidth to the memory channel means that the on-chip and in-system data movements associated with a given command issued by the memory controller are always limited to a single DRAM device. Moreover, Figure 12.31 illustrates that in the worst-case memory-access pattern, a sustainable stream of row activation and column access commands can be pipelined to a single device in a given channel of the Direct RDRAM memory system. Consequently, localized hot spots associated with high access rates to a given device can appear and disappear on different sections of the Direct RDRAM channel. The localized hot spots can, in turn, change the electrical characteristics of the transmission lines that Direct RDRAM memory systems rely on to deliver command and data packets, thus threatening the functional correctness of the memory system itself. To counter the problem of localized hot spots in Direct RDRAM memory systems, Rambus Corp. deployed two solutions: heat spreaders and new command issue rules designed to limit access rates to a given Direct RDRAM device. Unfortunately, the use of heat spreaders on the RIMMs further increased the cost of the Direct RDRAM memory system, and the new command issue rules further increased controller complexity and decreased available memory bandwidth in the Direct RDRAM memory system.

Low Request Rate Systems

The Direct RDRAM memory system provided a revolutionary approach to memory system architecture that required significant cost adders in many different components of the memory system. Consequently, first-generation Direct RDRAM memory systems were significantly more expensive than SDRAM memory systems. However, these first-generation Direct RDRAM memory systems were used in Pentium III-based computer systems, and these Direct

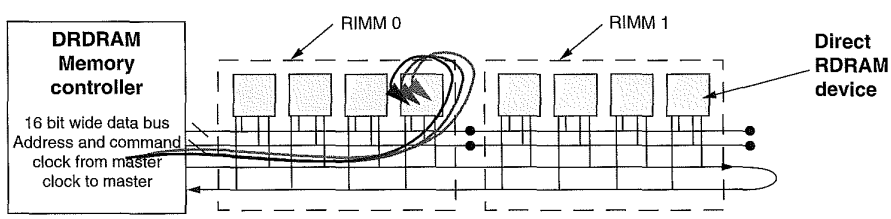


FIGURE 12.31: Worst-case memory-access pattern can create localized hot spots in DRDRAM system topology.

RDRAM memory systems did not illustrate significant performance benefits over SDRAM-based memory systems to justify their cost premium.

Figure 12.32 shows that in the memory-access sequence of a Direct RDRAM memory system, the controller first sends a packet command to the Direct RDRAM device, and then the DRAM device buffers the command, decodes it, moves the data to the I/O interface, and then bursts that data over 8 half cycles back to the controller. Memory access in the Direct RDRAM memory system thus suffers from the additional latency components of command transmission to the DRAM device and data burst time back to the controller. Consequently, the SDRAM memory system has lower idle system latency as compared to the Direct RDRAM memory system. Figure 12.32 illustrates that the advantage of the Direct RDRAM memory system is that its higher bandwidth and higher efficiency allows it to return data with lower average latency when it is coupled with a processor that has a high rate of memory access. Unfortunately, first-generation Direct RDRAM memory systems were primarily coupled to uniprocessor Pentium III-based systems that did not and could not sustain a high rate of memory access. Potentially, Direct RDRAM memory systems coupled to advanced high-frequency, multi-threaded, and multi-core processors would have provided the tangible performance benefit that could justify the cost premium. Unfortunately, the average consumer could not envision the advanced processors that were yet to come, but could readily observe that the Direct RDRAM memory system did not present significant performance advantage over SDRAM memory systems when attached to Pentium III-based systems.

Different Devices for Different Systems

In a Direct RDRAM memory system, a single Direct RDRAM device returns a dualoct for a given column access command. However, memory systems designed for reliability require additional data storage locations to store check bits. In traditional commodity memory systems, the check bits are stored in additional DRAM devices as the width of the data bus is increased from 64 to 72 bits. Consequently, the same DRAM devices can be used in desktop computers that do not require error correction and in high-end servers that do. In contrast, the 16-bit data bus width of the Direct RDRAM memory system means that it was not practical to create an analogous memory system where a single logical channel of memory consists of 9 physical channels of Direct RDRAM devices in parallel. Aside from the extraordinary cost of 9 physical channels of Direct RDRAM memory, a single column access command to such a memory system would move 144 bytes of data, a granularity that is too large

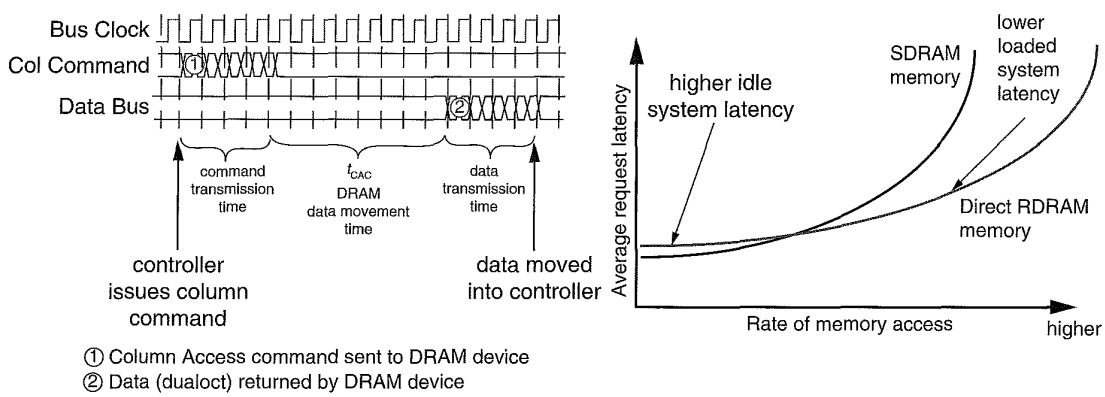


FIGURE 12.32: Longer idle system latency and lower loaded system latency for a Direct RDRAM system.

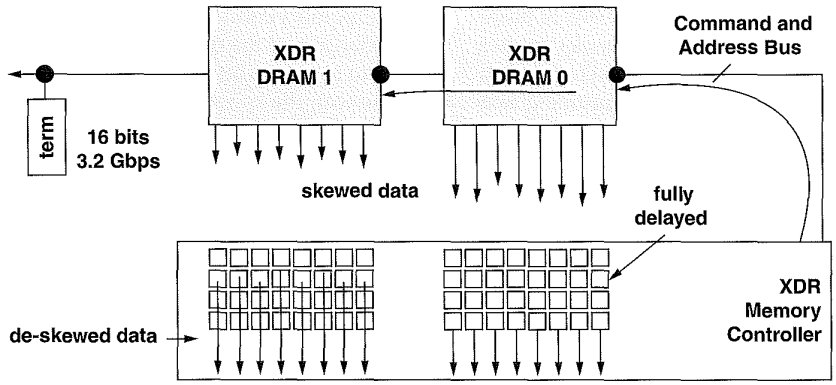


FIGURE 12.33: Basic topology of the XDR memory system.

for the cacheline sizes of all mainstream processors.[9] Consequently, different versions of Direct RDRAM devices had to be designed for error correcting memory systems and systems that did not require error correction. At the 64-Mbit density node, 64-Mbit Direct RDRAM devices with 16-bit-wide data busses were available alongside 72-Mbit Direct RDRAM devices with 18-bit-wide data busses. Similarly, 144-, 288-, and 576-Mbit Direct RDRAM devices were available alongside 128-, 256-, and 512-Mbit devices in each density node, respectively. The different versions of Direct RDRAM devices split the volume demand for Direct RDRAM devices, providing yet another obstacle that hindered cost-reduction efforts in Direct RDRAM memory systems.

Lessons for Future Memory Systems

The Direct RDRAM memory system introduced a high level of complexity and new cost adders into the memory system, but the fact that it was coupled to a relatively low-frequency Pentium III-based system meant that it did not demonstrate its best-case performance advantage over lower cost SDRAM and DDR SDRAM memory systems. Consequently, it met a great deal of resistance and encountered numerous obstacles in its way of reaching high volume production status, approaching price parity with SDRAM memory systems and demonstrating a tangible performance advantage in production systems. Ultimately, the Direct RDRAM memory system failed to reach commodity status. However, the technical challenges faced by Direct RDRAM memory systems were important lessons that Rambus Corp. heeded in the design of its next-generation high bandwidth memory system, the XDR memory system.

12.4.3 XDR Memory System

Rambus Corp., having learned valuable lessons from the drawbacks of the Direct RDRAM memory system, proceeded to design its next-generation high bandwidth memory system that avoided many of the technical pitfalls that hindered acceptance of the Direct RDRAM memory system. The *Extreme Data Rate* (XDR) memory system has been designed to further increase DRAM device and memory system signaling rate, yet at the same time to reduce system design complexity and DRAM device overhead.

Topology

Figure 12.33 illustrates the basic system topology of the XDR memory system. The figure shows

[9]Intel's Itanium and IBM's server-oriented POWER processors do have 128-byte cachelines, but neither system seriously contemplated such a memory system.

that in the XDR memory system, as in DDRx SDRAM systems, the command-and-address busses operates at a relatively lower data rate as compared to the data bus. In the XDR memory system, the command and address datapath is a path-length-matched, unidirectional datapath that the XDR memory controller uses to broadcast command and address information to DRAM devices. Figure 12.33 also shows that the XDR controller uses FlexPhase to do skew compensation on the datapath—removes skew on read data coming into the controller and adds skew to ensure that write data reaches DRAM device interface in sync.

Device Architecture

Figure 12.34 illustrates the device organization of an XDR device. The XDR DRAM device is characterized by numerous features that distinguish it from the Direct RDRAM device. For example, the XDR DRAM device has a prefetch length of 16, so the minimum burst length is 16. However, the XDR device architecture has been designed with variable device data bus width control so that a device may be configured with a data bus width as low as 1 bit. Consequently, the minimum granularity of data movement in such a device may be as small as 16 bits. Figure 12.34 also shows that the XDR device is internally organized into two different, odd and even, bank sets.

Finally, Figure 12.34 shows that the XDR device differs from the Direct RDRAM device in that unlike the Direct RDRAM device, the XDR DRAM device has a common command bus that is used by row access commands as well as column access commands. Moreover, unlike the Direct RDRAM device, the XDR DRAM device does not have a write buffer. Instead, it relies on the differing bank sets and an intelligent controller to alleviate the write-to-read data bus turnaround issue.

Signaling

Figure 12.35 shows the octal data rate signaling system that Rambus Corp. introduced with the XDR memory system. In this signaling system, a relatively slow master clock signal (400~800 MHz) is shared by both the memory controller and the DRAM device. The DRAM device and the memory controller use current mirrors to transmit differential signals that are locked in-phase with the system clock signal, but operate at four times the frequency of the system clock signal. Data is transmitted on both edges of the signal, so 8 1-bit symbols per pin pair are transmitted in every clock cycle. The low voltage swing, point-to-point, differential signaling enables high bandwidth in commodity ASIC and DRAM processor technologies at the cost of higher power consumption.

FlexPhase

In the XDR memory system, Rambus Corp. uses a system of bit-adjustable DLL circuitry to remove bit-to-bit signal skew from the high-speed parallel data bus. This system of bit-adjustable DLL circuitry is referred to as FlexPhase. Figure 12.35 shows how Rambus uses the FlexPhase circuit to account for differences in signal flight time. FlexPhase takes care of data-to-data skew, and it can be recalibrated to lock in new phase differentials to account for thermal drifts between cold and warm systems. Figure 12.35 shows that the FlexPhase circuitry is placed in the controller interface. In this manner, the FlexPhase technology removes the path-length matching requirement from the XDR memory system without increasing the cost of the DRAM device. In this case, the additional cost is paid for in terms of increased controller sophistication.

XDR Early Read After Write

To counter the various drawbacks of a write buffer in Direct RDRAM devices, Rambus did not include write buffers in the design of its next-generation, high-performance XDR DRAM device. Instead, XDR DRAM devices are architected to support a feature that Rambus refers to as the Early Read After Write (ERAW) feature. Essentially, XDR devices avoid the large write-to-read command overhead by using separate internal paths for banks separated into odd and even sets. In this organization, a read command can proceed in parallel with a write command, as long as the read and write commands are directed to banks in

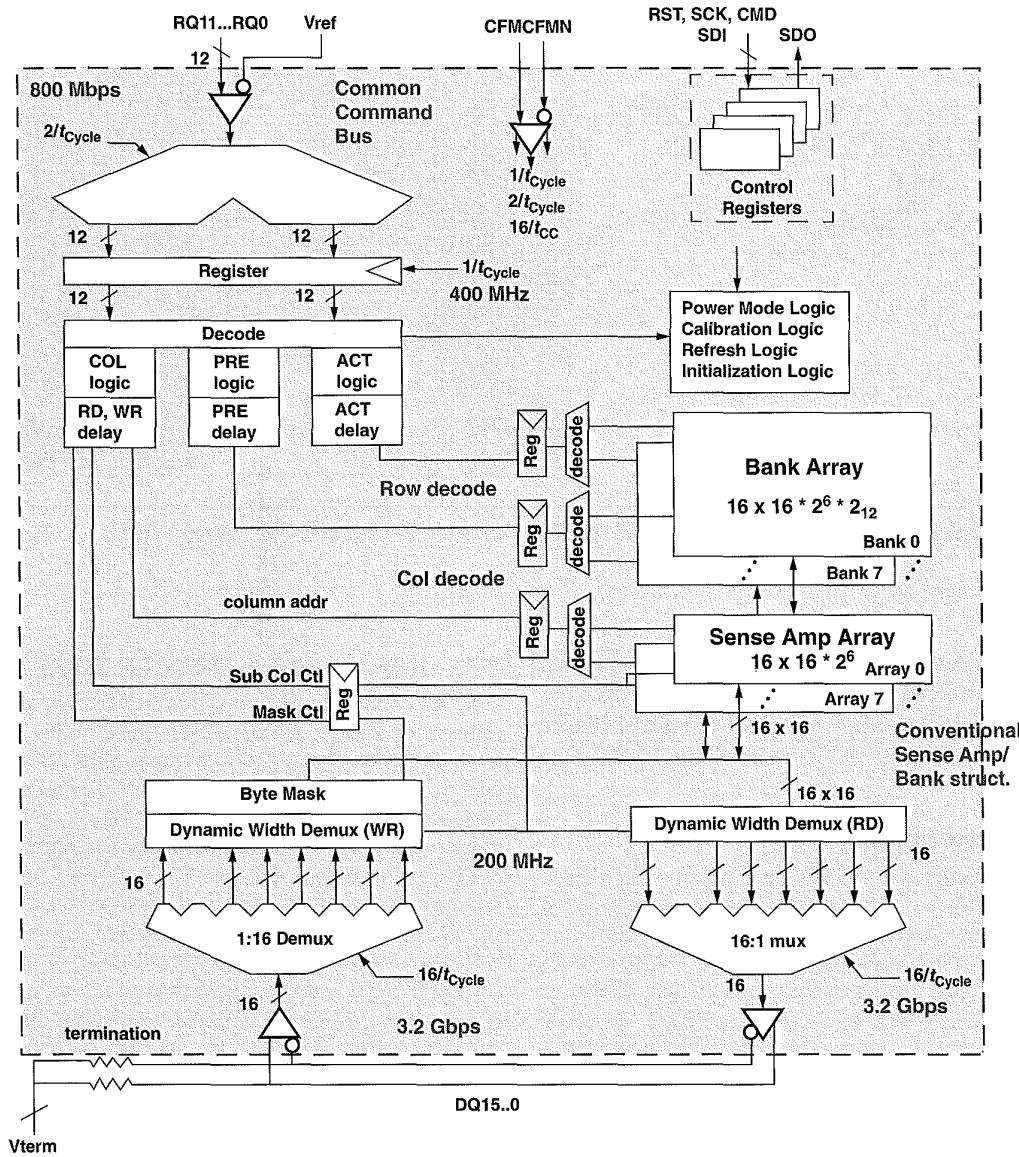


FIGURE 12.34: Architecture of an XDR device.

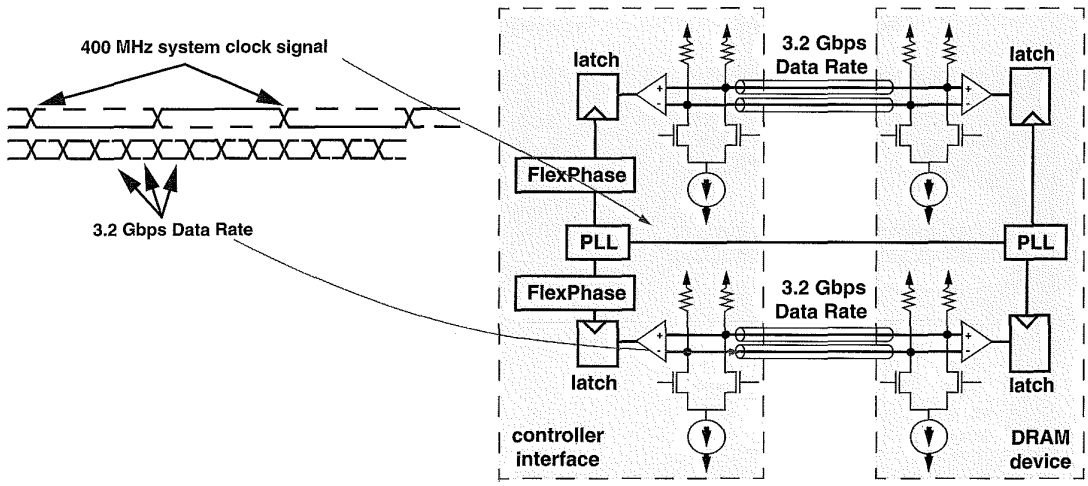


FIGURE 12.35: Octal data rate signaling using per-bit FlexPhase de-skewing and PLL in an XDR memory system.

different bank sets. Figure 12.36 shows that although the write-to-read turnaround overhead still occupies several cycles, the overhead is much smaller than if the write and read commands are issued to the same set of banks.

12.5 Low Latency

The proliferation of low-cost commodity DRAM devices means that high-speed, low-latency SRAM devices have been pushed into increasingly small niches as design engineers seek to use DRAM devices wherever they can to reduce system cost. However, even as DRAM devices replace SRAM in more and more applications, the desire for lower-than-commodity-DRAM latency characteristics remains. Consequently, DRAM device manufacturers have been developing specialty low-latency DRAM devices as semi-commodity, moderately low-latency replacements for SRAM replacement application. Two families of DRAM devices, the Reduced Latency DRAM (RLDRAM) and Fast Cycle DRAM (FCRAM), have been designed specifically to target the low-latency SRAM replacement market.

12.5.1 Reduced Latency DRAM (RLDRAM)

RLDRAM is a low-latency DRAM device with random access latency as low as 10–12 ns and row cycle times as low as 20 ns. The first-generation RLDRAM device was designed by Infineon to meet the demands of the market for networking equipment. Subsequently, Micron was brought in as a partner to co-develop RLDRAM and provide a second source. However, Infineon has since abandoned RLDRAM development, and Micron proceeded ahead to develop the second-generation RLDRAM II. RLDRAM and RLDRAM II were designed for use in embedded applications. Primarily, RLDRAM is designed for the network market. However, it may also be used for other applications where cost is less of a concern, low latency is a highly desirable feature, and smaller-than-commodity-DRAM capacity is not an issue. For example, RLDRAM may be used as a large lookup table or a large off-chip cache. RLDRAM has been designed with an SRAM-like mentality in that it is designed for SRAM-like random access to any part of the DRAM device; row address and column addresses are combined together and sent as a single access command.

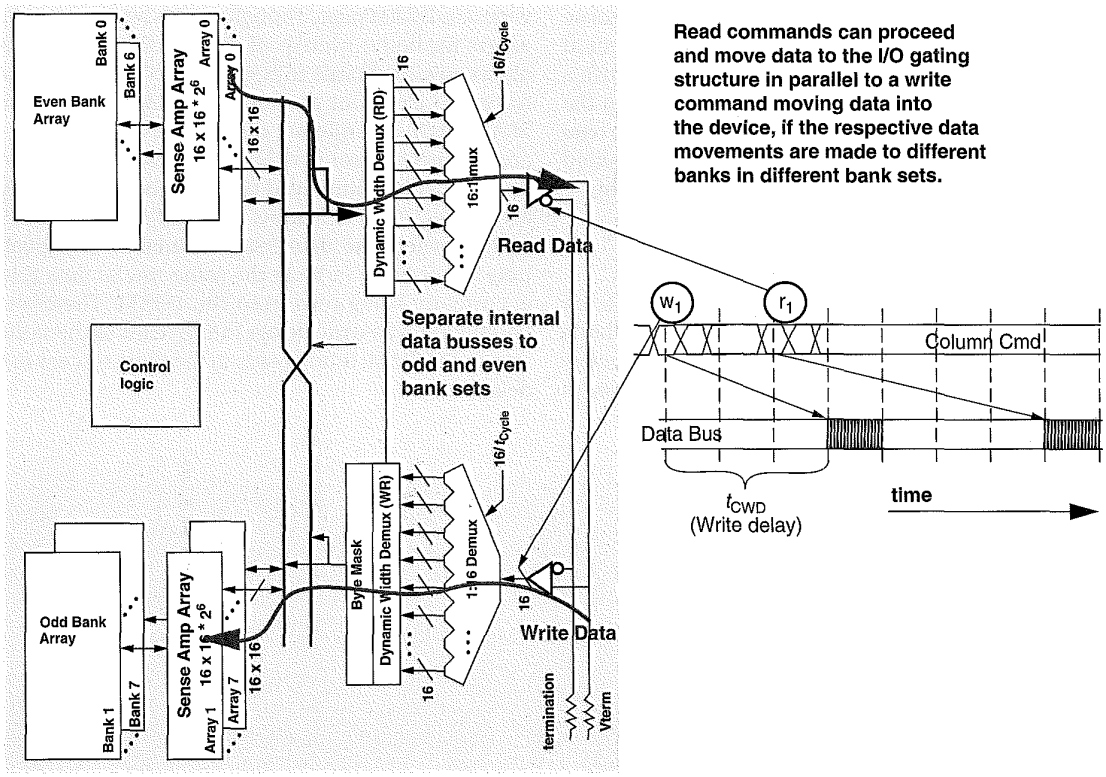


FIGURE 12.36: Read following write in an XDR DRAM device that supports ERAW.

12.5.2 Fast Cycle DRAM (FCRAM)

In contrast to RLDRAM, Fujitsu and Toshiba's FCRAM achieves low-latency data access by segmenting the data array into multiple subarrays, only one of which is driven during a row-activation command. The sub-row activation process effectively decreases the size of an array and improves access time. Unlike RLDRAM, FCRAM was designed with a DDRx SDRAM-like mentality, where row access commands and column access commands are sent separately, and the FCRAM command set was designed specifically to mimic the DDRx SDRAM command set. The purpose of the commonality is to enable controller designs that can use either FCRAM or commodity DDRx SDRAM devices. Finally, differing from RLDRAM devices, FCRAM was designed with a DIMM specification.

12.6 Interesting Alternatives

12.6.1 Virtual Channel Memory (VCDRAM)

Virtual channel adds a substantial SRAM cache to the DRAM that is used to buffer large blocks of data (called *segments*) that might be needed in the future. The SRAM segment cache is managed explicitly by the memory controller. The design adds a new step in the DRAM-access protocol: a row activate operation moves a page of data into the sense amplifiers; "prefetch" and "restore" operations (data-read and

data-write, respectively) move data between the sense amps and the SRAM segment cache one segment at a time; and column read or write operations move a column of data between the segment cache and the output buffers. The extra step adds latency to read and write operations, unless all of the data required by the application fits in the SRAM segment cache.

12.6.2 Enhanced SDRAM (ESDRAM)

Like EDO DRAM, ESDRAM adds an SRAM latch to the DRAM core, but whereas EDO added the latch after the column multiplexor, ESDRAM adds it *before* the column multiplexor. Therefore, the latch is as wide as a DRAM page. Though expensive, the scheme allows for better overlap of activity. For instance, it allows row precharge to begin immediately without having to close out the row (it is still active in the SRAM latch). In addition, the scheme allows a write-around mechanism whereby an incoming write can proceed without the need to close out the currently active row. Such a feature is useful for write-back caches, where the data being written at any given time is not likely to be in the same row as data that is currently being read from the DRAM. Therefore, handling such a write delays future reads to the same row. In ESDRAM, future reads to the same row are not delayed.

CHAPTER 13

DRAM Memory Controller

In modern computer systems, processors and I/O devices access data in the memory system through the use of one or more memory controllers. Memory controllers manage the movement of data into and out of DRAM devices while ensuring protocol compliance, accounting for DRAM-device-specific electrical characteristics, timing characteristics, and, depending on the specific system, even error detection and correction. DRAM memory controllers are often contained as part of the system controller, and the design of an optimal memory controller must consist of system-level considerations that ensure fairness in arbitration for access between different agents that read and store data in the same memory system.

The design and implementation of the DRAM memory controllers determine the access latency and bandwidth efficiency characteristics of the DRAM memory system. The previous chapters provide a bottom-up approach to the design and implementation of a DRAM memory system. With the understanding of DRAM device operations and system level provided by the previous chapters, this chapter proceeds to examine DRAM controller design and implementation considerations.

13.1 DRAM Controller Architecture

The function of a DRAM memory controller is to manage the flow of data into and out of DRAM devices connected to that DRAM controller in the memory system. However, due to the complexity of DRAM memory-access protocols, the large numbers of timing parameters, the innumerable combinations of memory system organizations, different workload characteristics, and different design goals, the design space of a DRAM memory controller for a given DRAM device has nearly as much freedom in the design space as the design space of a processor that implements a specific instruction-set architecture. In that sense, just as an instruction-set architecture defines the programming model of a processor, a DRAM-access protocol defines the interface protocol between a DRAM memory controller and the system of DRAM devices. In both cases, actual performance characteristics depend on the specific microarchitectural implementations rather than the superficial description of a programming model or interface protocol. That is, just as two processors that support the same instruction-set architecture can have dramatically different performance characteristics depending on the respective microarchitectural implementations, two DRAM memory controllers that support the same DRAM-access protocol can have dramatically different latency and sustainable bandwidth characteristics depending on the respective microarchitectural implementations. DRAM memory controllers can be designed to minimize die size, minimize power consumption, maximize system performance, or simply reach a reasonably optimal compromise of the conflicting design goals. Specifically, the *Row-Buffer-Management Policy,* the *Address Mapping Scheme,* and the *Memory Transaction and DRAM Command Ordering Scheme* are particularly important to the design and implementation of DRAM memory controllers.

Due to the increasing disparity in the operating frequency of modern processors and the access latency to main memory, there is a large body of active and ongoing research in the architectural community

devoted to the performance optimization of the DRAM memory controller. Specifically, the *Address Mapping Scheme,* designed to minimize bank address conflicts, has been studied by Lin et al. [2001] and Zhang et al. [2002a]. *DRAM Command and Memory Transaction Ordering Schemes* have been studied by Briggs et al. [2002], Cuppu et al. [1999], Hur and Lin [2004], McKee et al. [1996a], and Rixner et al. [2000]. Due to the sheer volume of research into optimal DRAM controller designs for different types of DRAM memory systems and workload characteristics, this chapter is not intended as a comprehensive summary of all prior work. Rather, the text in this chapter describes the basic concepts of DRAM memory controller design in abstraction, and relevant research on specific topics is referenced as needed.

Figure 13.1 illustrates some basic components of an abstract DRAM memory controller. The memory controller accepts requests from one or more microprocessors and one or more I/O devices and provides the arbitration interface to determine which request agent will be able to place its request into the memory controller. From a certain perspective, the request arbitration logic may be considered as part of the system controller rather than the memory controller. However, as the cost of memory access continues to increase relative to the cost of data computation in modern processors, efforts in performance optimizations are combining transaction scheduling and command scheduling policies and examining them in a collective context rather than separate optimizations. For example, a low-priority request from an I/O device to an already open bank may be scheduled ahead of a high-priority request from a microprocessor to a different row of the same open bank, depending on the access history, respective priority, and state of the memory system. Consequently, a discussion on transaction arbitration is included in this chapter.

Figure 13.1 also illustrates that once a transaction wins arbitration and enters into the memory controller, it is mapped to a memory address location and converted to a sequence of DRAM commands. The sequence of commands is placed in queues that exist in the memory controller. The queues may be arranged as a generic queue pool, where the controller will select from pending commands to execute, or the queues may be arranged so that there is one queue per bank or per rank of memory. Then, depending on the DRAM command scheduling policy, commands are scheduled to the DRAM devices through the electrical signaling interface.

In the following sections, the various components of the memory controller illustrated in Figure 13.1 are separately examined, with the exception of the electrical signaling interface. Although the electrical signaling interface may be one of the most critical

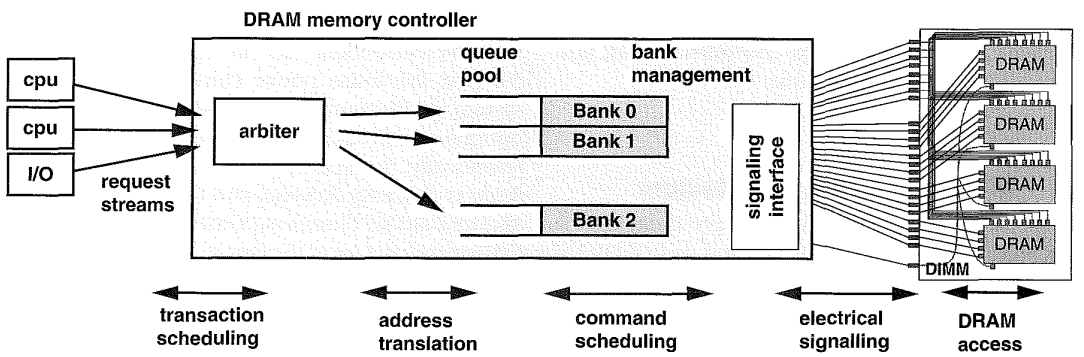


FIGURE 13.1: Illustration of an abstract DRAM memory controller.

components in modern, high data rate memory systems, the challenges of signaling are examined separately in Chapter 9. Consequently, the focus in this chapter is limited to the digital logic components of the DRAM memory controller.

13.2 Row-Buffer-Management Policy

In modern DRAM devices, the arrays of sense amplifiers can also act as buffers that provide temporary data storage. In this chapter, policies that manage the operation of sense amplifiers are referred to as *row-buffer-management policies*. The two primary row-buffer-management policies are the *open-page* policy and the *close-page* policy, and depending on the system, different row-buffer-management policies can be used to optimize performance or minimize power consumption of the DRAM memory system.

13.2.1 Open-Page Row-Buffer-Management Policy

In commodity DRAM devices, data access to and from the DRAM storage cells is a two-step process that requires separate row activation commands and column access commands.[1] In cases where the memory-access sequence possesses a high degree of temporal and spatial locality, memory system architects and design engineers can take advantage of the locality by directing temporally and spatially adjacent memory accesses to the same row of memory. The *open-page* row-buffer-management policy is designed to favor memory accesses to the same row of memory by keeping sense amplifiers open and holding a row of data for ready access. In a DRAM controller that implements the *open-page* policy, once a row of data is brought to the array of sense amplifiers in a bank of DRAM cells, different columns of the same row can be accessed again with the minimal latency of t_{CAS}. In the case where another memory read access is made to the same row, that memory access can occur with minimal latency since the row is already active in the sense amplifier and only a column access command is needed to move the data from the sense amplifiers to the memory controller. However, in the case where the access is to a different row of the same bank, the memory controller must first precharge the DRAM array, engage another row activation, and then perform the column access.

13.2.2 Close-Page Row-Buffer-Management Policy

In contrast to the open-page row-buffer-management policy, the *close-page* row-buffer-management policy is designed to favor accesses to random locations in memory and optimally supports memory request patterns with low degrees of access locality. The open-page policy and closely related variant policies are typically deployed in memory systems designed for low processor count, general-purpose computers. In contrast, the close-page policy is typically deployed in memory systems designed for large processor count, multiprocessor systems or specialty embedded systems. The reason that an open-page policy is typically deployed in memory systems of low processor count platforms while a close-page policy is typically deployed in memory systems of larger processor count platforms is that in large systems, the intermixing of memory request sequences from multiple, concurrent, threaded contexts reduces the locality of the resulting memory-access sequence. Consequently, the probability of row hit decreases and the probability of bank conflict increases in these systems, reaching a tipping point of sorts where a close-page policy provides better performance for the computer system. However, not all large processor

[1] In some DRAM devices that strive to be SRAM-like, the row-activation command and the column-access command are coupled into a single read or write command. These devices do not support the open-page row-buffer-management policies. Typically, these DRAM devices are designed as low-latency, random-access memory and used in speciality embedded systems.

count systems use close-page memory systems. For example, Alpha EV7's Direct RDRAM memory system uses open-page policy to manage the sense amplifiers in the DRAM devices. The reason for this choice is that a fully loaded Direct RDRAM memory system has 32 ranks of DRAM devices per channel and 32 split banks per rank. The large number of banks in the Direct RDRAM memory system means that even in the case where a large number of concurrent processes are accessing memory from the same memory system, the probability of bank conflicts remains low. Consequently, Alpha EV7's memory system demonstrates that the optimality in the choice of row-buffer-management policies depends on both the type and the number of the processor, as well as the parallelism available in the memory system.

13.2.3 Hybrid (Dynamic) Row-Buffer-Management Policies

In modern DRAM memory controllers, the row-buffer-management policy is often neither a strictly open-page policy nor a strictly close-page policy, but a dynamic combination of the two policies. That is, the respective analyses of the performance and power consumption impact of row-buffer-management policies illustrate that the optimality of the row-buffer-management policy depends on the request rate and access locality of the memory request sequences. To support memory request sequences whose request rate and access locality can change dramatically depending on the dynamic, run-time behavior of the workload, DRAM memory controllers designed for general-purpose computing can utilize a combination of access history and timers to dynamically control the row-buffer-management policy for performance optimization or power consumption minimization.

Previously, the minimum ratio of memory read requests that must be row buffer hits for an open-page memory system to have lower read latency than a comparable close-page memory system was computed as $t_{RP} / (t_{RCD} + t_{RP})$. The minimum ratio of row buffer hits means that if a sequence of bank conflicts occurs in rapid succession and the ratio of memory read requests that are row buffer hits falls below a precomputed threshold, the DRAM controller can switch to a close-page policy for better performance. Similarly, if a rapid succession of memory requests to a given bank is made to the same row, the DRAM controller can switch to an open-page policy to improve performance. One simple mechanism used in modern DRAM controllers to improve performance and reduce power consumption is the use of a timer to control the sense amplifiers. That is, a timer is set to a predetermined value when a row is activated. The timer counts down with every clock tick, and when it reaches zero, a precharge command is issued to precharge the bank. In case of a row buffer hit to an open bank, the counter is reset to a higher value and the countdown repeats. In this manner, temporal and spatial locality present in a given memory-access sequence can be utilized without keeping rows open indefinitely.

Finally, row-buffer-management policies can be controlled on a channel-by-channel basis or on a bank-by-bank basis. However, the potential gains in performance and power savings must be traded off against the increase in hardware sophistication and design complexity of the memory controller. In cases where high performance or minimum power consumption is not required, a basic controller can be implemented to minimize die size impact of the DRAM memory controller.

The definition of the row-buffer-management policy forms the foundation in the design of a DRAM memory controller. The choice of the row-buffer-management policy directly impacts the design of the address mapping scheme, the memory command reordering mechanism, and the transaction reordering mechanism in DRAM memory controllers. In the following sections, the address mapping scheme, the memory command reordering mechanism, and the transaction reordering mechanism are explored in the context of the row-buffer-management policy used.

13.2.4 Performance Impact of Row-Buffer-Management Policies

A formal analysis that compares the performance of row-buffer-management policies requires an in-depth analysis of system-level queuing delays, the locality

TABLE 13.1 Current specification for 16 256-Mbit Direct RDRAM devices in 32-bit RIMM modules

Condition Specification	Current
One RDRAM device per channel in Read, balance in NAP mode	1195 mA
One RDRAM device per channel in Read, balance in standby mode	2548 mA
One RDRAM device per channel in Read, balance in active mode	3206 mA

and rate of request arrival in the memory-access sequences. However, a first-order approximation of the performance benefits and trade-offs of different policies can be made through the analysis of memory read access latencies. Assuming nominally idle systems, the read latency in a close-page memory system is simply $t_{RCD} + t_{CAS}$. Comparably, the read latency in an open-page memory system is as little as t_{CAS} or as much as $t_{RP} + t_{RCD} + t_{CAS}$.[2] In this context, t_{CAS} is the row buffer hit latency, and $t_{RP} + t_{RCD} + t_{CAS}$ is the row buffer miss (bank conflict) latency. If x represents the percentage of memory accesses that hit in an open row buffer, and $1 - x$ represents the percentage of memory accesses that miss the row buffer, the average DRAM-access latency in an open-page memory system is $x * (t_{CAS}) + (1 - x) * (t_{RP} + t_{RCD} + t_{CAS})$. Taking the formula and equating to the memory read latency of $t_{RCD} + t_{CAS}$ in a close-page memory system, the minimum percentage of memory accesses that must be row buffer hits for an open-page memory system to have lower average memory-access latency than a close-page memory system can be solved for.

Solving for x, the minimum ratio of memory read requests that must be row buffer hits for an open-page memory system to have lower read latency is simply $t_{RP}/(t_{RCD} + t_{RP})$. That is, as t_{RP} approaches infinity,[3] the percentage of row buffer hits in an open-page memory system must be nearly 100% for the open-page memory system to have lower (idle system) memory read latency than a comparable close-page memory system. Alternatively, as the t_{RP} approaches zero,[4] an open-page system will have lower DRAM memory-access latency for any non-zero percentage of row buffer hits. Given specific values for t_{RCD} and t_{RP}, specific requirements of row buffer hit versus row buffer miss ratio can be computed, and the resulting ratio can be used to aid in design decisions of a row-buffer-management policy for a DRAM memory system.

13.2.5 Power Impact of Row-Buffer-Management Policies

The previous section presents a simple mathematical exercise that compares idle system read latencies of an abstract open-page memory system to a comparable close-page memory system. In reality, the choice of the row-buffer-management policy in the design of a DRAM memory controller is a complex and multifaceted issue. A second factor that can influence the selection of the row-buffer-management policy may be the power consumption of DRAM devices.[5] Table 13.1 illustrates the operating current ratings for a Direct RDRAM memory system that contains a total of 16 256-Mbit Direct RDRAM devices. The act of keeping the DRAM banks active and the DRAM device

[2]That is, assuming that the probability of having to wait for t_{RAS} of the previous row activation is equal in open-page and close-page memory systems. If the probability of occurrence is the same, then the latency overheads are the same and can be ignored.
[3]Or increase to a significantly higher multiple of t_{RCD}.
[4]Or decrease to a small fraction of t_{RCD}.
[5]Commodity DRAM devices such as DDR2 SDRAM devices consume approximately the same amount of power in active standby mode as it does in precharge standby mode, so power optimality of the paging policy is device-dependent.

in active standby mode requires a moderate amount of current draw in Direct RDRAM devices. Table 13.1 illustrates that a lower level of power consumption in Direct RDRAM devices can be achieved by keeping all the banks inactive and the DRAM device in a power-down NAP mode.

The power consumption characteristics of different DRAM device operating modes dictate that in cases where power consumption minimization is important, the optimality of the row-buffer-management policy can also depend on the memory request rate. That is, the close-page row-buffer-management policy is unambiguously better for memory request sequences with low access locality, but it is also better for power-sensitive memory systems designed for request sequences with relatively low request rates. In the power-sensitive memory systems, Table 13.1 shows that for Direct RDRAM devices, it may be better to pay the cost of the row activation and precharge current for each column access than it is to keep the rows active for an indefinite amount of time waiting for more column accesses to the same open row.

13.3 Address Mapping (Translation)

Many factors can impact the latency and sustainable bandwidth characteristics of a DRAM memory system. Aside from the row-buffer-management policy, one factor that can directly impact DRAM memory system performance is the address mapping scheme. In this text, the address mapping scheme is used to denote the scheme whereby a given physical address is resolved into indices in a DRAM memory system in terms of channel ID, rank ID, bank ID, row ID, and column ID. The task of address mapping is also sometimes referred to as address translation.

In a case where the run-time behavior of the application is poorly matched with the address mapping scheme of the DRAM memory system, consecutive memory requests in the memory request sequence may be mapped to different rows of the same bank of DRAM array, resulting in bank conflicts that degrade performance. On the other hand, an address mapping scheme that is better suited to the locality property of the same series of consecutive memory requests can map them to different rows of different banks, where accesses to different banks can occur with some degree of parallelism. Fundamentally, the task of an address mapping scheme is to minimize the probability of bank conflicts in temporally adjacent requests and maximize the parallelism in the memory system. To obtain the best performance, the choice of the address mapping scheme is often coupled to the row-buffer-management policy of the memory controller. However, unlike hybrid row-buffer-management policies that can be dynamically adjusted to support different types of memory request sequences with different request rates and access locality, address mapping schemes cannot be dynamically adjusted in conventional DRAM memory controllers.

Figure 13.2 illustrates and compares the conventional system architecture against a novel system architecture proposed by the Impulse memory controller research project. The figure shows that in the conventional system architecture, the processor operates in the virtual address space, and the TLB maps application addresses in the virtual address space to the physical address space without knowledge or regard to the mapping scheme in the DRAM memory address. The Impulse memory controller from the University of Utah proposes a technique that allows a system to utilize part of the address space as *shadow addresses*. The shadow addressing scheme utilizes a novel virtual-to-physical address translation scheme and, in cooperation with the intelligent memory controller, eliminates bank conflicts between sequences of streaming requests by dynamically remapping address locations in the address space.

Essentially, the Impulse memory controller assumes the system architecture where the memory controller is tightly integrated with the TLB. With full understanding of the organization of the DRAM memory system, the Impulse memory controller is better able to minimize bank conflicts in mapping application request sequences from the virtual address space to the DRAM address space.

However, Impulse memory controller research is based on a novel system architecture, and the technique is not currently utilized in contemporary DRAM memory controllers within the context of

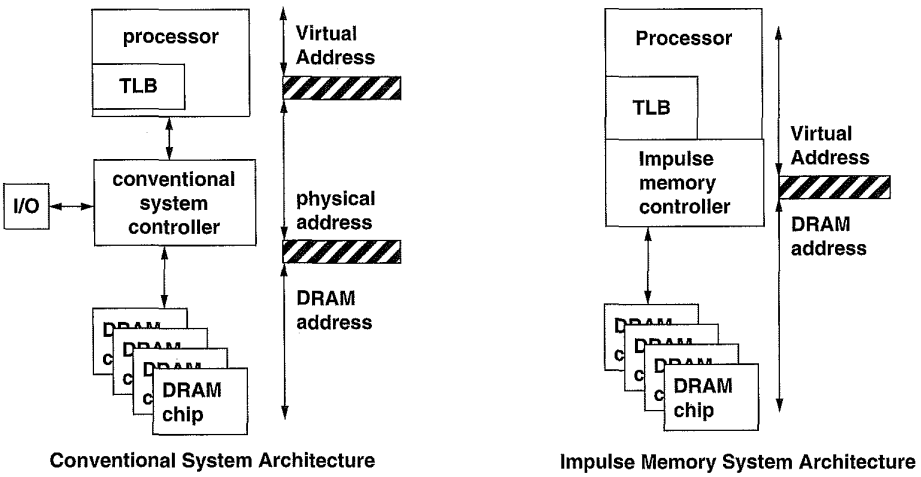


FIGURE 13.2: Comparison of conventional system architecture and the Impulse memory controller architecture.

conventional system architectures.[6] As a result, the discussion in the remaining sections of this chapter is focused on the examination of a DRAM controller in the context of conventional system architectures, and the address mapping schemes described herein are focused on the mapping from the physical address space into DRAM memory system organization indices.

13.3.1 Available Parallelism in Memory System Organization

In this section, available parallelism of channels, ranks, banks, rows, and columns is examined. The examination of available parallelism in DRAM memory system organization is then used as the basis of discussion of the various address mapping schemes.

Channel

Independent channels of memory possess the highest degree of parallelism in the organization of DRAM memory systems. There are no restrictions from the perspective of the DRAM memory system on requests issued to different logical channels controlled by independent memory controllers. For performance-optimized designs, consecutive cacheline accesses are mapped to different channels.[7]

Rank

DRAM accesses can proceed in parallel in different ranks of a given channel subject to the availability of the shared address, command, and data busses. However, rank-to-rank switching penalties in high-frequency, globally synchronous DRAM memory

[6]Fully Buffered DIMM memory systems present an interesting path to an unconventional system architecture that the Impulse memory controller or an Impulse-like memory controller may be able to take advantage of. That is, with multiple memory controllers controlling multiple, independent channels of Fully Buffered DIMMs, some channels may be designated for general-purpose access as in a conventional system architecture, while other channels may be used as high bandwidth, application-controlled, direct-access memory systems.

[7]The exploration of parallelism in the memory system is an attempt to extract maximum performance. For low-power targeted systems, different criteria may be needed to optimize the address mapping scheme.

systems such as DDRx SDRAM memory systems limit the desirability of sending consecutive DRAM requests to different ranks.

Bank

Similar to the case of consecutive memory accesses to multiple ranks, consecutive memory accesses can proceed in parallel to different banks of a given rank subject to the availability of the shared address, command, and data busses. In contemporary DRAM devices, scheduling consecutive DRAM read accesses to different banks within a given rank is, in general, more efficient than scheduling consecutive read accesses to different ranks since idle cycles are not needed to switch between different bus masters on the data bus. However, in most DRAM devices without a write buffer or separate internal datapaths for separate read and write data flow, a column read command that follows a column-write command is more efficiently performed to different ranks of memory as compared to a column-read command that follows a column-write command to different banks of the same rank. In modern computer systems, read requests tend to have higher spatial locality than write requests due to the existence of write-back caches. Moreover, the number of column-read commands that immediately follow column-write commands can be minimized in advanced memory controllers by deferring individual write requests and instead group schedule them as a sequence of consecutive write commands. Consequently, bank addresses are typically mapped lower than rank addresses in most controllers to favor the extraction of spatial locality from consecutive memory read accesses over the reduction of write-to-read turnaround times to open rows.[8]

Row

In conventional DRAM memory systems, only one row per bank can be active at any given instance in time provided that additional ESDRAM-like or VCDRAM-like row buffers are not present in the DRAM device. The result of the forced serialization of accesses to different rows of the same bank means that row addresses are typically mapped to the highest memory address ranges to minimize the likelihood that spatially adjacent consecutive accesses are made to different rows of the same bank.

Column

In open-page memory systems, cachelines with sequentially consecutive addresses are optimally mapped to the same row of memory to support streaming accesses. As a result, column addresses are typically mapped to the lower address bits of a given physical address in open-page memory systems. In contrast, cachelines with sequentially consecutive addresses are optimally mapped to different rows and different banks of memory to support streaming accesses in close-page memory systems. The mapping of sequentially consecutive cachelines to different banks, different ranks, and different channels scatters requests in streaming accesses to different rows and favors parallelism in lieu of spatial locality. The result is that in close-page memory systems, the low address bits of the column address that denote the column offset within a cacheline are optimally mapped to the lowest address bits of the physical address, but the remainder of the column address is optimally mapped to the high address ranges comparable to the row addresses.

13.3.2 Parameter of Address Mapping Schemes

To facilitate the examination of address mapping schemes, parametric variables are defined in this section to denote the organization of memory systems. For the sake of simplicity, a uniform memory system is assumed throughout this chapter. Specifically, the memory system under examination is assumed to

[8]Overhead-free scheduling of consecutive column accesses to different banks of a given rank of DRAM devices has long been the most efficient way to schedule memory commands. However, constraints such as burst chop in DDR3 SDRAM devices and t_{FAW} constraints in 8-bank DDRx SDRAM devices is now shifting the overhead distribution. Consequently, rank parallelism may be more favorable than bank parallelism in future DDRx SDRAM memory systems.

TABLE 13.2 Summary of system configuration variables

Symbol	Variable Dependence	Description
K	Independent	Number of channels in system
L	Independent	Number of ranks per channel
B	Independent	Number of banks per rank
R	Independent	Number of rows per bank
C	Independent	Number of columns per row
V	Independent	Number of bytes per column
Z	Independent	Number of bytes per cacheline
N	Dependent	Number of cachelines per row

have K independent channels of memory, and each channel consists of L ranks per channel, B banks per rank, R rows per bank, C columns per row, and V bytes per column. The total size of physical memory in the system is simply K * L * B * R * C * V. Furthermore, it is assumed that each memory request moves data with the granularity of a cacheline. The length of a cacheline is defined as Z bytes, and the number of cachelines per row is denoted as N. The number of cachelines per row is a dependent variable that can be computed by multiplying the number of columns per row by the number of bytes per column and divided through by the number of bytes per cacheline. That is, N = C * V / Z. The organization variables are summarized in Table 13.2.

In general, the value of system configuration parameters can be any positive integer. For example, a memory system can have six ranks of memory per channel and three channels of memory in the memory system. However, for the sake of simplicity, system parameters defined in this chapter are assumed to be integer powers of two, and the lowercase letters of the respective parameters are used to denote that power of two. For example, there are 2^b = B banks in each rank, and 2^l = L ranks in each channel of memory. A memory system that has the capacity of K * L * B * R * C * V bytes can then be indexed with k + l + b + r + c + v number of address bits.

13.3.3 Baseline Address Mapping Schemes

In the previous section, the available parallelism of memory channels, ranks, banks, rows, and columns was examined in abstraction. In this section, two baseline address mapping schemes are established. In an abstract memory system, the total size of memory is simply K * L * B * R * C * V. The convention adopted in this chapter is that the colon (:) is used to denote separation in the address ranges. As a result, k:l:b:r:c:v not only denotes the size of the memory, but also the order of the respective address ranges in the address mapping scheme. Finally, for the sake of simplicity, C * V is replaced with N * Z. That is, instead of the number of bytes per column multiplied by the number of columns per row, the number of bytes per cacheline multiplied by the number of cachelines per row can be used equivalently. The size of the memory system is thus K * L * B * R * N * Z, and an address mapping scheme for this memory system can be denoted as k:l:b:r:n:z.

Open-Page Baseline Address Mapping Scheme

In performance-optimized, open-page memory systems, adjacent cacheline addresses are striped across different channels so that streaming bandwidth can be sustained across multiple channels and then mapped into the same row, same bank, and same rank.[9]

[9]Again, assume a uniform memory system where all channels have identical configurations in terms of banks, ranks, rows, and columns.

The baseline open-page address mapping scheme is denoted as r:l:b:n:k:z.

Close-Page Baseline Address Mapping Scheme

Similar to the baseline address mapping scheme for open-page memory systems, consecutive cacheline addresses are mapped to different channels in a close-page memory system. However, unlike open-page memory systems, mapping cachelines with sequentially consecutive addresses to the same bank, same rank, and same channel of memory will result in sequences of bank conflicts and greatly reduce available memory bandwidth. To minimize the chances of bank conflict, adjacent lines are mapped to different channels, then to different banks, and then to different ranks in close-page memory systems. The baseline close-page address mapping scheme is denoted as r:n:l:b:k:z.

13.3.4 Parallelism vs. Expansion Capability

In modern computing systems, one capability that system designers often provide to end-users is the ability to configure the capacity of the memory system by adding or removing memory modules. In the context of address mapping schemes, the memory expansion capability means that respective channel, row, column, rank, and bank address ranges must be flexibly adjustable in the address mapping scheme depending on the configuration of the DRAM modules inserted into the memory system. As an example, in contemporary desktop computer systems, system memory capacity can be adjusted by adding or removing memory modules with one or two ranks of DRAM devices per module. In these systems, rank indices are mapped to the highest address range in the DRAM memory system. The result of such a mapping scheme means that an application that utilizes only a subset of the memory address space would typically make use of fewer ranks of memory than are available in the system. The address mapping scheme that allows for expansion capability thus presents less rank parallelism to memory accesses. Similarly, in cases where multiple channels can be configured independently and the memory system supports asymmetrical channel configurations, channel indices are also mapped to the high address ranges, and parallelism presented by multiple channels may not be available to individual applications. As a result, some high-performance systems enforce configuration rules that dictate symmetrical channel configurations.

In the respective baseline address mapping schemes described previously, channel and rank indices are mapped to the low-order address bits. However, in a flexible, user-configurable memory system, the channel and rank indices are moved to the high-order address bits. The result is that an expandable open-page memory system would utilize an address mapping scheme that is comparable to the ordering of k:l:r:b:n:z, and the k:l:r:n:b:z address mapping scheme would be used in an expandable, close-page memory system. In these address mapping schemes geared toward memory system expendability, some degrees of channel and rank parallelism are lost to single threaded workloads that use only a subset of the contiguous physical address space. The loss of parallelism for single threaded workloads in memory systems designed for configuration flexibility is less of a concern for memory systems designed for large multi-processor systems. In such systems, concurrent memory accesses from different memory-access streams to different regions of the physical address space would make better use of the parallelism offered by multiple channel and multiple ranks than the single threaded workload.

13.3.5 Address Mapping in the Intel 82955X MCH

In this section, Intel's 82955X Memory Controller Hub (MCH) is used as an example to illustrate address mapping schemes in a high-performance, multi-channel, multi-rank memory system. The 82955X MCH contains two memory controllers that can independently control two channels of DDR2 SDRAM devices. Each channel in the 82955X MCH supports up to four ranks of DRAM devices, and Table 13.3 summarizes six possible rank configurations supported by the 82955X MCH. In practical

terms, system boards that utilize the 82955X MCH can support one or two memory modules in each channel, and each memory module is composed of one or two identically configured ranks listed in Table 13.3.

The 82955X MCH supports address mapping schemes that are optimized for an open-page memory system. The interesting aspect of the 82955X MCH is that it supports different mapping schemes that are respectively targeted to obtain higher performance or configuration flexibility, and the 82955X MCH can deploy different mapping schemes depending on the organization of the memory modules inserted into the system. Specifically, the 82955X MCH can support the two channels configured with symmetric or asymmetric organizations of memory modules. Moreover, the 82955X MCH uses rank configuration registers to perform address mapping on a rank-by-rank basis. The topics of channel symmetry and per-rank address mapping in the 82955X MCH are examined in detail in the following sections.

Symmetric and Asymmetric Dual Channel Modes

In the case that the two channels are populated with memory modules with symmetrically matched capacities, the 82955X MCH can operate in *symmetric dual channel mode.* In the symmetric dual channel mode, sequentially consecutive cacheline addresses are mapped to alternating channels so that requests from a streaming request sequence are mapped to both channels concurrently. In the case where the 82955X MCH is configured with different capacities of memory modules in the two channels, the 82955X MCH operates in *asymmetric dual channel mode.* In the asymmetric dual channel mode, the physical address is mapped from 0 MB to the capacity of channel 0 and then to the full capacity of channel 1. In this manner, requests from a streaming request sequence are mapped to one channel at a time unless the array address space spans both channels.

Figure 13.3 illustrates a symmetric configuration and an asymmetric configuration for the 82955X MCH. The figure shows that in the symmetric configuration, both channel 0 and channel 1 are populated with a single-rank, 512-MB memory module that occupies rank 0 and a single-rank, 256-MB memory module that occupies rank 1. Although the 256-MB memory modules in rank 1 of both channels are not identical in organization, the fact that they are identical in capacity is sufficient for the 82955X MCH to utilize the symmetric dual channel mode. In contrast, the asymmetric configuration example shows that the two channels are populated with memory modules with different capacities and different numbers of ranks. In the asymmetric configuration, the physical address space extends from 0 to 512 MB in channel 0 and then from 512 to 1536 MB in channel 1.

Address Mapping Configuration Registers

The 82955X MCH uses configuration registers to support different address mapping schemes for performance optimization or configuration flexibility.

TABLE 13.3 DDR2 SDRAM rank configurations

Rank Capacity	Device Configuration: bank count x row count x col count x col size (col size in bytes)	Rank Composition: device density x device count	Rank Configuration: bank count x row count x col count x col size (B x R x C x V)	Bank Address Bits (b)	Row Address Bits (r)	Column Address Bits (c)	Column Address Offset (v)
128 MB	4 x 8192 x 512 x 2	256 Mbit x 4	4 x 8192 x 512 x 8	2	13	9	3
256 MB	4 x 8192 x 1024 x 2	512 Mbit x 4	4 x 8192 x 1024 x 8	2	13	10	3
256 MB	4 x 8192 x 1024 x 1	256 Mbit x 8	4 x 8192 x 1024 x 8	2	13	10	3
512 MB	8 x 8192 x 1024 x 2	1 Gbit x 4	8 x 8192 x 1024 x 8	3	13	10	3
512 MB	4 x 16384 x 1024 x 1	512 Mbit x 8	4 x 16384 x 1024 x 8	2	14	10	3

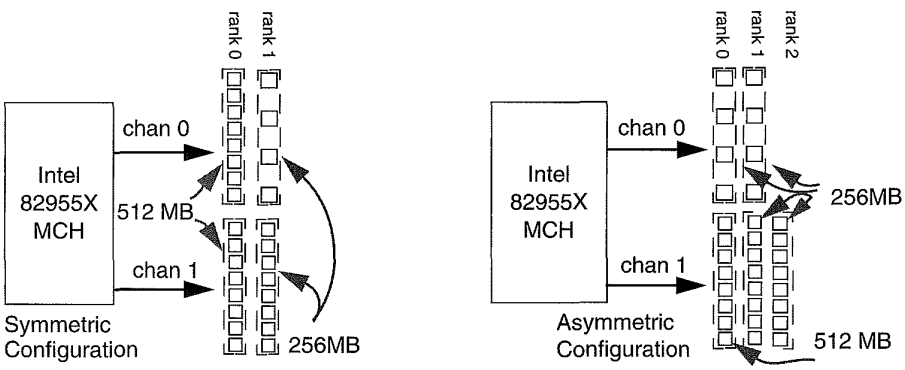


FIGURE 13.3: Symmetric and asymmetric channel configurations in the 82955X MCH.

Two types of configuration registers are used in the 82955X MCH to aid it in the task of address mapping: rank address boundary registers and rank architectural registers.

The function of the rank address boundary register is to define the upper address boundaries for a given rank of DRAM devices. There are four rank address boundary registers per channel. In asymmetrical channel model, each register contains the highest addressable location of a single channel of memory. In symmetrical channel mode, cacheline addresses are interleaved between the two channels, and the rank address boundary registers contain the upper address boundaries for a given rank of DRAM devices for both channels of memory. The rank architectural registers identify the size of the row for the DRAM devices inserted into the system. In the 82955X MCH, there are four rank architectural registers per channel, with one register per rank.

Per-Rank Address Mapping Schemes

The address mapping configuration registers are set at system initialization time by the 82955X MCH, and they contain values that reflect the capacity and organization of the DRAM devices in the memory system. With the aid of the rank address boundary configuration registers, the 82955X MCH can unambiguously resolve a physical address location to a given rank of memory. Then, with the aid of the rank architectural registers, the organization of the DRAM devices to the given rank of memory is also known, and the physical address can be further separated into row, bank, and column addresses. That is, the address mapping scheme in the 82955X MCH is generated on a rank-by-rank basis.

Figure 13.4 illustrates the address mapping scheme of 82955X MCH. The figure shows that in single channel or asymmetric dual channel mode, the 82955X MCH maps the three least significant bits of the physical address as the byte offset of the 8-byte-wide memory module. Then, depending on the number of columns on the memory module, the next 9 or 10 bits of the physical address are used to denote the column address field. The bank address then follows the column address, and the most significant address bits are used for the row address.

In the 82955X MCH, the channel address is mapped to different address bit fields, depending on the operating mode of the memory controller. Figure 13.4 does not show the channel address field for the single channel or asymmetric channel mode, since each channel is a contiguous block of memory, and the channel address is mapped to the highest available physical address bit field. However,

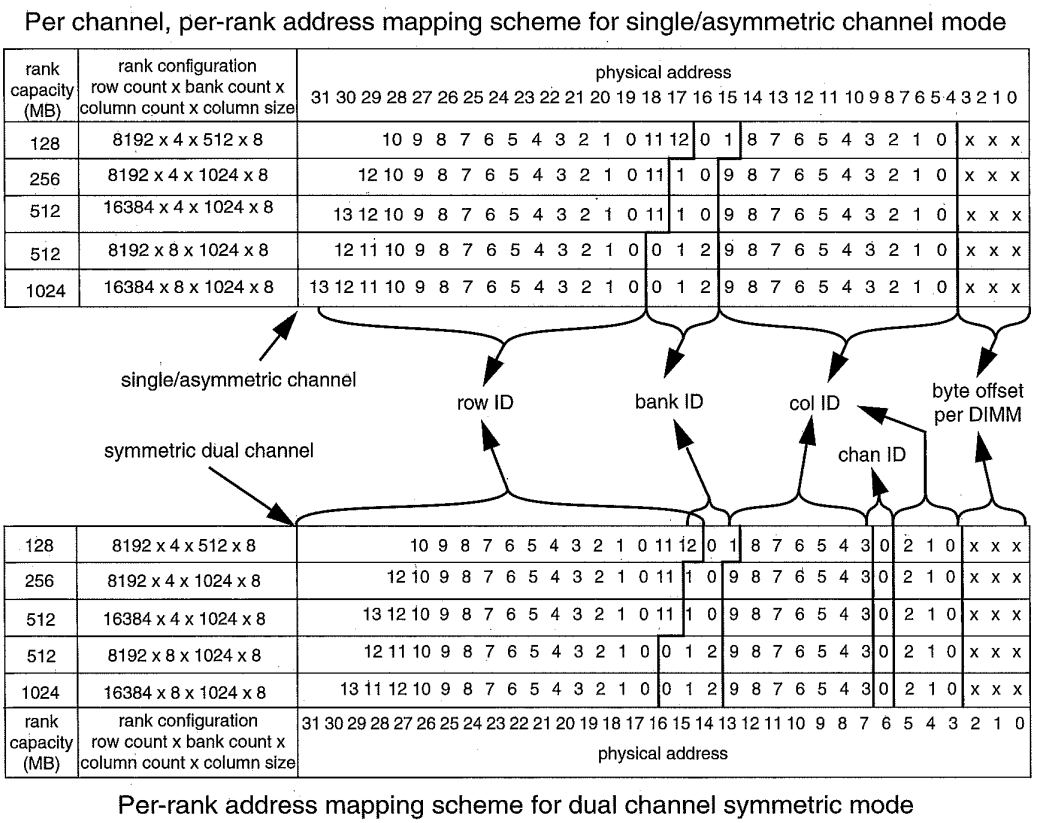


FIGURE 13.4: Per-rank address mapping schemes in the 82955X MCH.

in symmetric dual channel mode, the cacheline addresses are interleaved between the two channels, and the channel address is mapped to the low range of the address bit fields. Specifically, Figure 13.4 shows that the address mapping schemes shown for the single/asymmetric channel mode and the dual channel symmetric mode are essentially the same, and the only difference between the two sets of address mapping schemes is that the 6th bit in the physical address is used to denote the channel address, and the respective addresses in the dual channel mode are shifted over by 1 bit position to the left.

Quick Summary of Address Mapping in the 82955X MCH

Figure 13.4 serves as a concrete example that illustrates some interesting aspects of address mapping schemes used in contemporary memory controllers. In particular, the address mapping schemes illustrated in Figure 13.4 for the 82955X MCH are classical

open-page-optimal address mapping schemes. For example, Figure 13.4 shows that in the single/asymmetric channel mode, the address mapping scheme in the 82955X MCH can be represented as k:l:r:b:n:z, and in the symmetric dual channel mode, the address mapping scheme can be represented as l:r:b:n:k:z. In both cases, the column address fields are mapped to the low address ranges so that spatially adjacent memory address locations can be directed to the same open page. Similarly, in the various address mapping schemes illustrated in Figure 13.4, the 82955X MCH shows that the side effect of granting the end-users the ability to configure the memory system with differently organized memory modules is that rank parallelism to spatially adjacent memory accesses is lost. Although the rank address field is not explicitly illustrated in Figure 13.4, the use of the address boundary registers and per-rank address mapping schemes means that the rank address field is mapped to the high address ranges above the row address field.

Figure 13.4 shows that the 82955X MCH has been cleverly designed so that most of the bit positions are directed to the same address fields regardless of the organization of the memory modules in the memory system. For example, Figure 13.4 shows that physical address bits 16 through 26 are used to denote row addresses 0 through 10 in the single/asymmetric channel mode, regardless of the number and type of memory modules placed in the memory system. In this manner, only a few bit positions will have to be dynamically adjusted depending on the organization of the memory system, and bit positions shown with the grey background in Figure 13.4 are always directed to the same address fields.

Finally, the address mapping scheme in the 82955X MCH means that single threaded streaming applications often cannot take advantage of the parallelism afforded by multiple ranks and the two channels in asymmetric channel mode. Fortunately, multiprocessor and multi-threaded processor systems with concurrently executing contexts can access different regions of memory and may be able to take advantage of the parallelism afforded by the multiple ranks and multiple channels in asymmetric channel mode. However, the amount of achievable parallelism depends on the specific access request sequences and the locations of the data structures accessed by the concurrently executing process contexts.

13.3.6 Bank Address Aliasing (Stride Collision)

One additional issue in the consideration of an address mapping scheme is the problem of bank address aliasing. The problem of bank address aliasing occurs when arrays whose respective sizes are relatively large powers-of-two are accessed concurrently with strided accesses to the same bank. Figure 13.4 shows that in a system that uses 1-GB DDR2 SDRAM memory modules with the 82955X MCH in dual channel mode, the bank address for each access is obtained from physical address bit positions 14 through 16. That is, in this system configuration, all contiguously allocated arrays that are aligned on address boundaries that are integer multiples of 2^{17} bytes from each other would have array elements that map to identical banks for all corresponding array elements.

For example, the task of array summation, where the array elements of arrays A and B are added together and then stored into array C, requires that the corresponding elements of A, B, and C be accessed concurrently. In the case where arrays A, B, and C are contiguously allocated by the system and mapped to integer multiples of 128-kB address boundaries from each other, then array elements A[i], B[i], and C[i], would be mapped to different rows within the same bank for all valid array indices i, resulting in multiple bank conflicts for each step of the array summation process in the system described above.

In general, the bank address aliasing problem can be alleviated by several different methods. One method that can alleviate the bank address aliasing problem is the conscientious application of padding or offsets to large arrays so that bank conflicts are not generated throughout concurrent array accesses to those large arrays.[10] A second method that can alleviate the bank address aliasing problem is the conscientious design

[10] A simple offset insertion increased STREAM Triad bandwidth by 25% in a test system with an Intel i875P system controller.

of a memory management unit that can purposefully allocate large arrays to non-contiguous pages in the physical address space. In this manner, the chance of a bank conflict changes from a guaranteed event that occurs for every single access to the array to a probabilistic event that depends on the number of banks and ranks in the memory system. Finally, improved address mapping schemes have been proposed to alleviate the bank address aliasing problem, and they are described in the following section.

Hardware Solution to the Address Aliasing Problem

The bank address aliasing problem has been investigated by Lin et al. [2001] and Zhang et al. [2000]. The schemes proposed by Lin and Zhang are similar to schemes applied to different memory systems. The basic idea of taking the row address and bitwise XOR'ed with the bank address to generate new bank addresses that are not aligned for concurrent accesses to large arrays is common to both designs. However, the generous rank and bank parallelism in the fully configured Direct RDRAM memory system allowed Lin to create a 1:1 mapping that permutes the available number of banks through the entire address space in the system configuration examined. In contrast, Zhang illustrated a more modest memory system where the page index was larger than the bank index. The mapping scheme described by Zhang is shown in Figure 13.5. Figure 13.5 shows that the problem for the scheme described by Zhang is that there are relatively few banks in contemporary SDRAM and DDRx SDRAM memory systems, and for a DRAM memory system with 2^b banks, there are only 2^b possible permutations in a 1:1 mapping that maps a physical address to the memory address. In the bank address permutation scheme for the conventional SDRAM-type memory system proposed by Zhang, the address aliasing problem is simply shifted to a larger granularity. That is, without the bank permutation scheme illustrated in Figure 13.5, arrays aligned on address boundaries of $2^{(b+p)}$ bytes would suffer a bank conflict on every pair of concurrent array accesses. The implementation of the bank permutation scheme means that arrays aligned on address boundaries of $2^{(b+p)}$ bytes no longer suffer from the same address aliasing problem, but arrays that are aligned on address boundaries of $2^{(b+p+b)}$ bytes continue to suffer a bank conflict on every pair of concurrent array accesses. Essentially, there are not enough banks to rotate through the entire address space in a contemporary memory system to completely avoid the memory address aliasing problem.

13.4 Performance Optimization

The performance characteristic of a modern DRAM memory controller depends on implementation-specific DRAM command and memory transaction ordering policies. A DRAM controller can be designed to minimize complexity without regard to performance or designed to extract the maximum performance from the memory system by implementing aggressive DRAM command and memory transaction ordering policies. DRAM command and transaction

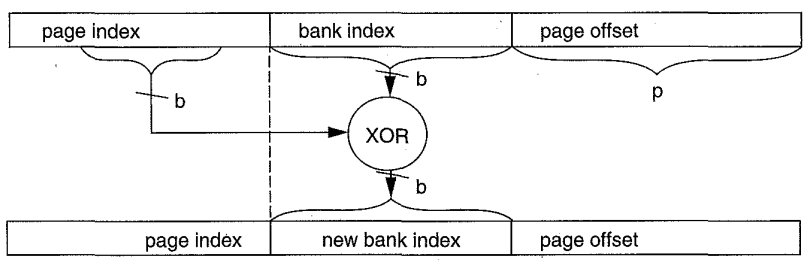


FIGURE 13.5: Address mapping scheme proposed by Zhang et al. [2000].

ordering policies have been studied by Briggs et al. [2002], Cuppu et al. [1999], Hur and Lin [2004], McKee et al. [1996], Lin et al. [2001], and Rixner et al. [2000]. In studies performed by Briggs et al., Cuppu et al., McKee et al., Lin et al., and Rixner et al., various DRAM-centric scheduling schemes are examined. In the study performed by Hur et al., the observation is noted that the ideal DRAM scheduling algorithm depends not only on the optimality of scheduling to the DRAM memory system, but also on the requirement of the application. In particular, the integration of DRAM memory controllers with the processor core onto the same silicon die means that the processor core can interact directly with the memory controller and provide direct feedback to select the optimal DRAM command scheduling algorithm.

The design of a high-performance DRAM memory controller is further complicated by the emergence of modern, high-performance, multi-threaded processors and multi-core processors. While the use of multi-threading has been promoted as a way to hide the effects of memory-access latency in modern computer systems, the net effect of multi-threaded and multi-core processors on a DRAM memory system is that the intermixed memory request stream from the multiple threaded contexts to the DRAM memory system disrupts the row locality of the request pattern and increases bank conflicts [Lin et al. 2001]. As a result, an optimal DRAM controller design not only has to account for the idiosyncrasies of specific DRAM memory systems, application-specific requirements, but also the type and number of processing elements in the system.

The large number of design factors that a design engineer must consider underlines the complexity of a high-performance DRAM memory controller. Fortunately, some basic strategies exist in common for the design of high-performance DRAM memory controllers. Specifically, the strategies of bank-centric organization, write caching, and seniors first are common to many high-performance DRAM controllers, while specific adaptive arbitration algorithms are unique to specific DRAM controllers and systems.

13.4.1 Write Caching

One strategy deployed in many modern DRAM controllers is the strategy of write caching. The basic idea for write caching is that write requests are typically non-critical in terms of latency, but read requests are typically critical in terms of latency. As a result, it is typically desirable to defer write requests and allow read requests to proceed ahead of write requests, as long as the memory ordering model of the system supports this optimization and the functional correctness of programs is not violated. Furthermore, DRAM devices are typically poorly designed to support back-to-back read and write requests. In particular, a column read command that occurs immediately after a column write command typically incurs a large penalty in the data bus turnaround time in conventional DDRx SDRAM devices due to the fact that the column read command must await the availability of the internal datapath of the DRAM device that is shared between read and write commands.

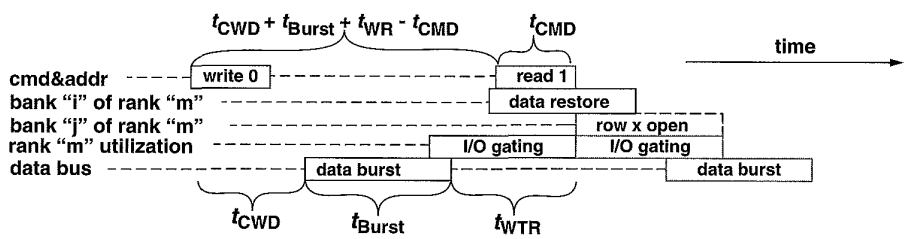


FIGURE 13.6: Write command following read command to open banks.

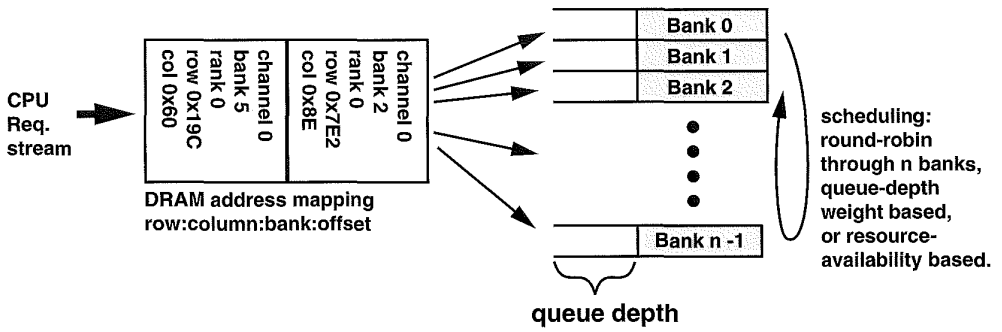


FIGURE 13.7: Per-bank organization of DRAM request queues.

Figure 13.6 repeats the illustration of a column read command that follows a write command and shows that, due to the differences in the direction of data flow between read and write commands, significant overheads exist when column read and write commands are pipelined back to back. The strategy of write caching allows read requests that may be critical to application performance to proceed ahead of write requests, and the write caching strategy can also reduce read-write overheads when combined with a strategy to burst multiple write requests to the memory system consecutively. One memory controller that utilizes the write caching strategy is Intel's i8870 system controller which can buffer upwards of 8 kB of write data to prioritize read requests over write requests. However, in systems that implement write caching, significant overhead in terms of latency or hardware complexity may exist due to the fact that the address of all pending read requests must be checked against the address of cached writes, and the memory controller must provide the consistency guarantee to ensure the correctness of memory-access ordering.

13.4.2 Request Queue Organizations

To control the flow of data between the DRAM memory controller and DRAM devices, memory transactions are translated into sequences of DRAM commands in modern DRAM memory controllers. To facilitate the pipelined execution of these DRAM commands, the DRAM commands may be placed into a single queue or multiple queues. With the DRAM commands organized in the request queuing structure, the DRAM memory controller can then prioritize DRAM commands based on many different factors, including, but not limited to, the priority of the request, the availability of resources to a given request, the bank address of the request, the age of the request, or the access history of the agent that made the request.

One organization that can facilitate the pipelined execution of DRAM commands in a high-performance DRAM memory controller is the per-bank queuing organization.[11] In the per-bank queuing structure, memory transaction requests, assumed to be of equal priority, are sorted and directed to different queues on a bank-by-bank basis. Figure 13.7 shows one organization of a set of request queues organized on a

[11]The per-bank request queuing construct is an abstract construct. Memory controllers can utilize a unified queue with sophisticated hardware to perform the transaction reordering and bank rotation described herein, albeit with greater difficulty.

per-bank basis. In the organization illustrated in Figure 13.7, memory transaction requests are translated into memory addresses and directed into different request queues based on their respective bank addresses. In an open-page memory controller with request queues organized comparably to Figure 13.7, multiple column commands can be issued from a given request queue to a given bank if the column access commands are directed to the same open row. In the case where a given request queue has exhausted all pending requests to the same open row and all other pending requests in the queue are addressed to different rows, the request queue can then issue a precharge command and allow the next bank to issue commands into the memory system.[12]

Figure 13.7 shows that one way to schedule requests from the per-bank queuing structure is to rotate the scheduling priority in a round-robin fashion from bank to bank. The round-robin, bank-rotation command scheduling scheme can effectively hide DRAM bank conflict overhead to a given bank if there are sufficient numbers of pending requests to other banks that can be processed before the scheduling priority rotates back to the same bank. In a close-page memory controller, the round-robin bank-rotation scheme maximizes the temporal distance between requests to any given bank without sophisticated logic circuits to resolve against starvation. However, the round-robin scheme may not always produce optimal scheduling, particularly, for open-page memory controllers. In open-page memory controllers, the address mapping scheme maps spatially adjacent cachelines to open rows, and multiple requests to an open row may be pending in a given queue. As a result, a weight-based priority scheduling scheme, where the queue with the largest number of pending requests, is prioritized ahead of other queues with fewer pending requests, may be more optimal than a strictly round-robin priority scheduling scheme.[13]

The per-bank queue organization may be favored in large memory systems where high request rates are directed to relatively few banks. In memory systems where there are relatively lower access rates to a large number of banks in the system, dedicated per-banks queues are less efficient in organizing requests. In these memory systems, the queue structure may be more optimally organized as a pool of general-purpose queue entries where each queue entry can be directed to different banks as needed.

13.4.3 Refresh Management

One issue that all modern DRAM controllers must deal with to ensure the integrity of data stored in DRAM devices is the refresh function. In the case where a modern memory system is inactive for a short period of time, all DRAM devices can make use of a DRAM device controlled self-refresh mode, where the DRAM memory controller can be temporarily powered down and placed into a sleep state while the DRAM device controls its own refresh function. However, the entry into and exit out of the self-refresh mode is typically performed under the explicit control of the DRAM memory controller, and the self-refresh action is not engaged in during normal operations in most modern DRAM devices.

One exception to the explicit management of the refresh function by the memory controller can be found in some pseudo-static DRAM devices such as MobileRAM, where temperature-compensated self-refresh is used as part of the normal operating mode to minimize refresh power consumption. Moreover, the hidden self-refresh removes the complexity of refresh control from the memory controller

[12]The bank-centric organization assumes that all requests from different agents are of equal priority and access rates from different agents are comparable. In practical terms, additional safeguards must be put in place to prevent the scenario where a constant stream of requests to an open row starves other requests to different rows. In some controllers, a limit is placed on the maximum number of consecutive column commands that can be scheduled to an open row before the open row is closed to prevent starvation and to ensure some degree of fairness to requests made to different rows of the same bank.

[13]Weight-based schemes must also be constrained by age considerations to prevent starvation.

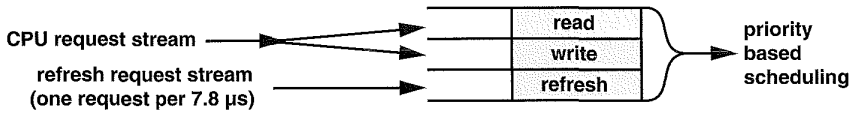


FIGURE 13.8: Priority-based scheduling for refresh requests.

and contributes to the illusion of the MobileRAM DRAM device as a pseudo-static memory device. The timing and interval of the DRAM refresh action in these devices are hidden from the memory controller. Therefore, in the case where a memory request from the memory controller collides with the hidden refresh action, the pseudo-static DRAM device asserts a wait signal to inform the memory controller that the data return from the pseudo-static DRAM device will be delayed until after the self-refresh action within the pseudo-static DRAM device is completed and the wait signal is deasserted by the pseudo-static DRAM device. However, a wait signal from the pseudo-static DRAM device that can delay state transition in the memory controller effectively introduces a slow signal path into the memory controller and effectively limits the operating frequency of the memory controller. Consequently, the explicit wait state signal that enables the hidden self-refresh function in normal operating mode is only used in relatively low-frequency, pseudo-static DRAM-based memory systems designed for battery-operated mobile platforms, and the refresh function in DRAM devices targeted for high-frequency DRAM memory systems remains under the purview of the DRAM memory controller.

To ensure the integrity of data stored in DRAM devices, each DRAM row that contains valid data must be refreshed at least once per refresh period, typically 32 or 64 ms in duration.[14] In terms of a DRAM device that requires 8192 refresh commands every refresh period, mathematics dictate that an all-banks-concurrent refresh command must be issued to the DRAM device once every 7.8 µs for the device with a 64-ms period requirement. Fortunately, DRAM refresh commands can be deferred for short periods of time to allow latency-critical memory read requests to proceed ahead. Consequently, the DRAM controller need not adhere to a strict requirement of having to send an all-banks-concurrent refresh command to the DRAM device every 7.8 µs. To take advantage of the fact that refresh commands can be deferred within a reasonable timing window, Figure 13.8 shows an organization of the queuing structure where the microprocessor request stream is separated into read and write request queues and the request commands are placed into the refresh queue at a constant rate of one refresh command every 7.8 µs. In the structure illustrated in Figure 13.8, each refresh request is attributed with a count that denotes the number of cycles that the request has been deferred. In this manner, in the case that the refresh request is below a preset deferral threshold, all read and write requests will have priority over the refresh request.[15] In the case where the system is idle with no other pending read or write requests, the refresh request can then be sent to the DRAM devices. In the case where the system is filled with pending read and write requests but a DRAM refresh request has nearly exceeded the maximum deferral time, that DRAM refresh request will then receive the highest scheduling priority to ensure that

[14]Some DRAM devices contain additional registers to define the ranges of rows that need to be refreshed. In these devices, the refresh action can be ignored for certain rows that do not contain valid data.
[15]The maximum refresh request deferral time is defined in the device data sheet for DRAM devices. In modern DRAM devices such as DDR2 SDRAM devices, the period DRAM refresh command can be deferred for as long a 97.8-µs refresh command intervals.

TABLE 13.4 Refresh cycle times of DDR2 SDRAM devices

Density	Bank Count	Row Count	Row Size (bits)	t_{RC}: Row Cycle Time (ns)	t_{RFC}: Refresh Cycle Time (ns)
256 Mbit	4	8192	8192	55	75
512 Mbit	4	16384	8192	55	105
1 Gbit	8	16384	8192	54	127.5
2 Gbit	8	32768	8192	54	197.5
4 Gbit	8	65536	8192	54	327.5

the refresh request occurs within the required time period to ensure data integrity in the memory system.

One feature under consideration that could further increase the complexity of future DRAM memory controllers is the functionality of per-bank refresh. Table 13.4 illustrates that t_{RFC}, the refresh cycle time, is increasing with each generation of higher density DRAM devices, and the bandwidth overhead of the refresh functionality grows proportionally.[16] One proposal to minimize the bandwidth impact of DRAM refresh is to replace or supplement the all-banks-concurrent refresh command with separate refresh commands that refresh one row in one bank at a time as opposed to refreshing one row in all banks within a rank of DRAM devices concurrently.[17]

The performance benefit of the separate per-bank refresh command can be easily computed since each per-bank refresh command need only respect the t_{RC} row cycle time constraint rather than the t_{RFC} refresh cycle time constraint. However, one caveat of the per-bank refresh command proposal is that the fine-grained control of the refresh function on a per-bank basis means that the complexity of the DRAM memory controller must increase proportionally to deal with separate refresh requests to each bank, and the queueing structure required may be far more complex than the sample queuing structure illustrated in Figure 13.7. Finally, the all-banks-concurrent refresh command also serves as a time period where the DRAM device can perform housekeeping duties such as signal recalibration between DRAM devices and the memory controller. Without the all-banks-concurrent refresh command, DRAM devices would lose the guaranteed time period where circuits within the DRAM devices are active, but the interface of the DRAM devices are idle to perform these types of housekeeping duties.

13.4.4 Agent-Centric Request Queuing Organization

In previous sections, techniques to maximize bandwidth utilization and decrease effective read latency were examined. However, one issue that was left out of previous discussions is that regardless of the performance techniques deployed, an overriding consideration in the design of a modern DRAM memory controller is one of fairness. That is, in any transaction request reordering mechanism, anti-starvation safeguards must exist to ensure that no

[16]There are more than 8192 rows in higher density DDR2 SDRAM devices, but the number of refresh commands per 64-ms time period remains constant. For example, the 4-Gbit DDR2 device must refresh 8 rows of data with each refresh command.

[17]The refresh command is a relatively energy-intensive operation. The instantaneous power draw of large, multi-rank memory systems, where all ranks are refreshed concurrently with a single refresh command, could significantly increase the peak power consumption profile of the memory system. To limit the peak power consumption profile, some DRAM memory controllers are designed to refresh each rank of DRAM devices separately and scheduled some time apart from each other.

request can be deferred for an indefinite period of time. The anti-starvation safeguards are particularly important in the context of multiple agents that share use of the same memory system. In particular, the issues of fairness and performance optimization for multiple agents with drastically different request rates and address sequences must be carefully traded off against each other. For example, a microprocessor running a typical application may require relatively low bandwidth, but read requests from the processor must be considered as latency critical. In contrast, a graphics processor that is connected to the same memory system may require a large amount of guaranteed bandwidth, but individual memory transaction requests from the graphics processor may be deferred in favor of requests from the microprocessor. Finally, memory transaction requests from relatively low-bandwidth and low-priority I/O devices may be deferred, but these requests cannot be deferred for an indefinite period of time so as to cause starvation for the I/O devices. That is, in the context of multiple agents that share a common memory system, better performance may be measured in terms of achieving an equitable balance in the usage of the memory system rather than obtaining the absolute maximum bandwidth from the DRAM devices.

The conflicting requirements of fairness and performance are important considerations that must be accounted for by the DRAM memory controller's scheduling mechanism. Fortunately, the scheduling mechanism used to deal with the DRAM device refresh requirement can be broadly extended to deal with a broad range of agents that require low latency or guaranteed bandwidth. That is, DRAM refresh commands can be considered as a sequence of requests from an agent that requires some amount of guaranteed bandwidth from the DRAM memory system. This agent, along with a number of other agents that require differing amounts of guaranteed bandwidth, must share the memory system with other agents that have no fixed requirements in terms of bandwidth, but must have low average access latency.

Figure 13.9 shows an organization of the queuing structure where the memory controller selects between requests from low-latency agents and guaranteed bandwidth agents. Figure 13.9 shows that requests from low-bandwidth and guaranteed bandwidth agents are directed to a two-level queuing structure, where requests are first sent to a pending queue and then moved to a scheduling queue under respective rate-controlled conditions. The rate controls ensure that the non-latency critical request agents

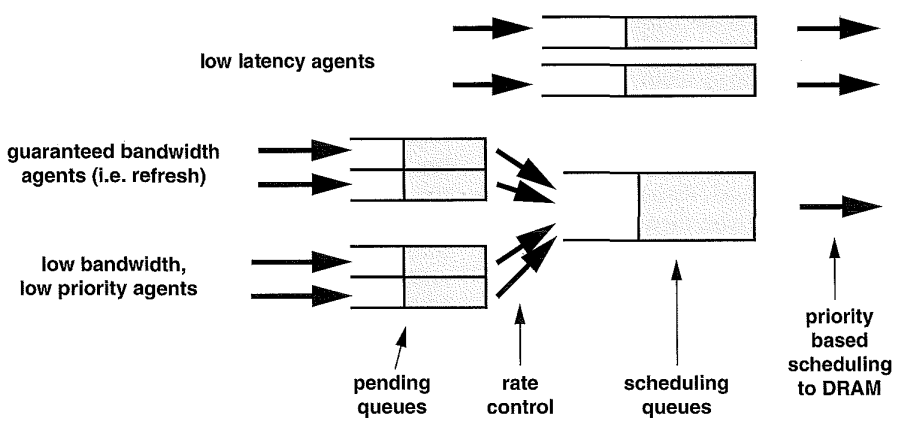


FIGURE 13.9: Sample system-level arbiter design.

cannot saturate the memory system with requests at the expense of other agents. In the queuing structure illustrated in Figure 13.9, requests from the low-latency agents are typically scheduled with the highest priority, except when the scheduling queue for the guaranteed bandwidth agents are full. To ensure that the bandwidth guarantees are met, the scheduling priority must favor the guaranteed bandwidth agents in the case where the shared scheduling queue for the guaranteed bandwidth agents is full.

In the previous section, DRAM-centric request scheduling algorithms are examined in the context of obtaining the highest performance from the DRAM memory system, given that all requests are equal in importance and that requests can be freely reordered for better performance. However, in a system where multiple agents must share the use of the memory system, not all requests from different agents are equal in importance. As a result, to obtain better performance in the system as a whole, both DRAM-centric and agent-centric algorithms must be considered. Figure 13.9 thus illustrates a two-level scheduling algorithm to ensure fairness and system throughput.

13.4.5 Feedback-Directed Scheduling

In modern computer systems, memory access is performed by the memory controller on behalf of processors or intelligent I/O devices. Memory-access requests are typically encapsulated in the form of transaction requests that contain the type, address, and data for the request in the case of write requests. However, in the majority of systems, transaction requests typically do not contain information to allow a memory controller to prioritize the transactions based on the specific requirements of the workload. Rather, memory controllers typically rely on the type, the access history, the requesting agent, and the state of the memory system to prioritize and schedule the memory transactions. In one recent study performed by Hur and Lin [2004], the use of a history-based arbiter that selects among different scheduling policies is examined in detail. In this study, the memory-access request history is used to select from different prioritization policies dynamically, and speedups between 5 and 60% are observed on some benchmarks.

The exploration of a history-based DRAM transaction and command scheduling algorithm by Hur and Lin is enabled by the use of a processor with an integrated DRAM controller, the IBM POWER5. The trend in the integration of memory controllers and processors means that the memory controllers will gain access to transaction scheduling information that they could not access as stand alone controllers.

As more processors are designed with integrated DRAM memory controllers, these processors can communicate directly with the DRAM memory controllers and schedule DRAM commands based not only on the availability of resources within the DRAM memory system, but also on the DRAM command-access history. In particular, as multi-threaded and multi-core processors are integrated with DRAM memory controllers, these DRAM memory controllers not only have to be aware of the availability of resources within the DRAM memory system, but they must also be aware of the state and access history of the respective threaded contexts on the processor in order to achieve the highest performance possible.

13.5 Summary

An analogy that may be made for the transaction queuing mechanism of a modern, high-performance DRAM memory controller is one that compares the transaction queuing mechanism of a high-performance DRAM controller to the instruction Reorder Buffer (ROB) of high-performance microprocessors that dynamically convert assembly instructions into internal microoperations that the processor executes out of order. In the ROB, the microprocessor accepts as input a sequence of assembly instructions that it converts to microoperations. In the transaction queue of the memory controller, the transaction queue accepts read and write requests that it must convert to DRAM commands that the memory controller then attempts to execute. Similar to the microoperations in the ROB, DRAM commands can be scheduled subject to the ordering constraints of the transaction requests and availability of the resources.

Computer Systems Design / Computer Hardware

MEMORY SYSTEMS
Cache, DRAM, Disk

Bruce Jacob, *University of Maryland at College Park*
Spencer W. Ng, *Hitachi Global Storage Technologies*
David T. Wang, *MetaRAM*

"*Memory Systems: Cache, DRAM, Disk* fills a huge void in the literature about modern computer architecture...Jacob, Ng, and Wang have created one of the truly great technology books that turns reading about bits and bytes into an exciting journey towards understanding technology."
— Michael Schuette, *VP of Technology Development at OCZ Technology*

"*Memory Systems: Cache, DRAM, Disk* is remarkable in both its scope and depth.... This book will doubtless be the definitive reference for students and designers of memory systems for many years to come."
— Jim Smith, *University of Wisconsin – Madison*

The performance gap between processor and memory speeds has increased dramatically. System designers have talked for over a decade about "hitting the memory wall," a condition in which overall system speed is significantly compromised by the memory system's inability to keep pace with the processor. That condition is today's reality—the memory system has become the most critical performance bottleneck, and memory subsystems design is now the greatest challenge facing design engineers.

Memory Systems: Cache, DRAM, Disk is the first book to cover comprehensively the logical design and operation, physical design and operation, performance characteristics and resulting design trade-offs, and the energy consumption of modern memory hierarchies. This book describes the physical design and detailed software emulation of the entire memory hierarchy, from cache to disk and prepares designers to tackle the challenging optimization problems that result from the side effects that can appear at any point in the entire memory hierarchy.

CONTENT HIGHLIGHTS

- Provides a comprehensive, in-depth understanding of all levels of the system hierarchy—cache, DRAM, and disk.
- Describes a holistic approach for evaluating the system-level effects of all design choices.
- Includes a link to the open-source software DRAMsim, a detailed and accurate parameter-driven simulator of modern DRAM-based memory systems that models performance and energy consumption for each component.

ABOUT THE AUTHORS

Bruce Jacob is an Associate Professor of Electrical and Computer Engineering at the University of Maryland at College Park.

Spencer W. Ng is a senior technical staff member with Hitachi Global Storage Technologies. He has been a researcher in the field of storage for over twenty years, and is the holder of about twenty issued U.S. patents.

David T. Wang is a senior technical staff member at MetaRAM, a memory systems startup in Silicon Valley.

Online support materials for this book are available at
textbooks.elsevier.com/9780123797513

ISBN: 978-0-12-379751-3
90000
9 780123 797513

ELSEVIER

MK
MORGAN KAUFMANN PUBLISHERS
AN IMPRINT OF ELSEVIER SCIENCE
www.mkp.com