DRAM
Memory Access Protocols
develop generic model for thinking about timing

Reference: “Memory Systems: Cache, DRAM, Disk” & Micron website
Bruce Jacob, Spencer Ng, & David Wang
Today's material & any uncredited diagram came from chapter 11

Generic Structure

Read sequence
Write: reverse 2,3,4
Abstract Command Structure

• Reality
  • huge variety of command sequences possible
    » all with heavily constrained timing issues
      • 2 roles of timing
        1) physical: latency, set-up and hold, signal integrity, lane retiming
        2) power: limit concurrency to stay under thermal/power ceiling

• Start simple
  • command & phase overlap
    CMD duration  phase 2 duration
    duration of multiple bank resource usage
    Note other overlaps - also specified by timing parameters

Row Access Command

• Row activation
  • move data from the mats to sense amps and restore the mats
    » controlled by 2 timing parameters
      • \( t_{RCD} \) - row command delay
        - time to move the data from the mats to the sense amps
        - after a RAS command + \( t_{RCD} \) column reads or writes can commence
      • \( t_{RAS} \) - interval between a RAS command and row restore
        - after a RAS command + \( t_{RAS} \) sense amps can be precharged to activate another row
Column Read Command

• Bank specific
  • move data from sense amps through I/O’s to the Mem_Ctrl
  » 3 timing parameters
    • \( t_{\text{CAS}} \) (or \( t_{\text{CL}} \)) - column address strobe
      - time between col-rd (CAS) command and data valid on the data bus
      - DDRx devices do this in short continuous bursts
    • \( t_{\text{CCD}} \) - minimum column to column command delay due to burst I/O gating
      - 1 cycle for DDR, 2 cycles for DDR2, 4 cycles for DDR3, etc.
    • \( t_{\text{BURST}} \) - duration of the data burst on the bus

Note: some devices have \( t_{\text{CCD}} > t_{\text{Burst}} \) where \( t_{\text{CCD}} \) becomes the limiting factor in what can happen next

Column Write Command

• Move data from mem_ctrl to sense amps
  • timing parameters
    » \( t_{\text{CWD}} \) - delay between col-write and data valid on bus from mem-ctrl
      • some per device differences differences
        - SDRAM: \( t_{\text{CWD}} \) is typically 0
        - DDR - typically 1 memory clock cycle
        - DDR2 - \( t_{\text{CAS}} \) - 1 cycle
        - DDR3 - \( t_{\text{CWD}} \) is programmable
    » Other parameters control a subsequent command’s timing
      • \( t_{\text{WTR}} \) - write to read delay
        - end of write data burst to column read command delay
      • \( t_{\text{WR}} \) - write recovery delay
        - min. interval between end of a write data burst and start of a precharge command
        - I/O gating allowed to overdrive sense amps prior to col-rd-cmdn (mat restore)
      • \( t_{\text{CMD}} \) - time command occupies command bus
Precharge Command

- **Basic sequence**
  - precharge -> RAS -> (CAS R/W)* - precharge ....

- **Timing constraints**
  - $t_{RP}$ - row precharge delay
    - time delay between precharge and row access command
  - $t_{RC}$ - row cycle time
    - $t_{RC} = t_{RAS} + t_{RP}$
    - limits independent row access commands in same bank
Refresh

- Necessary evil of 1T1C DRAM density advantage
  - +: density improves $/bit
    - but the T is not a perfect switch due to leakage
  - -: parasitic
    - power, bandwidth, and resource availability

- Refresh approach varies
  - options exist to reduce 1 of the parasitic effects
    - total refresh power will be constant
      - reduced peak power of the device has some options
  - typical
    - concurrent row precharge in all of the device's banks
      - mem_ctlr issues periodic refresh commands
      - most devices contain row precharge address counter
        - holds addr. of last precharged row
    - tRFC - refresh cycle time
      - duration between refresh commands and an activation (RAS) command

Refresh Overview

- Typical refresh model is block refresh
  - refresh entire device all at once
    - avoids trying to be smart & associated control complexity
    - refresh counter wraps to 0 to indicate done
Refresh Trends

- $t_{RFC}$ is going up
  - decreases availability $\Rightarrow$ slower system memory
  - vendor choice
    - keep inside the 64 ms refresh period
      - even though the number of rows goes up

<table>
<thead>
<tr>
<th>Device</th>
<th>Capacity</th>
<th>Vdd</th>
<th># Banks</th>
<th># Rows</th>
<th>Row Size</th>
<th>Refresh Count</th>
<th>t_{RFC} ns</th>
<th>t_{RFC} ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR</td>
<td>256</td>
<td>2.5V</td>
<td>4</td>
<td>8192</td>
<td>1</td>
<td>8192</td>
<td>60</td>
<td>67</td>
</tr>
<tr>
<td>DDR</td>
<td>512</td>
<td>2.5V</td>
<td>4</td>
<td>8192</td>
<td>2</td>
<td>8192</td>
<td>55</td>
<td>70</td>
</tr>
<tr>
<td>DDR2</td>
<td>1024</td>
<td>1.8V</td>
<td>4</td>
<td>16384</td>
<td>1</td>
<td>8192</td>
<td>55</td>
<td>105</td>
</tr>
<tr>
<td>DDR2</td>
<td>2048</td>
<td>1.8V</td>
<td>8</td>
<td>16384</td>
<td>1</td>
<td>8192</td>
<td>54</td>
<td>127.5</td>
</tr>
<tr>
<td>DDR2</td>
<td>4096</td>
<td>1.8V</td>
<td>8</td>
<td>65536</td>
<td>1</td>
<td>8192</td>
<td>~</td>
<td>~ 327.5</td>
</tr>
</tbody>
</table>

Other Refresh Options

- All have control overhead
  - usually pushed to memory controller
    - since device vendors need to minimize $$/bit
      - device could do it
        - classic cost-performance dilemma
  - Separate bank refresh
    - allow a bank to be refreshed
      - while other bank accesses are still allowed
        - bandwidth win since memory bus can still be active
        - peak power win since 1 RAS on command bus at a time
        - mem_ctlr schedule gets harder
    - next step
      - only refresh what is going to expire
        - huge scheduling problem - probably too hard
Effects of Variable Command Sequences

- Significant performance variation
- Best case
  - read everything in a row and move to next row
    - 1-2 kB in a row - lots of energy expended
      - pass 64-128 B cache-lines to the mem_ctlr
      - access all 8-32 cache lines before opening another row in same bank
        - low probability
        - observed trend as core # increases, # lines/row approaches 1
  - open page memory systems - typical
    - keep row buffer open hoping for the best
      - w/ additional energy cost
- Worst case
  - Precharge -> RAS -> single CAS -> precharge ....
  - closed page memory systems
    - expect the worst but why not make the row smaller?

Read and Write Sequences

Note: % of time data bus bandwidth is utilized
Compound Commands

- **DRAM evolution**
  - allows compound commands
    - mem_ctlr options and scheduling complexity increase
  - column read and precharge
    - use when next scheduled access is to a new row
      - 2 commands rather than 3
      - timing constraints carried over however

Other DDR2 Trends

- **tRAS lockout**
  - Internal timer to make sure tRAS isn't violated
    - if col_rd_pch issues before data restore complete
      - device delays the implicit precharge command
    - allows closed page systems to issue col_rd_pch w/ optimistic timing
      - mem_ctlr doesn't need to worry about precharge of random row access

- **Posted CAS**
  - CAS issued but delayed (posted) by tAL cycles
    - tAL - added latency to column access
      - programmed into the device
      - usually once via initialization commands
  - XDR does same thing via CAS tag
    - logs of mem_ctlr flexibility and complexity hides in this one
    - 1 simplification
      - MC can issue posted CAS immediately after read
      - tAL is set to respect the other timing constraints once
JEDEC Posted CAS

Other Considerations

- Until now
  - view based on resource utilization & single bank timing constraints
- Reality
  - multi-bank DRAM devices & multi-rank DIMMs
    - allows much higher resource utilization via pipelining
    - but package (DRAM die & DIMM) limitations exist
      - peak current limited
        - remember the small pin count
      - thermal constraints
        - how many banks can remain active
  - enter package based timing parameters
    - $t_{RAW}$ - four bank activation window
      - time that 4 banks can be active (DDR2 and DDR3)
    - $t_{RCD}$ - row activate to row activate delay for any DRAM device
      - limits peak current profile
- Combine to impact minimum scheduling times
**Hot DRAMs & Packaging**

*source: random web photos*

- Heat spreaders: DDR 1st step
- Fins and Fans: DDR2 and beyond
- Passive heat pipes

*$$ Ka-ching $$$*

---

**Pipelining Reads**

- Typically \( t_{\text{burst}} > t_{\text{CCD}} \)
- except DDR3 where \( t_{\text{CCD}} = 4 \) cycles
  - so general form is to pick the maximum
Read to Precharge Timing

- Burst may consist of multiple internal bursts
  - interleaved or phased mat returns for bandwidth improvement
  - \( t_{\text{RTP}} \) - read to precharge command interval
    - more general: \( t_{\text{RTP}} + (N-1)t_{\text{CCD}} \) for \( N \) internal bursts
  - sense amps kept open to drive multiple internal bursts through the I/O circuitry

\[ +t_{\text{AL}} \text{ for posted CAS} \]

\[ +t_{\text{BURST}} \text{ is due to burst from previous CAS} \]

\[ f_{\text{CCD}} \]

\[ f_{\text{BURST}} + f_{\text{RTP}} - f_{\text{CCD}} \]

Internal burst length may be different from device burst length

Consecutive Reads

- Different rows same bank
  - best case: \( t_{\text{RAS}} \) elapsed and mats have been restored
  - worst case: have to wait for \( t_{\text{RAS}} \) to complete data restore phase

\[ t_{\text{BURST}} + t_{\text{RTP}} - f_{\text{CCD}} + t_{\text{RP}} + f_{\text{RCD}} \]

\[ t_{\text{CAS}}(t_{\text{CL}}) \]

\[ f_{\text{RAS}} + t_{\text{RP}} = t_{\text{RC}} \]

\[ \text{best case} \]

\[ \text{worst case} \]
Bank Read Conflict

- Consecutive reads to different banks in same rank w/ conflict
  - 2nd read to an inactive row
  - Improvement if commands can be reordered by mem_ctlr

Consecutive Reads to Different Ranks

- Pipeline option more restricted than to same rank different bank command sequences
  - Depends on
    - System level synchronization issues & DRAM operating rate
    - Synchronization is very tricky due to bit-lane jitter and varying trace lengths
    - \( t_{\text{RTRS}} \) - rank to rank switching time due to resynchronization issues
Consecutive Writes

- **Different ranks but to open banks**
  - may be pipelined depending on the bus termination strategy
    - data in this case comes from a common source
      - hence no need to give up control of shared bus
    - termination strategy gets hairier as x gets larger in DDRx
      - on die termination (ODT) for DDR2
        - problem: 2 cycles to turn on ODT and 2.5 cycles to turn it off
    - **t_{ODT}** - time it takes to switch ODT from rank to rank
      - In reality this applies to reads as well but typically \( t_{RAS} > t_{ODT} \)

- **Bank conflict - 2nd write to an inactive row**
  - same rank - bigger delay due to \( t_{BURST} \) and \( t_{WR} \)
  - different rank - more overlap
  - for now best case assume \( t_{RAS} \) has been satisfied
Write After Read: Open Banks

- Pipelining possible if requests are to open banks
  - timing control is primarily restricted by burst length
    - no new timing parameter for this one - phew!
    - different banks allows tighter packing
      - since no new row needs to be precharged & data restore time is overlapped
      - note this case can have a lot of variance in different DRAM technologies

Write After Read: Bank Conflict

- Different banks, bank conflict, no reordering
  - best case for data already restored in old open row
    - e.g. time > tAAS has passed
Read After Write

- Same rank, open banks
  - main issue is the reversal of the data flow
    - RAW vs. WAR
      - write first is worse since data restore time is needed
        - hence RDRAM uses write buffers to improve performance
        - allows I/O gating to be used by another command
        - effectively allows HW support for dynamic command reordering
    - controlled by \( t_{WTR} \) constraint
      - shared I/O gating happens in both cases but with different timing restrictions

Write to Precharge Timing

- Subtle difference
  - write to read timing vs write to precharge
  - due to I/O mux gating time needed to drive the data into the sense amps
    - hence write to precharge must additionally wait for the data to be restored in the mats
Read After Write

- Different ranks, open banks (no bank conflict)
  - data movement change timing issue doesn't apply to the different rank case
    - but rank switching synchronization time does come into play

### Diagram 1

Read After Write

- Bank conflict this time
  - assumes $t_{ras}$ (data restore) time has already elapsed
  - write recovery time must be respected
  - NOTE: If there was a write buffer
    - then a write commit command would be necessary
    - OR retrieve from write buffer which is not currently being done
    - it's that density and cost/bit thing again

### Diagram 2
Read After Write

- **Same rank, bank conflict, no reordering**
  - plus best case - data restore complete

- **Issues**
  - re-ordering will help
  - many relative timing constraints in play
    - I/O gating is critical in this case
    - min scheduling time is:
      \[ \text{max} \left( t_{\text{CMD}} + t_{\text{RP}} + t_{\text{RCD}}, t_{\text{CWD}} + t_{\text{BURST}} + t_{\text{WR}} \right) \]

Col_Rd_&_Precharge Command

- **previously**
  - precharge after column read minimum timing
    - \( t_{\text{BURST}} + t_{\text{RP}} + t_{\text{CCD}} \)
    - \( +t_{\text{AL}} \) if it's a posted read

- unified read and precharge command would be the same
  - but there is an issue of respecting \( t_{\text{RAS}} \) data restore time
  - DDR2 has additional support to delay precharge to insure \( t_{\text{RAS}} \) req's
Col_Wr_Precharge Command

- Tricky - well what isn’t with DRAM?
  - \( t_{\text{RAS}} \) could be defined to include reads and writes
    - this is the case here but not necessarily true in general
    - depends on how complicated you want the mem_ctlr to be
      - BEWARE - how \( t_{\text{RAS}} \) is defined for the components you actually target

---

Additional Constraints

- Power - it’s the biggest problem as things get “better?”
- Rules
  - first rule - things must work
  - second rule - things must get faster
  - third rule - devices must protect themselves
    - Intel learned this the hard way
    - for DRAM this is enforced via timing constraints
- Row activation in the main culprit
  - \( K \)’s of bits moved to the sense amp latches
    - question is how much of them do you use
      - multi-core land indicates a cache line
        - for large num's of cores
- Remember
  - large current profile changes
    - cause timing delays
      - bit line jitter depends on \( V_{\text{dd}} \)
      - Ohm’s law \( V = IR \)
        - not just a good idea - it’s the law
Double Edged Sword

- Active power
  - $P_a = aCV^2f$
- non-adiabatic charge regime
  - $\approx 0.5P$ given off as heat
    - the other half is returned to the power supply
    - $Vdd$ variations on the power lines are an issue
      - also supply tolerance to high variance loads is a design issue
        - requires over provisioning
  - higher temps increase passive $P$ component
- Faster is better
  - except for power since both $f$ and $a$ go up
    - hence so does $P$ and leakage
      - leakage impacts resource availability
      - can't ignore refresh and the 64 ms standard target
Hence Delay 5th Row Activate

Enter Power Driven Timing Parameters

- Limit row activation - \( t_{\text{RRD}} \)
- \# of active banks
  - conflict between performance and power
  - current limit is 4 bank activation window \( t_{\text{FAW}} \)
  - both get worse as device width goes up

<table>
<thead>
<tr>
<th>Micron</th>
<th>Device Configuration</th>
<th>512 Mb x 4</th>
<th>256 Mb x 8</th>
<th>128 Mb x 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus width</td>
<td></td>
<td>4</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>Bank count</td>
<td></td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Row Count</td>
<td></td>
<td>16384</td>
<td>16384</td>
<td>8196</td>
</tr>
<tr>
<td>Column Count</td>
<td></td>
<td>2048</td>
<td>1024</td>
<td>1024</td>
</tr>
<tr>
<td>Row Size</td>
<td></td>
<td>8192</td>
<td>8192</td>
<td>16384</td>
</tr>
<tr>
<td>( t_{\text{RRD}} ) ns</td>
<td></td>
<td>7.5</td>
<td>7.5</td>
<td>10</td>
</tr>
<tr>
<td>( t_{\text{FAW}} ) ns</td>
<td></td>
<td>37.5</td>
<td>37.5</td>
<td>50</td>
</tr>
</tbody>
</table>
## Partitioned Address and Command Bus

- Alleviates variable trace length to some extent
  - binary tree partition
    - doesn’t work for DDR2 and DDR3
    - but similar to BoB

![Diagram of Partitioned Address and Command Bus](image)

## Summary Timing Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tAL</td>
<td>added latency to column accesses for posted CAS</td>
</tr>
<tr>
<td>tBURST</td>
<td>data burst duration on the data bus</td>
</tr>
<tr>
<td>tCAS</td>
<td>interval between CAS and start of data return</td>
</tr>
<tr>
<td>tCCD</td>
<td>column command delay - determined by internal burst timing</td>
</tr>
<tr>
<td>tCMD</td>
<td>time command is on bus from MC to device</td>
</tr>
<tr>
<td>tCWDB</td>
<td>column write delay, CAS write to write data on the bus from the MC</td>
</tr>
<tr>
<td>tFAW</td>
<td>rolling temporal window for how long four banks can remain active</td>
</tr>
<tr>
<td>tOST</td>
<td>interval to switch ODT control from rank to rank</td>
</tr>
<tr>
<td>tRAS</td>
<td>row access command to data restore interval</td>
</tr>
<tr>
<td>tRC</td>
<td>interval between accesses to different rows in same bank = tRAS+tRP</td>
</tr>
<tr>
<td>tRCD</td>
<td>interval between row access and data ready at sense amps</td>
</tr>
<tr>
<td>tRFC</td>
<td>interval between refresh and activation commands</td>
</tr>
<tr>
<td>tRP</td>
<td>interval for DRAM array to be precharged for another row access</td>
</tr>
<tr>
<td>tRRD</td>
<td>interval between two row activation commands to same DRAM device</td>
</tr>
<tr>
<td>tRTP</td>
<td>interval between a read and a precharge command</td>
</tr>
<tr>
<td>tTRTS</td>
<td>rank to rank switching time</td>
</tr>
<tr>
<td>tWR</td>
<td>write recovery time - interval between end of write data burst and a precharge command</td>
</tr>
<tr>
<td>tWTR</td>
<td>interval between end of write data burst and start of a column read command</td>
</tr>
</tbody>
</table>
### Summary Minimal Timing Equations

<table>
<thead>
<tr>
<th>Prev</th>
<th>Next</th>
<th>Rank</th>
<th>Bank</th>
<th>Min. Timing</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>s</td>
<td>s</td>
<td>tRC</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>s</td>
<td>d</td>
<td>tRD</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>A</td>
<td>s</td>
<td>d</td>
<td>tRP</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>A</td>
<td>s</td>
<td>s</td>
<td>tRFC</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>R</td>
<td>s</td>
<td>s</td>
<td>tRCD-tAL</td>
<td>TAL=0 unless posted CAS</td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td>s</td>
<td>a</td>
<td>tICCD</td>
<td>tBURST+</td>
</tr>
<tr>
<td>P</td>
<td>R</td>
<td>d</td>
<td>a</td>
<td>tRRS</td>
<td>tBURST+</td>
</tr>
<tr>
<td>F</td>
<td>W</td>
<td>s</td>
<td>a</td>
<td>tWFR</td>
<td>tBURST prev CASW same rank</td>
</tr>
<tr>
<td>W</td>
<td>W</td>
<td>a</td>
<td>a</td>
<td>tCAS</td>
<td>tBURST prev CASW diff rank</td>
</tr>
<tr>
<td>A</td>
<td>W</td>
<td>s</td>
<td>s</td>
<td>tRC(t)+AL</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>W</td>
<td>d</td>
<td>a</td>
<td>tCAS+tBURST</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>W</td>
<td>s</td>
<td>a</td>
<td>tICCD</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>P</td>
<td>s</td>
<td>a</td>
<td>tRAP</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>P</td>
<td>s</td>
<td>s</td>
<td>tICCD</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>P</td>
<td>s</td>
<td>s</td>
<td>tBURST+1W</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>F</td>
<td>s</td>
<td>a</td>
<td>tRFC</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>F</td>
<td>s</td>
<td>a</td>
<td>tRFC</td>
<td></td>
</tr>
</tbody>
</table>

- **A** = row access
- **R** = col_rd
- **W** = col_wr
- **P** = precharge
- **F** = Refresh
- **s** = same
- **d** = different
- **a** = any

---

### Projects

- **Note this is an abstracted view**
  - Individual devices may vary and do particularly RAMBUS
  - **project ideas**
    - **#1**
      - specify the things that count for a particular technology
      - and then write a mem_ctlr that respects these constraints
      - It’s all about scheduling
    - **#2**
      - simulate RAMBUS vs. JEDEC for various workloads
      - pick your technology step
    - **#3**
      - evolution is not always good
      - compare memory performance SDRAM, DDR, DDR2, DDR3 ...
      - use basic timing model but with device specific values