DRAM: Architectures, Interfaces, and Systems

A Tutorial

Bruce Jacob and David Wang

Electrical & Computer Engineering Dept.
University of Maryland at College Park

http://www.ece.umd.edu/~blj/DRAM/
Outline

• Basics
• DRAM Evolution: Structural Path
• Advanced Basics
• DRAM Evolution: Interface Path
• Future Interface Trends & Research Areas
• Performance Modeling: Architectures, Systems, Embedded

Break at 10 a.m. — Stop us or starve
Basics

DRAM ORGANIZATION

- Storage element (capacitor)
- Word Line
- Bit Line
- Switching element
- Data In/Out Buffers
- Sense Amps
- Column Decoder
- Memory Array
- DRAM
- Row Decoder
- ... Bit Lines...
Basics

BUS TRANSMISSION

CPU

MEMORY CONTROLLER

BUS

DRAM

Column Decoder

Sense Amps

... Bit Lines...

Row Decoder

... Word Lines...

Memory Array

Data In/Out Buffers
At this point, all but lines are at 1/2 voltage level.
The read discharges the capacitors onto the bit lines...
this pulls the lines just a little bit high or a little bit low;
the sense amps detect the change and recover the full signal.
The read is destructive—the capacitors have been discharged....
however, when the sense amps pull the lines to the full logic-level (either high or low),
the transistors are kept open and so allow their attached capacitors to become recharged
(if they hold a '1' value).

Basics

[PRECHARGE and] ROW ACCESS

AKA: OPEN a DRAM Page/Row
or
ACT (Activate a DRAM Page/Row)
or
RAS (Row Address Strobe)
once the data is valid on ALL of the bit lines, you can select a subset of the bits and send them to the output buffers.

CAS picks one of the bits.

big point: cannot do another RAS or precharge of the lines until finished reading the column data.

can't change the values on the bit lines or the output of the sense amps until it has been read by the memory controller.

Basics

COLUMN ACCESS

**READ Command**
**or**
**CAS:** Column Address Strobe
Basics

DATA TRANSFER

note: page mode enables overlap with CAS
Basics

BUS TRANSMISSION

- CPU
- MEMORY CONTROLLER
- BUS
- DRAM
  - Column Decoder
  - Sense Amps
  - Memory Array
  - Word Lines...
  - ... Bit Lines...
- Row Decoder
- Data In/Out Buffers
“DRAM Latency” = A + B + C + D + E + F
Basics

PHYSICAL ORGANIZATION

This is per bank ...
Typical DRAMs have 2+ banks
Basics

Read Timing for Conventional DRAM

### Basics

**Read Timing for Conventional DRAM**

- **RAS**
- **CAS**
- **Row Access**
- **Column Access**
- **Data Transfer**

### Diagram

- **Address**
  - **Row Address**
  - **Column Address**
- **DQ**
  - **Valid Dataout**

### Chart

- **Row Access**
- **Column Access**
- **Data Transfer**
Since DRAM’s inception, there have been a stream of changes to the design, from FPM to EDO to Burst EDO to SDRAM. The changes are largely structural modifications — that target throughput.

Everything up to and including SDRAM has been relatively inexpensive, especially when considering the pay-off (FPM was essentially free, EDO cost a latch, PBEDO cost a counter, SDRAM cost a slight re-design). However, we’re running out of “free” ideas, and now all changes are considered expensive... thus there is no consensus on new directions and myriad of choices has appeared.

Detailed modifications:
- FPM
- EDO
- P/BEDO
- SDRAM
- ESDRAM
- VCDRAM
- MOSYS
- FCRAM
- Rambus, DDR/2

Future trends: MOSYS, FCRAM, VCDRAM, MOSYS.

Diagram: DRAM Evolutionary Tree.

- Conventional DRAM
- (Mostly) Structural Modifications Targeting Throughput
  - FPM
  - EDO
  - P/BEDO
  - SDRAM
- Interface Modifications Targeting Throughput
  - Rambus, DDR/2
- Future Trends
DRAM Evolution

Read Timing for Conventional DRAM

- RAS (Row Access)
- CAS (Column Access)
- Address
- DQ (Dataout)
- Row Address
- Column Address
- Valid Dataout
- Transfer Overlap
- Data Transfer
DRAM Evolution

Read Timing for Fast Page Mode

- **Row Access**
- **Column Access**
- **Transfer Overlap**
- **Data Transfer**

Diagram showing the timing sequence for row access, column address, and data transfer.
solution to that problem -- instead of simple tri-state buffers, use a latch as well.

by putting a latch after the column mux, the next column address command can begin sooner.

Read Timing for Extended Data Out

Row Access
Column Access
Transfer Overlap
Data Transfer
DRAM Evolution

Read Timing for Burst EDO
"pipeline" refers to the setting up of the read pipeline... first CAS toggle latches the column address, all following CAS toggles drive data out onto the bus. Therefore data stops coming when the memory controller stops toggling CAS.

### DRAM Evolution

#### Read Timing for Pipeline Burst EDO

<table>
<thead>
<tr>
<th></th>
<th><strong>RAS</strong></th>
<th><strong>CAS</strong></th>
<th><strong>Address</strong></th>
<th><strong>Column Address</strong></th>
<th><strong>DQ</strong></th>
<th><strong>Valid Data</strong></th>
<th><strong>Valid Data</strong></th>
<th><strong>Valid Data</strong></th>
<th><strong>Valid Data</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Row Access</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Column Access</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Transfer Overlap</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data Transfer</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The main benefit: frees up the CPU or memory controller from having to control the DRAM's internal latches directly. Even though the time-to-first-word latency actually gets worse, the scheme increases system throughput.

(RAS + CAS + OE ... == Command Bus)
DRAM Evolution

Inter-Row Read Timing for ESDRAM

Regular CAS-2 SDRAM, R/R to same bank

ESDRAM, R/R to same bank
DRAM Evolution

Write-Around in ESDRAM

Regular CAS-2 SDRAM, R/W/R to same bank, rows 0/1/0

ESDRAM, R/W/R to same bank, rows 0/1/0

(can second READ be this aggressive?)
The main thing is that it is like having a bunch of open row buffers (a la rambus), but the problem is that you must deal with the cache directly (move into and out of it), not the DRAM banks. This adds an extra couple of cycles of latency. However, you get good bandwidth if the data you want is in the cache, and you can "prefetch" into cache ahead of when you want it.

Originally targeted at reducing latency, now that SDRAM is CAS-2 and RCD-2, this makes sense only in a throughput way.

Segment cache is software-managed, reduces energy.
FCRAM opts to break up the data array, only activate a portion of the word line. An 8K row requires 13 bits to select...

FCRAM uses 15 (assuming the array is 8K x 1K...

the data sheet does not specify)

**DRAM Evolution**

**Internal Structure of Fast Cycle RAM**

SDRAM

- **8M Array** (8Kr x 1Kb)
- **Row Decoder**
- **Sense Amps**
- **t\textsubscript{RCD} = 15\text{ns}** (two clocks)

FCRAM

- **8M Array** (?)
- **Row Decoder**
- **Sense Amps**
- **t\textsubscript{RCD} = 5\text{ns}** (one clock)

Reduces access time and energy/access

\[ RCD = 15\text{ns} \quad \text{(two clocks)} \]

\[ RCD = 5\text{ns} \quad \text{(one clock)} \]
MoSys takes this one step further... DRAM with an SRAM interface & speed but DRAM energy.

[physical partitioning: 72 banks]
Auto refresh -- how to do this transparently? The logic moves through the arrays, refreshing them when not active.

But what if one bank gets repeated access for a long duration? All other banks will be refreshed, but that one will not.

Solution: they have a bank-sized cache of lines... in theory, should never have a problem (magic).
## DRAM Evolution

### Comparison of Low-Latency DRAM Cores

<table>
<thead>
<tr>
<th>DRAM Type</th>
<th>Data Bus Speed</th>
<th>Bus Width (per chip)</th>
<th>Peak BW (per Chip)</th>
<th>RAS–CAS ($t_{RCD}$)</th>
<th>RAS–DQ ($t_{RAC}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC133 SDRAM</td>
<td>133</td>
<td>16</td>
<td>266 MB/s</td>
<td>15 ns</td>
<td>30 ns</td>
</tr>
<tr>
<td>VCDRAM</td>
<td>133</td>
<td>16</td>
<td>266 MB/s</td>
<td>30 ns</td>
<td>45 ns</td>
</tr>
<tr>
<td>FCRAM</td>
<td>200 * 2</td>
<td>16</td>
<td>800 MB/s</td>
<td>5 ns</td>
<td>22 ns</td>
</tr>
<tr>
<td>1T-SRAM</td>
<td>200</td>
<td>32</td>
<td>800 MB/s</td>
<td>—</td>
<td>10 ns</td>
</tr>
<tr>
<td>DDR 266</td>
<td>133 * 2</td>
<td>16</td>
<td>532 MB/s</td>
<td>20 ns</td>
<td>45 ns</td>
</tr>
<tr>
<td>DRDRAM</td>
<td>400 * 2</td>
<td>16</td>
<td>1.6 GB/s</td>
<td>22.5 ns</td>
<td>60 ns</td>
</tr>
<tr>
<td>RLDRAM</td>
<td>300 * 2</td>
<td>32</td>
<td>2.4 GB/s</td>
<td>???</td>
<td>25 ns</td>
</tr>
</tbody>
</table>
Outline

• Basics
• DRAM Evolution: *Structural Path*
• Advanced Basics
• Memory System Details (Lots)
• DRAM Evolution: *Interface Path*
• Future Interface Trends & Research Areas
• Performance Modeling: *Architectures, Systems, Embedded*
What Does This All Mean?

- xDDR II
- DDR II
- netDRAM
- EDO
- RLDRAM
- ESDRAM
- BEDO
- FPM
- SDRAM
- FCRAM
- DDR
- SDRAM
- D-RDRAM
- SDRAM
What is a "good" system?

It's all about the cost of a system. This is a multi-dimensional tradeoff problem. Especially tough when the relative cost factors of pins, die area, and the demands of bandwidth and latency keeps on changing. Good decisions for one generation may not be good for future generations. This is why we don't keep a DRAM protocol for a long time. FPM lasted a while, but we've quickly progressed through EDO, SDRAM, DDR/RDRAM, and now DDR II and whatever else is on the horizon.

![Cost-Benefit Criterion Diagram]

- Package Cost
- Interconnect Cost
- Logic Overhead
- Test and Implementation
- Power Consumption

DRAM System Design

Bandwidth

Latency

Cost - Benefit Criterion
Now we'll really get our hands dirty, and try to become DRAM designers. That is, we want to understand the tradeoffs, and design our own memory system with DRAM cells. By doing this, we can gain some insight into some of the basis of claims by proponents of various DRAM memory systems.

A Memory System is a system that has many parts. It's a set of technologies and design decisions. All of the parts are inter-related, but for the sake of discussion, we'll split the components into the areas seen here, and try to examine each part of a DRAM system separately.

Memory System Design

- Clock Network
- I/O Technology
- Topology
- Chip Packaging
- DRAM Chip Architecture
- Pin Count
- Access Protocol
- Address Mapping
- Row Buffer Management
- DRAM Memory System

[Diagram showing the inter-relationships between the components of a memory system]
DRAM Interfaces

The Digital Fantasy

Pretend that the world looks like this

But...
The Real World

*Toshiba Presentation, Denali MemCon 2002
First, we have to introduce the concept that signal propagation takes finite time. Limited by the speed of light, or rather ideal transmission lines we should have speed of approximately 2/3 the speed of light. That gets us 20 cm/ns. All signals, including system-wide clock signals, have to be sent on a system board, so if you sent a clock signal from point A to point B on an ideal signal line, point B won't be able to tell that the clock has changed until at the earliest, 1/20 ns/cm * distance later that the clock has risen.

Then again, PC boards are not exactly ideal transmission lines. (ringing effect, drive strength, etc)

The concept of "Synchronous" breaks down when different parts of the system observe different clocks. Kind of like relativity.

Signal Propagation

Ideal Transmission Line

\[ \sim 0.66c = 20 \text{ cm/ns} \]

PC Board + Module Connectors + Varying Electrical Loads

= Rather non-Ideal Transmission Line
When we build a "synchronous system" on a PCB board, how do we distribute the clock signal? Do we want a sliding time domain? Is H Tree doable to N-modules in parallel? Skew compensation?

What Kind of Clocking System?
Clocking Issues

We need different “clocks” for R/W
We purposely "routed path #2 to be a bit longer than path #1 to illustrate the point between the signal path length differentials. As illustrated, signals will reach load B at a later time than load A simply because it is farther away from the controller than load A.

It is also difficult to do path length and impedance matching on a system board. Sometimes heroic efforts must be utilized to get us a nice "parallel" bus.

High Frequency AND Wide Parallel Busses are Difficult to Implement.
It’s hard to bring the Wide parallel bus from point A to point B, but it’s easier to bring in smaller groups of signals from A to B. To ensure proper timing, we also send along a source synchronous clock signal that is path length matched with the signal group it covers. In this figure, signal groups 1, 2, and 3 may have some timing skew with respect to each other, but within the group the signals will have minimal skew. (smaller channel can be clocked higher.)

Subdividing Wide Busses

Narrow Channels, Source Synchronous Local Clock Signals
Why Subdivision Helps

Worst Case Skew must be Considered in System Timing
How many DIMMs in System?
How many devices on each DIMM?
Who built the memory module?

Infinite variations on timing!
To ensure that a lightly loaded system and a fully loaded system do not differ significantly in timing, we either have duplicate signals sent to different memory modules, or we have the same signal line, but the signal line uses variable strengths to drive the I/O pads, depending on if the system has 1, 2, 3 or 4 loads.
DRAM System Topology Determines Electrical Loading Conditions and Signal Propagation Lengths
Very simple topology. The clock signal that turns around is very nice. Solves problem of needing multiple clocks.

**SDRAM Topology Example**

**Loading Imbalance**
All signals in this topology, Addr/Cmd/Data/Clock, are sent from point to point on channels that is path length matched by definition.

RDRAM Topology Example

Packets traveling down parallel paths. Skew is minimal by design.

Clock turns around
What is "Logic Low", what is "Logic High"? Different Electrical Signalling protocols differ on voltages, high/low levels, etc.

δt is on the order of ns, we want it to be on the order of ps.

I/O Technology

Slew Rate = \( \frac{\Delta v}{\Delta t} \)

Smaller \( \Delta v \) = Smaller \( \Delta t \) at same slew rate

Increase Rate of bits/s/pin
Used on clocking systems, i.e., RDRAM (all clock signals are pairs, clk and clk#).

Highest noise tolerance, does not need as many ground signals. Whereas single-ended signals need many ground connections. Also differential pair signals may be clocked even higher, so pin-bandwidth disadvantage is not nearly 2:1 as implied by the diagram.

I/O - Differential Pair

Single Ended Transmission Line

Differential Pair Transmission Line

Increase Rate of bits/s/pin?
Cost Per Pin?
Pin Count?
One of several ways on the table to further increase the bit rate of the interconnects.

I/O - Multi Level Logic

Increase Rate of bits/s/pin
Different packaging types impact costs and speed. Slow parts can use the cheapest packaging available. Faster parts may have to use more expensive packaging. This has long been accepted in the higher margin processor world, but to DRAM, each cent has to be hard fought for. To some extent, the demand for higher performance is pushing memory makers to use more expensive packaging to accommodate higher frequency parts.

When Rambus first specified FBGA, module makers complained, since they have to purchase expensive equipment to validate that chips were properly soldered to the module board, whereas something like TSOP can be done with visual inspection.

<table>
<thead>
<tr>
<th>Features</th>
<th>Target Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td>FBGA</td>
</tr>
<tr>
<td>Speed</td>
<td>800MBps</td>
</tr>
<tr>
<td>Vdd/Vddq</td>
<td>2.5V/2.5V (1.8V)</td>
</tr>
<tr>
<td>Interface</td>
<td>SSTL_2</td>
</tr>
<tr>
<td>Row Cycle Time</td>
<td>35ns</td>
</tr>
</tbody>
</table>

Memory Roadmap for Hynix NetDDR II
16 bit wide command I have to send from A to B, I need 16 pins, or if I have less than 16, I need multiple cycles.

How many bits do I need to send from point A to point B? How many pins do I get?

Cycles = Bits/Pins.

Access Protocol

Single Cycle Command

Multiple Cycle Command
There is inherent latency between issuance of a read command, and the response of the chip with data. To increase efficiency, a pipeline structure is necessary to obtain full utilization of the command, address and data busses.

Different from an "ordinary" pipeline on a processor, a memory pipeline has data flowing in both directions.

Architecture wise, we should be concerned with full utilization everywhere, so we can use the least number of pins for the greatest benefit, but in actual use, we are usually concerned with full utilization of the data bus.

Access Protocol (r/r)

**Consecutive Cache Line Read Requests to Same DRAM Row**

- **a** = Active (open page)
- **r** = Read (Column Read)
- **d** = Data (Data chunk)
The DRAM chips determine the latency of data after a read command is received, but the controller determines the timing relationship between the write command and the data being written to the DRAM chips.

Case 1: Controller sends write data at the same time as the write command to different devices (pipelined).

Case 2: Controller sends write data at the same time as the write command to the same device (not pipelined).

Access Protocol (r/w)

One Datapath - Two Commands

Col w0 r1

Data d0 d0 d0 d1 d1 d1 d1

Case 1: Read Following a Write Command to Different DRAM Devices

Soln: Delay Data of Write Command to match Read Latency
To increase "efficiency", pipelines is required. How many commands must one device support concurrently? 2? 3? 4? (depends on what?)

Imagine we must increase data rate (higher pin freq), but allow DRAM core to operate slightly slower. (2X pin freq., same core latency)

This issue ties access protocol to internal DRAM architecture issues.

Access Protocol (pipelines)

Three Back-to-Back Pipelined Read Commands

"Same" Latency, 2X pin frequency, Deeper Pipeline

When pin frequency increases, chips must either reduce "real latency", or support longer bursts, or pipeline more commands.
Outline

• Basics
• DRAM Evolution: *Structural Path*
• Advanced Basics
• DRAM Evolution: *Interface Path*
• SDRAM, DDR SDRAM, RDRAM Memory System Comparisons
• Processor-Memory System Trends
• RLDRAM, FCRAM, DDR II Memory Systems Summary
• Future Interface Trends & Research Areas
• Performance Modeling: *Architectures, Systems, Embedded*
SDRAM System In Detail

"Mesh Topology"
SDRAM Chip

133 MHz (7.5ns cycle time)

Multiplexed Command/Address Bus

Programmable Burst Length, 1, 2, 4 or 8

Quad Banks Internally

Supply Voltage of 3.3V

Low Latency, CAS = 2, 3

LVTTL Signaling (0.8V to 2.0V)
(0 to 3.3V rail to rail.)

<table>
<thead>
<tr>
<th>Condition Specification</th>
<th>Cur.</th>
<th>Pwr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating (Active) Burst = Continous</td>
<td>300mA</td>
<td>1W</td>
</tr>
<tr>
<td>Operating (Active) Burst = 2</td>
<td>170mA</td>
<td>560mW</td>
</tr>
<tr>
<td>Standby (Active) All banks active</td>
<td>60mA</td>
<td>200mW</td>
</tr>
<tr>
<td>Standby (powerdown) All banks inactive</td>
<td>2mA</td>
<td>6.6mW</td>
</tr>
</tbody>
</table>
We've spent some time discussing some pipelined back-to-back read commands sent to the same chip, now let's try to pipeline commands to different chips.

In order for the memory controller to be able to latch in data on the data bus on consecutive cycles, chip #0 has to hold the data value past the rising edge of the clock to satisfy the hold time requirements, then chip #0 has to stop, allow the bus to go "quiet", then chip #1 can start to drive the data bus at least some "setup time" ahead of the rising edge of the next clock.

Clock cycles have to be long enough to tolerate all of the timing requirements.

Clock Cycles are still long enough to allow for pipelined back-to-back Reads.
SHOW different paths, but these signals are sharing the bi-directional data bus. For a read to follow a write to a different chip, the worst case skew is when we write to the (N-1)th chip, then expect to pipeline a read command in the next cycle right behind it. The worst case signal path skew is the sum of the distances. Isn't from N to N even worse? No, SDRAM does not support pipelined read behind a write on the same chip. Also, it's not as bad as I project here, since read cycles are center aligned, and writes are edge aligned, so in essence, we get 1 1/2 cycles to pipeline this case, instead of just 1 cycle.

Still, this problem limits the freq scalable of SDRAM, and idle cycles may be inserted to meet timing.

Looks just like SDRAM!

SDRAM Access Protocol (w/r)

Figure 1: Consecutive Reads

0th • • • Nth

Worst case = Dist(N) - Dist(0)

Bus Turn Around

Figure 1: Read After Write

0th • (N-1)th Nth

Worst case = Dist(N) + Dist(N-1)
SDRAM Access Protocol (w/r)

Read Following a Write Command to Same SDRAM Device
Since data bus has a much lighter load, if we can use better signaling technology, perhaps we can run just the data bus at a higher frequency. At the higher frequency, the skew we talked about would be terrible with a 64 bit wide data bus. So we use a source synchronous strobe signals (called DQS) that is routed parallel to each 8 bit wide sub-channel.

DDR is newer, so let’s use lower core voltage, saves on power too!
DDR SDRAM Chip

133 MHz (7.5ns cycle time)
Multiplexed Command/Address Bus
Programmable Burst Lengths, 2, 4 or 8*
Quad Banks Internally
Supply Voltage of 2.5V*
Low Latency, CAS = 2, 2.5, 3 *
SSTL-2 Signaling (Vref +/- 0.15V)
(0 to 2.5V rail to rail)
Here we see that two consecutive column read commands to different chips on the DDR memory channel cannot be placed back to back on the Data bus due to the DQS signal hand-off issue. They may be pipelined with one idle cycle in between bursts.

This situation is true for all consecutive accesses to different chips, r/r, r/w, w/r. (except w/w, when the controller keeps control of the DQS signal, just changes target chips)

Because of this overhead, short bursts are inefficient on DDR, longer bursts are more efficient. (32-byte cache line = 4 burst, 64-byte line = 8 burst)
Very different from SDRAM. Everything is sent around in 8 (half cycle) packets. Most systems now run at 400 MHz, but since everything is DDR, it’s called “800 MHz”. The only difference is that packets can only be initiated at the rising edge of the clock, other than that, there’s no difference between 400 DDR and 800.

Very clean topology, very clever clocking scheme. No clock handoff issue, high efficiency.

Write delay improves matching with read latency. (not perfect, as shown) since data bus is 16 bits wide, each read command gets 16*8=128 bits back. Each cacheline fetch = multiple packets.

Up to 32 devices.

Packet Protocol: Everything in 8 (half) cycle packets
Direct RDRAM Chip

400 MHz (2.5ns cycle time)
Separate Row-Col Command Busses
Burst Length = 8*
4/16/32 Banks Internally*
Supply Voltage of 2.5V*
Low Latency, CAS = 4 to 6 full cycles*
RSL Signaling (Vref +/- 0.2V)
(800 mV rail to rail)

All packets are 8 (half) cycles in length,
the protocol allows near 100% bandwidth
utilization on all channels. (Addr/Cmd/Data)
RDRAM Drawbacks

- High Frequency I/O Test and Package Cost
- RSL: Separate Power Plane
- 30% die cost for logic @ 64 Mbit node
- Active Decode Logic + Open Row Buffer. (High power for “quiet” state)
- Single Chip Provides All Data Bits for Each Packet (Power)

Significant Cost Delta for First Generation
## System Comparison

<table>
<thead>
<tr>
<th></th>
<th>SDRAM</th>
<th>DDR</th>
<th>RDRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency (MHz)</td>
<td>133</td>
<td>133*2</td>
<td>400*2</td>
</tr>
<tr>
<td>Pin Count (Data Bus)</td>
<td>64</td>
<td>64</td>
<td>16</td>
</tr>
<tr>
<td>Pin Count (Controller)</td>
<td>102</td>
<td>101</td>
<td>33</td>
</tr>
<tr>
<td>Theoretical Bandwidth (MB/s)</td>
<td>1064</td>
<td>2128</td>
<td>1600</td>
</tr>
<tr>
<td>Theoretical Efficiency (data bits/cycle/pin)</td>
<td>0.63</td>
<td>0.63</td>
<td>0.48</td>
</tr>
<tr>
<td>Sustained BW (MB/s)*</td>
<td>655</td>
<td>986</td>
<td>1072</td>
</tr>
<tr>
<td>Sustained Efficiency* (data bits/cycle/pin)</td>
<td>0.39</td>
<td>0.29</td>
<td>0.32</td>
</tr>
<tr>
<td>RAS + CAS (t_{RAC}) (ns)</td>
<td>45 ~ 50</td>
<td>45 ~ 50</td>
<td>57 ~ 67</td>
</tr>
<tr>
<td>CAS Latency (ns)**</td>
<td>22 ~ 30</td>
<td>22 ~ 30</td>
<td>40 ~ 50</td>
</tr>
</tbody>
</table>

133 MHz P6 Chipset + SDRAM CAS Latency ~ 80 ns

*StreamAdd

**Load to use latency
Differences of Philosophy

SDRAM - Variants
- Controller
- Complex Interconnect
- Inexpensive Interface
- DRAM Chips

RDRAM - Variants
- Controller
- Simplified Interconnect
- expensive Interface
- Complex Logic
- DRAM Chips

Complexity Moved to DRAM
To begin with, we look in a crystal ball to look for trends that will cause changes or limit scalability in areas that we are interested in.

ITRS = International Technology Roadmap for Semiconductors. Transistor frequencies are supposed to nearly double every generation, and transistor budget (as indicated by Million Logic Transistors per cm^2) are projected to double.

Interconnects between chips are a different story. Measured in cents/pin, pin cost decreases only slowly, and pin budget grows slowly each generation.

Punchline: In the future, Free Transistors and Costly Interconnects.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Semi Generation (nm)</td>
<td>90</td>
<td>65</td>
<td>45</td>
<td>32</td>
<td>22</td>
</tr>
<tr>
<td>CPU MHz</td>
<td>3990</td>
<td>6740</td>
<td>12000</td>
<td>19000</td>
<td>29000</td>
</tr>
<tr>
<td>MLogicTransistors/ cm(^2)</td>
<td>77.2</td>
<td>154.3</td>
<td>309</td>
<td>617</td>
<td>1235</td>
</tr>
<tr>
<td>High Perf chip pin count</td>
<td>2263</td>
<td>3012</td>
<td>4009</td>
<td>5335</td>
<td>7100</td>
</tr>
<tr>
<td>High Performance chip cost (cents/pin)</td>
<td>1.88</td>
<td>1.61</td>
<td>1.68</td>
<td>1.44</td>
<td>1.22</td>
</tr>
<tr>
<td>Memory pin cost (cents/pin)</td>
<td>0.34 - 1.39</td>
<td>0.27 - 0.84</td>
<td>0.22 - 0.34</td>
<td>0.19 - 0.39</td>
<td>0.19 - 0.33</td>
</tr>
<tr>
<td>Memory pin count</td>
<td>48-160</td>
<td>48-160</td>
<td>62-208</td>
<td>81-270</td>
<td>105-351</td>
</tr>
</tbody>
</table>

Trend: Free Transistors & Costly Interconnects
Choices for Future

- Direct Connect
  - Custom DRAM: Highest Bandwidth + Low Latency

- Direct Connect
  - semi-comm. DRAM: High Bandwidth + Low/Moderate Latency

- Direct Connect
  - Commodity DRAM: Low Bandwidth + Low Latency

- Indirect Connection
  - Inexpensive DRAM
  - Highest Latency
  - Highest Bandwidth
EV7 + RDRAM (Compaq/HP)

- RDRAM Memory (2 Controllers)
- Direct Connection to processor
- 75ns Load to use latency
- 12.8 GB/s Peak bandwidth
- 6 GB/s read or write bandwidth
- 2048 open pages (2 * 32 * 32)

Each column read fetches 128 * 4 = 512 b (data)
What if EV7 Used DDR?

- **Peak Bandwidth 12.8 GB/s**
- **6 Channels of 133*2 MHz DDR SDRAM**
- **6 Controllers of 6 64 bit wide channels, or**
- **3 Controllers of 3 128 bit wide channels**

<table>
<thead>
<tr>
<th>System</th>
<th>EV7 + RDRAM</th>
<th>EV7 + 6 controller DDR SDRAM</th>
<th>EV7 + 3 controller DDR SDRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency</td>
<td>75 ns</td>
<td>~ 50 ns*</td>
<td>~ 50 ns*</td>
</tr>
<tr>
<td>Pin count</td>
<td>~265** + Pwr/Gnd</td>
<td>~ 600** + Pwr/Gnd</td>
<td>~ 600** + Pwr/Gnd</td>
</tr>
<tr>
<td>Controller Count</td>
<td>2</td>
<td>6***</td>
<td>3***</td>
</tr>
<tr>
<td>Open pages</td>
<td>2048</td>
<td>144</td>
<td>72</td>
</tr>
</tbody>
</table>

* page hit CAS + memory controller latency.
** including all signals, address, command, data, clock, not including ECC or parity
*** 3 controller design is less bandwidth efficient.
What’s Next?

- DDR II
- FCRAM
- RLDRAM
- RDRAM (Yellowstone etc)
- Kentron QBM
DDR II - DDR Next Gen

- Lower I/O Voltage (1.8V)
- DRAM core operates at 1:4 freq of data bus freq (SDRAM 1:1, DDR 1:2)
- 400 Mbps - multidrop
- 800 Mbps - point to point
- FPBGA package
- Burst Length == 4 Only!
- 4 Banks internally (same as SDRAM and DDR)
- Backward Comp. to DDR (Common modules possible)
- No more Page-Transfer-Until-Interrupted Commands (removes speedpath)
- Write Latency = CAS -1 (increased Bus Utilization)
Instead of a controller that keeps track of cycles, we can now have a “dumber” controller. Control is now simple, kind of like SRAM. Part I of address one cycle, part II the next cycle.

SDRAM & DDR SDRAM relies on memory controller to know $t_{RCD}$ and issue CAS after $t_{RCD}$ for lowest latency.

Internal counter delays CAS command, DRAM chip issues “real” command after $t_{RCD}$ for lowest latency.
FCRAM

Fast Cycle RAM (aka Network-DRAM)

<table>
<thead>
<tr>
<th>Features</th>
<th>DDR SDRAM</th>
<th>FCRAM/Network-DRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vdd, Vddq</td>
<td>2.5 +/- 0.2V</td>
<td>2.5 +/- 0.15</td>
</tr>
<tr>
<td>Electrical Interface</td>
<td>SSTL-2</td>
<td>SSTL-2</td>
</tr>
<tr>
<td>Clock Frequency</td>
<td>100~167 MHz</td>
<td>154~200 MHz</td>
</tr>
<tr>
<td>t&lt;sub&gt;RAC&lt;/sub&gt;</td>
<td>~40ns</td>
<td>22~26ns</td>
</tr>
<tr>
<td>t&lt;sub&gt;RC&lt;/sub&gt;</td>
<td>~60ns</td>
<td>25~30ns</td>
</tr>
<tr>
<td># Banks</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Burst Length</td>
<td>2,4,8</td>
<td>2,4</td>
</tr>
<tr>
<td>Write Latency</td>
<td>1 Clock</td>
<td>CASL -1</td>
</tr>
</tbody>
</table>

FCRAM/Network-DRAM looks like DDR+
Faster \( t_{RC} \) allows Samsung to claim higher bus efficiency

* Samsung Electronics, Denali MemCon 2002
Another Variant, but RLDRAM is targeted to toward embedded systems. There are no connector specifications, so it can target a higher frequency off the bat.

### RLDRAM

<table>
<thead>
<tr>
<th>DRAM Type</th>
<th>Frequency</th>
<th>Bus Width (per chip)</th>
<th>Peak Bandwidth (per Chip)</th>
<th>Random Access Time ($t_{RAC}$)</th>
<th>Row Cycle Time ($t_{RC}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC133 SDRAM</td>
<td>133</td>
<td>16</td>
<td>200 MB/s</td>
<td>45 ns</td>
<td>60 ns</td>
</tr>
<tr>
<td>DDR 266</td>
<td>133 * 2</td>
<td>16</td>
<td>532 MB/s</td>
<td>45 ns</td>
<td>60 ns</td>
</tr>
<tr>
<td>PC800 RDRAM</td>
<td>400 * 2</td>
<td>16</td>
<td>1.6 GB/s</td>
<td>60 ns</td>
<td>70 ns</td>
</tr>
<tr>
<td>FCRAM</td>
<td>200 * 2</td>
<td>16</td>
<td>0.8 GB/s</td>
<td>25 ns</td>
<td>25 ns</td>
</tr>
<tr>
<td>RLDRAM</td>
<td>300 * 2</td>
<td>32</td>
<td>2.4 GB/s</td>
<td>25 ns</td>
<td>25 ns</td>
</tr>
</tbody>
</table>

Comparable to FCRAM in latency
Higher Frequency (No Connectors)
non-Multiplexed Address (SRAM like)
RLDRAM is a great replacement to SRAM in L3 cache applications because of its high density, low power and low cost.

* Infineon Presentation, Denali MemCon 2002
RAMBUS Yellowstone

- Bi-Directional Differential Signals
- Ultra low 200mV p-p signal swings
- 8 data bits transferred per clock
- 400 MHz system clock
- 3.2 GHz effective data frequency
- Cheap 4 layer PCB
- Commodity packaging

Octal Data Rate (ODR) Signaling

System Clock

Data
Quad Band Memory

Uses Fetswitches to control which DIMM sends output.

Two DDR memory chips are interleaved to get Quad memory.

Advantages, uses standard DDR chips, extra cost is low, only the wrapper electronics.

Modification to memory controller required, but minimal.

Has to understand that data is being burst back at 4X clock frequency. Does not improve efficiency, but cheap bandwidth.

Supports more loads than "ordinary DDR", so more capacity.

"Wrapper Electronics around DDR memory"

Generates 4 data bits per cycle instead of 2.

Quad Band Memory
A Different Perspective

Everything is bandwidth

Latency and Bandwidth

Pin-bandwidth and

Pin-transition *Efficiency (bits/cycle/sec)
DRAM TUTORIAL

ISCA 2002

Bruce Jacob
David Wang

University of Maryland

Research Areas: Topology

Unidirectional Topology:

- Write Packets sent on Command Bus
- Pins used for Command/Address/Data
- Further Increase of Logic on DRAM chips
Memory Commands?

Instead of $A[ ] = 0$; Do “write 0”

Why do $A[ ] = B[ ]$ in CPU?

Move Data inside of DRAM or between DRAMs.

Why do STREAMadd in CPU?

$A[ ] = B[ ] + C[ ]$

Active Pages *(Chong et. al. ISCA ‘98)
For a given physical address, there are a number of ways to map the bits of the physical address to generate the "memory address" in terms of device ID, row/cell address, and bank ID.

The mapping policies could impact performance, since badly mapped systems can cause bank conflicts in consecutive accesses.

Now, mapping policies must also take temperature control into account, as consecutive accesses that hit the same DRAM chip can potentially create undesirable hot spots.

One reason for the additional cost of RDRAM initially was the use of heat spreaders on the memory modules to prevent the hotspots from building up.

Address Mapping

Access Distribution for Temp Control
Avoid Bank Conflicts
Access Reordering for performance
Each Memory system consists of one or more memory chips, and most times, accesses to these chips can be pipelined. Each chip also has multitudes of banks, and most of the times, accesses to these banks can also be pipelined. (key to efficiency is to pipeline commands).

Example: Bank Conflicts

- Multiple Banks to Reduce Access Conflicts
- Read 05AE5700 → Device id 3, Row id 266, Bank id 0
- Read 023BB880 → Device id 3, Row id 1BA, Bank id 0
- Read 05AE5780 → Device id 3, Row id 266, Bank id 0
- Read 00CBA2C0 → Device id 3, Row id 052, Bank id 1

More Banks per Chip == Performance == Logic Overhead
Example: Access Reordering

1. Read 05AE5700 → Device id 3, Row id 266, Bank id 0
2. Read 023BB880 → Device id 3, Row id 1BA, Bank id 0
3. Read 05AE5780 → Device id 3, Row id 266, Bank id 0
4. Read 00CBA2C0 → Device id 1, Row id 052, Bank id 1

Act = Activate Page (Data moved from DRAM cells to row buffer)
Read = Read Data (Data moved from row buffer to memory controller)
Prec = Precharge (close page/evict data in row buffer/sense amp)
Outline

• Basics
• DRAM Evolution: Structural Path
• Advanced Basics
• DRAM Evolution: Interface Path
• Future Interface Trends & Research Areas
• Performance Modeling: Architectures, Systems, Embedded
Simulator Overview

CPU: SimpleScalar v3.0a
- 8-way out-of-order
- L1 cache: split 64K/64K, lockup free x32
- L2 cache: unified 1MB, lockup free x1
- L2 blocksize: 128 bytes

Main Memory: 8 64Mb DRAMs
- 100MHz/128-bit memory bus
- Optimistic open-page policy

Benchmarks: SPEC ’95
DRAM Configurations

**FPM, EDO, SDRAM, ESDRAM, DDR:**

- CPU and caches
- 128-bit 100MHz bus
- Memory Controller

**Rambus, Direct Rambus, SLDRAM:**

- CPU and caches
- 128-bit 100MHz bus
- Memory Controller

**DIMM**

Note: TRANSFER WIDTH of Direct Rambus Channel

- equals that of ganged FPM, EDO, etc.
- is 2x that of Rambus & SLDRAM
DRAM Configurations

Rambus & SLDRAM dual-channel:

Strawman: Rambus, etc.

Multiple Parallel Channels
**First ... Refresh Matters**

Assumes refresh of each bank every 64ms
Overhead: Memory vs. CPU

Total Execution Time in CPI — SDRAM

- Stalls due to Memory Access Time
- Overlap between Execution & Memory
- Processor Execution (includes caches)

Variable: speed of processor & caches
Definitions (var. on Burger, et al)

- $t_{\text{PROC}}$ — processor with perfect memory
- $t_{\text{REAL}}$ — realistic configuration
- $t_{\text{BW}}$ — CPU with wide memory paths
- $t_{\text{DRAM}}$ — time seen by DRAM system
**Memory & CPU — PERL**

**Bandwidth-Enhancing Techniques I:**

- Stalls due to Memory Bandwidth
- Stalls due to Memory Latency
- Overlap between Execution & Memory
- Processor Execution

---

**DRAM Configuration**

Yesterday's CPU

Tomorrow's CPU

Today's CPU

Newer DRAMs

Tomorrow's CPU

Newer DRAMs

Updated DRAMs

**Cycles Per Instruction (CPI)**

- FPM
- EDO
- SDRAM
- RDRAM
- SDRAM
- DRDRAM
- ESDRAM
- DDR

---

**University of Maryland**

**Bruce Jacob**

**David Wang**

**DRAM TUTORIAL**

**ISCA 2002**
Memory & CPU — PERL

Bandwidth-Enhancing Techniques II:

- Stalls due to Memory Bandwidth
- Stalls due to Memory Latency
- Overlap between Execution & Memory
- Processor Execution

Cycles Per Instruction (CPI)

DRAM Configuration (10GHz CPUs)
Average Latency of DRAMs

note: SLDRAM & RDRAM 2x data transfers
Average Latency of DRAMs

note: SLDRAM & RDRAM 2x data transfers
DDR2 Study Results

Architectural Comparison

Normalized Execution Time (DDR2)

Benchmark

cc1  com  go  jpeg  li  lineawalk  mpg  mpg2c  mpg2en  pgw  perl  rando  stream  streamno

pc100  ddr133  drd  ddr2  ddr2ems  ddr2vc
**DDR2 Study Results**

**Perl Runtime**

![Graph showing execution time for different processor frequencies and DRAM types (pc100, ddr133, ddr, ddr2, ddr2ems, ddr2vc) at 1 Ghz, 5 Ghz, and 10 Ghz.](image-url)
Row-Buffer Hit Rates

No L2 Cache

1MB L2 Cache

4MB Cache

FPMDRAM
EDODRAM
SDRAM
ESDRAM
DDRSDRAM
SLDRAM
RDGRAM
DRDRAM
Row-Buffer Hit Rates

Hits vs. Depth in Victim-Row FIFO Buffer

Inter-arrival time (CPU Clocks)

Inter-arrival time (CPU Clocks)
Row Buffers as L2 Cache

- Stalls due to Memory Bandwidth
- Stalls due to Memory Latency
- Overlap between Execution & Memory
- Processor Execution

Clocks Per Instruction (CPI)

- Compress
- Gcc
- Go
- Ijpeg
- Li
- M88ksim
- Perl
- Vortex
Each memory transaction has to break down into a two part access, a row access and a column access. In essence the row buffer/sense amplifiers act as a cache. Where a page is brought in from the memory array and stored in the buffer, then the second step is to move that data from the row buffers back into the memory controller. From a certain perspective, it makes sense to speculatively move pages from memory arrays into the row buffers to maximize the page hit rate of a column access, and reduce latency. The cost of a speculative row activation command is the ~20 bit of bandwidth sent on the command channel from controller to DRAM. Instead of prefetching into DRAM, we’re just prefetching inside of DRAM. Row buffer hit rates 40~90%, depending on application. *Could* be near 100% if memory system gets speculative row buffer management commands. (This only makes sense if memory controller is integrated.)

RAS is like Cache Access
Why not Speculate?

---

Row Buffer Management

ROW ACCESS

- Data In/Out Buffers
- Column Decoder
- Sense Amps
- Memory Array
- RAS

COLUMN ACCESS

- Data In/Out Buffers
- Column Decoder
- Sense Amps
- Memory Array
- CAS
Cost-Performance

FPM, EDO, SDRAM, ESDRAM:
- Lower Latency => Wide/Fast Bus
- Increase Capacity => Decrease Latency
- Low System Cost

Rambus, Direct Rambus, SLDRAM:
- Lower Latency => Multiple Channels
- Increase Capacity => Increase Capacity
- High System Cost

However, 1 DRDRAM = Multiple SDRAM
Conclusions

100MHz/128-bit Bus is Current Bottleneck

- Solution: Fast Bus/es & MC on CPU (e.g. Alpha 21364, Emotion Engine, …)

Current DRAMs Solving Bandwidth Problem (but not Latency Problem)

- Solution: New cores with on-chip SRAM (e.g. ESDRAM, VCDRAM, …)
- Solution: New cores with smaller banks (e.g. MoSys “SRAM”, FCRAM, …)

Direct Rambus seems to scale best for future high-speed CPUs
now -- let’s talk about DRAM performance at the system level.

Previous studies show Memory Bus is a significant bottleneck in today’s high-performance systems. Schumann reports that in Alpha workstations, 30-60% of Primary Memory Latency, is due to System Overhead other than DRAM latency. Harvard study cites Bus Turnaround as responsible for factor-of-two difference between Predicted and Measured performance in P6 systems. Our previous work shows today's buses (1999's buses) are bottlenecks for tomorrow's DRAMs. So -- look at bus, model system overhead.

Outline

• Basics
• DRAM Evolution: Structural Path
• Advanced Basics
• DRAM Evolution: Interface Path
• Future Interface Trends & Research Areas
• Performance Modeling:
  Architectures, Systems, Embedded
Motivation

Even when we restrict our focus ...

SYSTEM-LEVEL PARAMETERS

- Number of channels
- Channel latency
- Banks per channel
- Request-queue size
- Row-access
- DRAM precharge
- DRAM buffering
- Number of MSHRs
- Width of channels
- Channel bandwidth
- Turnaround time
- Request reordering
- Column-access
- CAS-to-CAS latency
- L2 cache blocksize
- Bus protocol

Fully | partially | not independent (this study)
Motivation

... the design space is highly non-linear ...

![Graph showing cycles per instruction (CPI) vs. system bandwidth for different burst lengths and channel configurations.](image)

- **32-Byte Burst**
- **64-Byte Burst**
- **128-Byte Burst**

System Bandwidth

(GB/s = Channels * Width * 800MHz)
so we have the worst possible scenario: a design space that is very sensitive to changes in parameters and execution times that can vary by a factor of three from worst-case to best. Clearly, we would be well-served to understand this design space... and the cost of poor judgment is high.
so by now we're very familiar with this picture ...

we cannot use it in this study, because this represents the interface between the DRAM and the MEMORY CONTROLLER.

typically, the CPU's interface is much simpler: the CPU sends all of the address bits at once with CONTROL INFO (r/w), and the memory controller handles the bit addressing and the RAS/CAS timing.
System-Level Model

Timing diagrams are at the DRAM level
... not the system level

Diagram showing synchronization between clock, command, address, data, and various active states such as DRAM Bank Active, ABUS Active, DBUS Active, Row Access, Column Access, and Data Transfer.
System-Level Model

Timing diagrams are at the DRAM level ...
... not the system level
so let's formalize this system-level interface. Here's the request timing in a slightly different way, as well as an example system model taken from Schumann's paper describing the 21174 memory controller.

The DRAM's data pins are connected directly to the CPU (simplest possible model), the memory controller handles the RAS/CAS timing, and the CPU and memory controller only talk in terms of addresses and control information.

**Request Timing**

```
<table>
<thead>
<tr>
<th>Backside bus</th>
<th>Frontside bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache</td>
<td>CPU</td>
</tr>
<tr>
<td>Data bus</td>
<td>Data bus (800 MHz)</td>
</tr>
<tr>
<td>Address</td>
<td>Address (800 MHz)</td>
</tr>
<tr>
<td>Control</td>
<td>Control</td>
</tr>
</tbody>
</table>

Row/Column Addresses & Control

READ REQUEST TIMING:

\[ t_0 \]

ADDRESS BUS

DRAM BANK

DATA BUS

<ROW> <COL> <PRE>

<DB0><DB1><DB2><DB3>
Read/Write Request Shapes

READ REQUESTS:

ADDRESS BUS
DRAM BANK
DATA BUS

WRITE REQUESTS:
Pipelined/Split Transactions

(a) Legal if R/R to different banks:

(b) Nestling of writes inside reads is legal if R/W to different banks:
Legal if turnaround <= 10ns:
Legal if no turnaround:

(c) Back-to-back R/W pair that cannot be nestled:
Channels & Banks

<table>
<thead>
<tr>
<th>Channels &amp; Banks</th>
<th>One independent channel</th>
<th>Two independent channels</th>
<th>Four independent channels</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Banking degrees of 1, 2, 4, ...</td>
<td>Banking degrees of 1, 2, 4, ...</td>
<td>Banking degrees of 1, 2, 4, ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1, 2, 4  800 MHz Channels
8, 16, 32, 64  Data Bits per Channel
1, 2, 4, 8  Banks per Channel (Indep.)
32, 64, 128  Bytes per Burst
Burst Scheduling
(Back-to-Back Read Requests)

128-Byte Bursts:

64-Byte Bursts:

32-Byte Bursts:

- Critical-burst-first
- Non-critical bursts are promoted
- Writes have lowest priority
  (tend back up in request queue …)
- Tension between large & small bursts:
  amortization vs. faster time to data
New Bar-Chart Definition

- $t_{PROC}$ — CPU with 1-cycle L2 miss
- $t_{REAL}$ — realistic CPU/DRAM config
- $t_{SYS}$ — CPU with 1-cycle DRAM latency
- $t_{DRAM}$ — time seen by DRAM system
so—we’re modeling a memory system that is fairly aggressive in terms of:
- scheduling policies
- support for concurrency

And we’re trying to find which of the following is to blame for the most overhead:
- concurrency
- latency
- system (queueing, precharge, chunks, etc)

### System Overhead

<table>
<thead>
<tr>
<th>Bandwidth</th>
<th>Cycles per Instruction (CPI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bank/channel</td>
<td>1.6</td>
</tr>
<tr>
<td>2 banks/channel</td>
<td>3.2</td>
</tr>
<tr>
<td>4 banks/channel</td>
<td>6.4</td>
</tr>
<tr>
<td>8 banks/channel</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**System Bandwidth**
(GB/s = Channels * Width * Speed)

**Benchmark** = BZIP (SPEC 2000), 32-byte burst, 16-bit bus
The figure shows:

- System overhead is significant (usually 20-40% of the total memory overhead).
- The most significant overhead tends to be the DRAM latency.
- Turnaround is relatively insignificant (however, remember that this is an 800MHz bus system...).

System overhead 10–100% over perfect memory.

Benchmark = BZIP (SPEC 2000), 32-byte burst, 16-bit bus.
Concurrency Effects

Benchmark = BZIP (SPEC 2000), 32-byte burst, 16-bit bus

Banks/channel as significant as channel BW
Bandwidth vs. Burst Width

Benchmark = GCC (SPEC 2000), 2 banks/channel
so -- there are some obvious trade-offs related to BURST SIZE, which can affect the TOTAL EXECUTION TIME by 30% or more, keeping all else constant.

Bandwidth vs. Burst Width

System Bandwidth
(GB/s = Channels * Width * 800MHz)

Benchmark = GCC (SPEC 2000), 2 banks/channel
if we look more closely at individual system organizations, there are some clear rules of thumb that appear...
Bandwidth vs. Burst Width

Wide channels (32/64-bit) want large bursts

System Bandwidth
(GB/s = Channels * Width * 800MHz)

Benchmark = GCC (SPEC 2000), 2 banks/channel
Bandwidth vs. Burst Width

Narrow channels (8-bit) want small bursts

System Bandwidth
(\text{GB/s} = \text{Channels} \times \text{Width} \times 800\text{MHz})

Benchmark = GCC (SPEC 2000), 2 banks/channel
so -- if CONCURRENCY were all important, we would expect small bursts to be best, because they would allow a lower average time-to-critical-word for a larger number of simultaneous requests.

what we actually see is that the optimal burst width scales with the bus width, suggesting an optimal number of DATA TRANSFERS per BANK ACTIVATION/PRECHARGE cycle.

i'll illustrate that...

**Bandwidth vs. Burst Width**

Medium channels (16-bit) want medium bursts

System Bandwidth
(GB/s = Channels * Width * 800MHz)

Benchmark = GCC (SPEC 2000), 2 banks/channel
This figure shows the entire range of burst widths that we modeled. Note that some of the figures represent several different combinations, for example, the one third down from top is:

- 2-byte channel + 32-byte burst
- 4-byte channel + 64-byte burst
- 8-byte channel + 128-byte burst

Each burst width scale with bus range of burst-widths modeled.
Burst Width Scales with Bus

Range of Burst-Widths Modeled

- 64-bit channel x 32-byte burst
- 32-bit channel x 32-byte burst
- 64-bit channel x 64-byte burst
- 32-bit channel x 64-byte burst
- 64-bit channel x 128-byte burst
- 16-bit channel x 32-byte burst
- 32-bit channel x 128-byte burst
- 16-bit channel x 64-byte burst
- 8-bit channel x 32-byte burst
- 16-bit channel x 128-byte burst
- 8-bit channel x 64-byte burst
- 8-bit channel x 128-byte burst

Some burst widths are optimal for maximizing bandwidth and minimizing latency.
Focus on 3.2 GB/s — MCF

3.2 GB/s System Bandwidth (channels x width x speed)
Focus on 3.2 GB/s — MCF

Cycles per Instruction (CPI)

3.2 GB/s System Bandwidth (channels x width x speed)

#Banks not particularly important given large burst sizes...

Focus on 3.2 GB/s — MCF

Cycles per Instruction (CPI)

3.2 GB/s System Bandwidth (channels x width x speed)

#Banks not particularly important given large burst sizes...
Focus on 3.2 GB/s — MCF

3.2 GB/s System Bandwidth (channels x width x speed)

... even less so with multi-channel systems
Multi-channel systems sometimes (but not always) a good idea
Focus on 3.2 GB/s — MCF

- Wide Channels: An improvement is seen by increasing the burst size.
- Narrow Channels: Either no improvement or a slight degradation is seen by increasing burst size.

Cycles per Instruction (CPI)

3.2 GB/s System Bandwidth (channels x width x speed)

- 4x 1-byte channels
- 2x 2-byte channels
- 1x 4-byte channels

Legend:
- 1 Bank per Channel
- 2 Banks per Channel
- 4 Banks per Channel
- 8 Banks per Channel
Focus on 3.2 GB/s — BZIP

3.2 GB/s System Bandwidth (channels x width x speed)

Cycles per Instruction (CPI)
Focus on 3.2 GB/s — BZIP

BEST CONFIGS are at SMALLER BURST SIZES

3.2 GB/s System Bandwidth (channels x width x speed)
Queue Size & Reordering

BZIP: 1.6 GB/s (1 channel)

![Graph showing cycles per instruction (CPI) for different queue sizes and bursting modes.]
Conclusions

DESIGN SPACE is NON-LINEAR, COST of MISJUDGING is HIGH

CAREFUL TUNING YIELDS 30–40% GAIN

MORE CONCURRENCY == BETTER (but not at expense of LATENCY)

- Via Channels → NOT w/ LARGE BURSTS
- Via Banks → ALWAYS SAFE
- Via Bursts → DOESN’T PAY OFF
- Via MSHRs → NECESSARY

BURSTS AMORTIZE COST OF PRECHARGE

- Typical Systems: 32 bytes (even DDR2) → THIS IS NOT ENOUGH
Outline

- Basics
- DRAM Evolution: *Structural Path*
- Advanced Basics
- DRAM Evolution: *Interface Path*
- Future Interface Trends & Research Areas
- Performance Modeling: *Architectures, Systems, Embedded*
Embedded DRAM Primer

Embedded

Not Embedded
Whither Embedded DRAM?


- Two predict imminent merger of CPU and DRAM
- Another states we cannot keep cramming more data over the pins at faster rates (implication: embedded DRAM)
- A fourth wants gigantic on-chip L3 cache (perhaps DRAM L3 implementation?)

SO WHAT HAPPENED?
Embedded DRAM for DSPs

MOTIVATION

DSP Compilers => Transparent Cache Model
DSP Buffer Organization Used for Study

Bandwidth vs. Die-Area Trade-Off for DSP Performance
E-DRAM Performance

Embedded Networking Benchmark - Patricia
200MHz C6000 DSP : 50, 100, 200 MHz Memory

Cache Line Size
- 32 bytes
- 64 bytes
- 128 bytes
- 256 bytes
- 512 bytes
- 1024 bytes

Bandwidth
- 0.4 GB/s
- 0.8 GB/s
- 1.6 GB/s
- 3.2 GB/s
- 6.4 GB/s
- 12.8 GB/s
- 25.6 GB/s

CPI

32 bytes
64 bytes
128 bytes
256 bytes
512 bytes
1024 bytes
E-DRAM Performance

Embedded Networking Benchmark - Patricia
200MHz C6000 DSP : 50MHz Memory

Cache Line Size
- 32 bytes
- 64 bytes
- 128 bytes
- 256 bytes
- 512 bytes
- 1024 bytes

Bandwidth
- 0.4 GB/s
- 0.8 GB/s
- 1.6 GB/s
- 3.2 GB/s
- 6.4 GB/s
- 12.8 GB/s
- 25.6 GB/s

Increasing bus width

CPI
- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

Increasing bus width
E-DRAM Performance

Embedded Networking Benchmark - Patricia

200MHz C6000 DSP : 100MHz Memory

Cache Line Size

- 32 bytes
- 64 bytes
- 128 bytes
- 256 bytes
- 512 bytes
- 1024 bytes

Bandwidth

Increasing bus width

CPI

0.4 GB/s 0.8 GB/s 1.6 GB/s 3.2 GB/s 6.4 GB/s 12.8 GB/s 25.6 GB/s
E-DRAM Performance

Embedded Networking Benchmark - Patricia
200MHz C6000 DSP: 200MHz Memory

Increasing bus width

Cache Line Size
- 32 bytes
- 64 bytes
- 128 bytes
- 256 bytes
- 512 bytes
- 1024 bytes

Bandwidth
0.4 GB/s 0.8 GB/s 1.6 GB/s 3.2 GB/s 6.4 GB/s 12.8 GB/s 25.6 GB/s
Performance-Data Sources

“A Performance Study of Contemporary DRAM Architectures,”


“Concurrency, Latency, or System Overhead: Which Has the Largest Impact on DRAM-System Performance?”
*Proc. ISCA ’01*. V. Cuppu and B. Jacob.

“Transparent Data-Memory Organizations for Digital Signal Processors,”
*Proc. CASES ’01*. S. Srinivasan, V. Cuppu, and B. Jacob.

“High Performance DRAMs in Workstation Environments,”
V. Cuppu, B. Jacob, B. Davis, and T. Mudge.

Recent experiments by Sadagopan Srinivasan, Ph.D. student at University of Maryland.
Acknowledgments

The preceding work was supported in part by the following sources:

- NSF CAREER Award CCR-9983618
- NSF grant EIA-9806645
- NSF grant EIA-0000439
- DOD award AFOSR-F496200110374
- … and by Compaq and IBM.
CONTACT INFO

Bruce Jacob

Electrical & Computer Engineering
University of Maryland, College Park
http://www.ece.umd.edu/~blj/
blj@eng.umd.edu

Dave Wang

Electrical & Computer Engineering
University of Maryland, College Park
http://www.wam.umd.edu/~davewang/
davewang@wam.umd.edu