ECE 5730
Memory Systems
Spring 2009

MC Case Studies
Announcements

• **Exam I**
  – Average = 80.1

• **Passwd for new accounts on courses.csl.cornell.edu**

• **Benchmarks for SESC and SimpleScalar**
  – See my email
MC Case Studies

• **Microunity**
  – Schedules cmds to different banks out of order

• **Intel**
  – Schedules cmds within banks and among banks/ranks out of order
Microunity Memory Controller

- “Controller for a Synchronous DRAM that Maximizes Throughput by Allowing Memory Requests and Commands to be Issued Out of Order”
  - Patent filed by Microunity Systems Engineering, Inc. in 1995

- Memory controller for SDRAMs with two internal banks
  - This allows for out of order scheduling in bank parallelism

- Takes advantage of bank parallelism to overlap operations and improve throughout
  - Operations between the two banks may be issued out of order
MC Block Diagram

- Holds 2 requests for bank0
- You can queue up 2 requests

- Puts incoming requests into bank queue
- Re-orders data if necessary
- Line added for clarity

Single rank of SDRAMs (w/ 2 banks)

Selection of commands
Key MC Components

- COMMAND UPDATE UNIT
- BANK QUALIFICATION UNIT
- BANK ARBITRATION UNIT
- C-STATE UPDATE UNIT

Each one is a 210 FIFO with 2 requests (it holds 2) one per bank.

- control block

- B-STATE(n) each bank has some state

- bank data path

- Current state handles DMRM timing constraint

- this is all in the bank data path
Key MC Components

- **Command update unit**
  - Updates the queue for each bank data path

- **Bank qualification unit**
  - Determines when oldest operation in bank is ready

- **Bank arbitration unit**
  - Arbitrates between ready cmd in each bank
  - Schedules between banks out of order all intra-bank operation are in order

- **Constraint update unit**
  - Keeps track of timing constraints given previously issued cmds

The above operations are performed in a single SDRAM clock period
# Command Update Unit

- Updates queued operations for each bank when requests arrive or are popped

## Operation Description

<table>
<thead>
<tr>
<th>Incoming Request</th>
<th>TOS</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>go</td>
<td>tr</td>
<td>np</td>
</tr>
<tr>
<td>rw</td>
<td>tn</td>
<td>pop</td>
</tr>
<tr>
<td>np</td>
<td>st</td>
<td>pnp</td>
</tr>
</tbody>
</table>

### New Cmd

- New page?

### Clear Cmd

- Clear np bit

### Top Entry

- Before dequeue
- After dequeue

| Number of queue entries | Operations
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0 0 X X X X X</td>
<td>1 rw np X X</td>
</tr>
<tr>
<td>0 X X 1 0 1 X X X X X</td>
<td>2 tr tnp rw np</td>
</tr>
<tr>
<td>0 X X 1 0 2 X X X X X</td>
<td>0 X X X X</td>
</tr>
<tr>
<td>1 nr nnp X X</td>
<td>1 nr nnp X X</td>
</tr>
</tbody>
</table>

---

[US5630096] These are coupled. pnp changes np
**Bank Qualification Unit**

- Determines when TOS operation in bank is ready

![Diagram showing TOS bank qualification process](image)

<table>
<thead>
<tr>
<th>New Page</th>
<th>Bank Ready to Activate</th>
<th>Bank Precharged (Closed)</th>
<th>Ready to Activate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\geq 0$</td>
<td>$X$ 1 0 $X$ $X$ 1 $X$ $X$</td>
<td>$X$ 0 0 $X$ 0 0 $X$</td>
<td>$X$ 0 0 $X$ 0 0 $X$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Write</th>
<th>Bank Ready to Activate</th>
<th>Bank Precharged (Closed)</th>
<th>Ready to Activate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\geq 0$</td>
<td>$X$ 1 0 $X$ $X$ 1 $X$ $X$</td>
<td>$X$ 0 0 $X$ 0 0 $X$</td>
<td>$X$ 0 0 $X$ 0 0 $X$</td>
</tr>
<tr>
<td>$\geq 0$</td>
<td>$X$ 0 0 0 $X$ $X$ 1 $X$ $X$</td>
<td>$X$ 0 0 $X$ 0 0 $X$</td>
<td>$X$ 0 0 $X$ 0 0 $X$</td>
</tr>
<tr>
<td>$\geq 0$</td>
<td>$X$ 0 0 0 $X$ $X$ 1 $X$ $X$</td>
<td>$X$ 0 0 $X$ 0 0 $X$</td>
<td>$X$ 0 0 $X$ 0 0 $X$</td>
</tr>
</tbody>
</table>

[US5630096]
Bank Qualification Unit

- Determines when TOS operation in bank is ready

<table>
<thead>
<tr>
<th>TOS</th>
<th>OK to Read Bank</th>
<th>OK to Write Bank</th>
<th>OK to Activate Bank</th>
<th>OK to Prechg Bank</th>
<th>Precharge if</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>st</td>
<td>tr</td>
<td>tmp</td>
<td>Active</td>
<td></td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>&gt;0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Idle</td>
</tr>
<tr>
<td></td>
<td>read</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If no command, precharge if active, otherwise idle:

<table>
<thead>
<tr>
<th></th>
<th>OK to Read Bank</th>
<th>OK to Write Bank</th>
<th>OK to Activate Bank</th>
<th>OK to Prechg Bank</th>
<th>Precharge if</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Bl1</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Bl1</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>Bl1</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>Bl1</td>
</tr>
</tbody>
</table>

[US5630096]
Bank Arbitration Unit

- Chooses between ready cmds in each bank
- Fixed priority
  - Column read
  - Column write
  - Activate
  - Precharge

- Tie goes to the bank other than the one from which the last cmd was issued
Constraint Update Unit

- Tracks timing constraints of the bank operations

<table>
<thead>
<tr>
<th>BL (Burst Length)</th>
<th>tRCD (R/C Delay)</th>
<th>C State</th>
<th>Current Bank</th>
<th>Counter</th>
<th>OK to Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>2</td>
<td>A1</td>
<td>1</td>
<td>trCD-1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>3</td>
<td>A1</td>
<td>1</td>
<td>tRCD-1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>R1</td>
<td>X</td>
<td>BL-1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>W1</td>
<td>1</td>
<td>BL-1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tAA (CAS Latency)</th>
<th>BL</th>
<th>tRCD</th>
<th>C-State</th>
<th>Current Bank</th>
<th>Counter</th>
<th>OK to Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>2</td>
<td>A1</td>
<td>1</td>
<td>tRCD-1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>3</td>
<td>A1</td>
<td>1</td>
<td>tRCD-1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>R1</td>
<td>X</td>
<td>tAA + BL-1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>W1</td>
<td>X</td>
<td>BL-1</td>
<td>0</td>
</tr>
</tbody>
</table>

[US5630096]
## Constraint Update Unit

- Tracks timing constraints of the bank operations

<table>
<thead>
<tr>
<th>tRCm1 (Read Cycle Time)</th>
<th>tRRD (Row-Row Delay)</th>
<th>tRP (RAS Precharge)</th>
<th>C-State</th>
<th>Current Bank</th>
<th>Counter</th>
<th>OK to Activate</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>M1</td>
<td>X</td>
<td>tRCm1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>RF1 → refresh</td>
<td>X</td>
<td>tRCm1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>0 ... 1</td>
<td>P1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>&gt;1</td>
<td>P1</td>
<td>1</td>
<td>tRP-1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>0 ... 1</td>
<td>X</td>
<td>A1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>&gt;1</td>
<td>X</td>
<td>A1</td>
<td>0</td>
<td>tRRD-1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tAA (Write Recovery Time)</th>
<th>BL</th>
<th>tWR (Write Recovery Time)</th>
<th>C-State</th>
<th>Current Bank</th>
<th>Counter</th>
<th>OK to Precharge</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ... 1</td>
<td>&lt;4</td>
<td>X</td>
<td>R1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0 ... 1</td>
<td>4</td>
<td>X</td>
<td>R1</td>
<td>1</td>
<td>tAA + 2</td>
<td>0</td>
</tr>
<tr>
<td>&gt;1</td>
<td>X</td>
<td>X</td>
<td>R1</td>
<td>1</td>
<td>tAA + BL-3</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>W1</td>
<td>1</td>
<td>tWR + BL-2</td>
<td>0</td>
</tr>
</tbody>
</table>

[US5630096]
Read Cmd Example

5, 6, 7, 9: same row in bank0
8: bank1

FIG. 4(c)
Intel Memory Controller

- “Method and Apparatus for Out of Order Memory Scheduling”
  - Patent filed by Intel in 2003

- MC for multiple ranks of SDRAMs

- Schedules reads and writes to different banks/ranks out of order to increase throughput
Multiple Read Queue Algorithm

• **Cmds completing a read sequence have priority**
  – Example: column read to complete activate/read
    
    ➔ already have an open page

• Then, other page hits to same page (up to a limit)

• Then, oldest page hit in all queues

• Etc

 prioritize age, hits over misses
Multiple Read Queue Algorithm

501 - Set last_scheduled_transaction to empty, at power up cold start

502 - Previously scheduled page miss or Page empty unblocked?

503 - Sort Read transactions into queues based on selected attributes (e.g. arrival time)

504 - Determine globally oldest unblocked transaction and locally oldest unblocked transaction for each queue

505 - Last_scheduled_read was a page hit & more unblocked page hits queued to the same page & (consecutive # hits serviced from that page < N or no reads pending to other banks)?

506 - Globally oldest Unblocked read is a page hit?

510 - Last_scheduled_read is a page hit?

524 - if scheduled a page hit & last_scheduled_read is also a page hit
   If both are to the same page, increment consecutive #hits serviced from that page
   Else
   set consecutive #hits to 0
   Else
   set consecutive #hits to 0

526 - Update last_scheduled_read
      Reset last_scheduled_read to hit if read queues are empty for preset duration

528 - Schedule next DRAM command for Oldest PM/PE

529 - Schedule unblocked page hit

530 - Schedule globally oldest unblocked page hit

532 - Schedule globally oldest unblocked read (minimize latency to oldest read)
Multiple Read Queue Algorithm

506
Globally oldest Unblocked read is a page hit?
No
Yes

510
Last_scheduled_read is a page hit?
No
Yes

514
Oldest of Locally oldest unblocked page hit exists?
No
Yes

518
Oldest of Locally oldest unblocked page empty exists?
No
Yes

522
Schedule globally oldest unblocked read page Miss (minimize latency to oldest read)

Schedule globally oldest unblocked page hit

Schedule globally oldest unblocked read (minimize latency to oldest read)

Schedule oldest of locally oldest unblocked page hit

Schedule oldest of locally oldest unblocked page empty

500
FIG. 5

[US7127574]
Single Queue Read Algorithm 1

- Uses a single read queue
- Cmds completing a read sequence have priority
- Then, read to different bank
- Etc
Single Queue Read Algorithm 2

• Globally oldest page hit has priority
  ⇒ favor short operations

• Then, oldest page empty (requires precharge)

• Then, oldest unblocked read
Single Queue Read Algorithm 2

1. **OOO scheduling cycle start**

2. **Globally oldest unblocked page hit?**
   - **Yes**: Schedule oldest unblocked page hit and return.
   - **No**: Proceed to the next step.

3. **Globally oldest unblocked page empty?**
   - **Yes**: Schedule oldest unblocked page empty and return.
   - **No**: Schedule oldest unblocked read transaction and return.

**FIG 7**

[US7127574] Lecture 19: 23
Write Scheduling Algorithm 1

Reset last_scheduled_write

902 - Unblocked write that is on-page to Last_scheduled_write exists?
   Yes → Update last_scheduled_write
   No

906 - Non-conflicting unblocked Write exists?
   Yes → Schedule selected non-conflicting Unblocked write transaction
   No → Schedule oldest unblocked write transaction

900 FIG. 9
Write Scheduling Algorithm 2

1. Set last_scheduled_write to empty
2. If Last_scheduled_write is a page hit?
   - Yes: Update last_scheduled_write
   - No: Schedule oldest unblocked transaction
3. If Non-conflicting unblocked write exists?
   - Yes: Schedule oldest non-conflicting unblocked write transaction
   - No: Schedule oldest unblocked write transaction
Read Versus Write Priority

- Reads have priority over writes, unless the write buffer is getting full
- Unfinished writes (e.g., activate done but not column write) can be preempted by new reads

Final notes:
- There's no optimum for MC policies in all workloads.
- Use machine learning — Sally McKee & Jose Martinez.
Next Time

Overview of Disk Drives