Flipping Bits in Memory Without Accessing Them

An Experimental Study of DRAM Disturbance Errors

Onur Mutlu et al.

Jan Mazur, Oct. 2019
DRAM - Dynamic Random Access Memory

- Dynamic - capacitors leak charge over time, they need to be periodically refreshed (i.e. read and rewritten)
- Random-access - memory can be read and changed in any order
Basic DRAM overview

Row and column identify single bit.

Sense Amps also called row buffers.
Multiple memory arrays - x4 DRAM, x8 DRAM

DRAM outputs as many bits as it has arrays.

Shared column decoder and row decoder.
FIGURE 8.1: A 64-Mbit Fast Page Mode DRAM device (4096 x 1024 x 16).
Bank - independent group of memory arrays

- DRAM chips on previous slide had multiple arrays but single bank
- Banks can be activated, precharged, read etc. at the same time as other banks. Share command, address, data busses.

Why?

Interleave memory accesses to achieve high-bandwidth busses using low-bandwidth devices.
Banks are (sort of) 3D

3D = 2D arrays with #memory_arrays bits in each cell.

Cell ~ column

DRAM page also called DRAM row.
DRAM subsystem organization

- Multiple channels - multiple memory controllers. Each covers multiple ranks
- Multiple DIMMs containing multiple ranks
- Multiple ranks spanning multiple chips
- Chips within a rank accept the same commands.

- Channel
- DIMM
- Rank
- Chip
- Bank
- Row/Column
Memory system organization

One DRAM with eight internal BANKS, each of which connects to the shared I/O bus.

One DRAM bank is comprised of potentially many DRAM ARRAYS, depending on the part’s configuration. This example shows 4 arrays, indicating a x4 part.

One DIMM can have one RANK or two RANKS of DRAM on it, depending on its configuration.
Column, row, and bank in different contexts (and publications) mean different things.
DRAM devices in a rank respond to the same command & addr but send different data.

Together they form wider data bus. 64bit data bus = 8 * x8 DRAM = 4 * x16 DRAM
“Mesh Topology”
decode-dimms

--- Memory Characteristics ---

Maximum module speed: 1600 MHz (PC3-12800)

Size:
- 4096 MB

Banks x Rows x Columns x Bits:
- 8 x 16 x 10 x 64

Ranks:
- 1

SDRAM Device Width:
- 8 bits

Number of SDRAM DIMMs detected and decoded: 1

Bits - probably rank width. So 64 / SDRAM Device Width = 8 DRAM chips within rank.

8 DRAM chips * DRAM width * 2^{16} rows * 2^{10} columns * 8 banks = 4 GB of memory.

Install i2c-tools; modprobe eeprom; decode-dimms
Access sequence (simplified; to single DRAM)

1. Precharge bitlines in appropriate bank.
2. Open/Aactivate row by raising appropriate wordline. This connects rows to bitlines, transferring data into the bank’s row buffer (it’s also amplified).
3. Read/write columns. Row buffer’s data is accessed.
4. Close row. In order to access another row, current row’s wordline must be lowered. Row-buffer is cleared. Charge is restored.

$t_{RC}$ - row cycle time - timing constraint defined between a pair of ACTIVATEs to the same row (in the same bank). Usually ~ 55ns.
DRAM commands & refreshing

<table>
<thead>
<tr>
<th>Operation</th>
<th>Command</th>
<th>Address(es)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Open Row</td>
<td>ACTIVATE (ACT)</td>
<td>Bank, Row</td>
</tr>
<tr>
<td>2. Read/Write Column</td>
<td>READ/WRITE</td>
<td>Bank, Column</td>
</tr>
<tr>
<td>3. Close Row</td>
<td>PRECHARGE (PRE)</td>
<td>Bank</td>
</tr>
<tr>
<td>Refresh (Section 2.4)</td>
<td>REFRESH (REF)</td>
<td>—</td>
</tr>
</tbody>
</table>

REFRESH == ACTIVATE

When row is activated, charges are amplified by sense-amps.

The DDR3 DRAM specifications guarantee a retention time of at least 64 milliseconds, meaning that all cells within a rank need to be refreshed at least once during this time window.

REF command refreshes many rows at a time. When a rank receives a REF, it automatically refreshes several of its least-recently-refreshed rows by internally generating ACT and PRE pairs to them.
Disturbance errors

When two circuit components interact with each other in unintended way.

Some errors can be detected by manufactures (DRAM is packed, it’s hard to make it precise => remapping).

Hypothesis:

*When a wordline’s voltage is toggled repeatedly, some cells in nearby rows leak charge at a much faster rate. Such cells cannot retain charge for even 64ms, the time interval at which they are refreshed.*

“Among 129 DRAM modules we analyzed (comprising 972 DRAM chips), we discovered disturbance errors in 110 modules (836 chips).”
Test code

<table>
<thead>
<tr>
<th>code1a:</th>
<th>code1b:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 mov (X), %eax</td>
<td>1 mov (X), %eax</td>
</tr>
<tr>
<td>2 mov (Y), %ebx</td>
<td>2 clflush (X)</td>
</tr>
<tr>
<td>3 clflush (X)</td>
<td>4</td>
</tr>
<tr>
<td>4 clflush (Y)</td>
<td>5</td>
</tr>
<tr>
<td>5 mfence</td>
<td>6 mfence</td>
</tr>
<tr>
<td>6 jmp code1a</td>
<td>7 jmp code1b</td>
</tr>
</tbody>
</table>

a. Induces errors  b. Does not induce errors

clflush - evicts data from the cache.
mfence - ensures data is fully flushed*.

Physical addr to bank mapping disclosed on AMD, reverse-engineered on Intel.

X and Y map to the same bank, but to different rows within the bank. This forces the memory controller to open and close the two rows repeatedly: (ACT(X), READ(X), PRE(X), ACT(Y), READ(Y), PRE(Y), ... ).

Execute with different (X,Y) until every row is opened/closed millions of times.
Bit-flips induced by disturbance on a 2GB module

<table>
<thead>
<tr>
<th>Bit-Flip</th>
<th>Sandy Bridge</th>
<th>Ivy Bridge</th>
<th>Haswell</th>
<th>Piledriver</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘0’ → ‘1’</td>
<td>7,992</td>
<td>10,273</td>
<td>11,404</td>
<td>47</td>
</tr>
<tr>
<td>‘1’ → ‘0’</td>
<td>8,125</td>
<td>10,449</td>
<td>11,467</td>
<td>12</td>
</tr>
</tbody>
</table>

The faster a processor accesses DRAM, the more bit-flips it has.

Code1b:

Access pattern: (reqX, reqX, reqX, ...). In this case, the memory controller minimizes the number of DRAM commands by opening and closing the row just once, while issuing many column reads in between: (ACTX, READX, READX, READX, ..., PREX).

No disturbance errors detected.
FPGA-based test environment

**AI** - activation interval; time it takes to execute one iteration of the inner for loop.

**RI** - refresh interval; how frequently module is refreshed

**DP** - data pattern

---

**TestBulk**(AI, RI, DP)

1. setAI(AI)
2. setRI(RI)
3. \(N \leftarrow (2 \times RI)/AI\)
4. writeAll(DP)
5. for \(r \leftarrow 0 \ldots ROW_{MAX}\)
   6. for \(i \leftarrow 0 \ldots N\)
      7. ACT \(r^{th}\) row
      8. READ \(0^{th}\) col.
      9. PRE \(r^{th}\) row
   10. readAll()
11. findErrors()

**TestEach**(AI, RI, DP)

1. setAI(AI)
2. setRI(RI)
3. \(N \leftarrow (2 \times RI)/AI\)
4. for \(r \leftarrow 0 \ldots ROW_{MAX}\)
   5. writeAll(DP)
   6. for \(i \leftarrow 0 \ldots N\)
      7. ACT \(r^{th}\) row
      8. READ \(0^{th}\) col.
      9. PRE \(r^{th}\) row
   10. readAll()
11. findErrors()

**a.** Test all rows at once  
**b.** Test one row at a time

AI - 55ns - t\(_{RC}\) min  
RI - 64ms - default
Nomenclature

- Victim row
- Aggressor row

If a cell experienced an error in either of the runs, we refer to it as a victim cell for that module.

In most of the tests, AI=55ns and RI=64ms, for which the corresponding value of $N$ is $2.33 \times 10^6$. 
A graph showing the number of errors per $10^9$ cells for A, B, and C modules over time. The x-axis represents the module manufacture date from 2008 to 2014, and the y-axis represents the errors per $10^9$ cells on a logarithmic scale from $10^0$ to $10^6$. The data points for each module type are marked with different symbols: A Modules (black circles), B Modules (red squares), and C Modules (green diamonds). Error bars are also visible for some data points.
Errors

Refresh Interval (ms)

$y_A = 4.39e-6 \times x^{6.23}$

$y_B = 1.23e-8 \times x^{7.3}$

$y_C = 8.11e-10 \times x^{7.3}$
\[ y_A = 5.63 \times 10^6 \times 1.04^{-x} \]
\[ y_B = 1.06 \times 10^6 \times 1.04^{-x} \]
\[ y_C = 1.90 \times 10^5 \times 1.05^{-x} \]
ECDED (single error-correction, double error-detection) can correct only a single-bit error within a 64-bit word. If a word contains two victims, however, SECDED cannot correct the resulting double-bit error. And for three or more victims, SECDED cannot even detect the multi-bit error, leading to silent data corruption.

<table>
<thead>
<tr>
<th>Module</th>
<th>Number of 64-bit words with X errors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X = 1</td>
</tr>
<tr>
<td>A_{23}</td>
<td>9,709,721</td>
</tr>
<tr>
<td>B_{11}</td>
<td>2,632,280</td>
</tr>
<tr>
<td>C_{19}</td>
<td>141,821</td>
</tr>
</tbody>
</table>
How many times aggressor row flips X cells
How many times aggressor row affects X rows
- Aggressor is refreshing its data => 0 errors.
- Errors in non-adjacent rows => re-mapping
- Logical / physical adjacency
1 -> 0 & 0 -> 1 flips

- Some cells might represent logical value 1 as charged and some as discharged.
- RowStripe = (even/odd rows ‘0’s/‘1’s); Solid (all ‘0’s)

<table>
<thead>
<tr>
<th>Module</th>
<th>TestBulk(DP) + TestBulk(~DP)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Solid</td>
</tr>
<tr>
<td>A_{23}</td>
<td>112,123</td>
</tr>
<tr>
<td>B_{11}</td>
<td>12,050</td>
</tr>
<tr>
<td>C_{19}</td>
<td>57</td>
</tr>
</tbody>
</table>

**Table 6. Number of errors for different data patterns**
It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after.
Flipping bits in memory for fun (and profit)

- Generic strategy
  - Identify data structure that, if randomly bit-flipped, yields improved privileges
  - Fill as much memory as possible with this data structure
  - Wait for the bit flip to occur

- Apply this to JVM:
  - Spray memory with references
  - Bit flip causes reference to point to object of wrong type
Flipping bits in memory for fun (and profit)

https://github.com/google/rowhammer-test

Escape JVM, NaCl. Get access to kernel-memory:

![Diagram of Physical-Page Base Address](image)

**Figure 5-21. 4-Kbyte PTE—Long Mode**

Could flip:
- “Writable” permission bit (RW): 1 bit $\rightarrow$ 2% chance
- Physical page number: 20 bits on 4GB system $\rightarrow$ 31% chance
Error correction mechanisms

- Parity - detection
- SEC ECC single-bit error correction
- SECDED ECC Single-bit error correction, double-bit error detection
- Multi-Bit Error Detection and Correction: Bossen’s b-Adjacent Algorithm

More about that in “Memory Systems: Cache, DRAM, Disk” - chapter 30.

Consider State(full/less), Space-cost, Time-cost, energy-cost.
Solutions

1. **Make better chips** - improve circuit design (more precision). Problem could resurface when the process technology is upgraded.
2. **Correct errors** - ECC. Space inefficient, high cost, cannot correct multi-bit disturbance errors.
3. **Refresh all rows frequently** - energy and time inefficient.
4. **Retire cells (manufacturer)** - days to find faulty rows. Might not be enough healthy rows.
5. **Dynamically identify “hot” rows and refresh neighbours** - needs too much hw to track hot rows; heuristics and filters sometimes cause multiple rows to be refreshed.
PARA - probabilistic adjacent row activation

Every time a row is opened and closed, one of its adjacent rows is also opened (i.e., refreshed) with some low probability p. Choose adjacent row with equal ppb.

Implemented in memory-controller. Stateless.

Probability of error on N accesses = \((1-p/2)^N\) (binomial distribution)
Mapping disclosure

To enable low-overhead solutions, manufacturers should disclose how they map logical rows onto physical rows. Along with other metadata about the module (e.g., capacity, and bus frequency), the mapping function could be stored in a small ROM (called the SPD) that exists on every DRAM module.

The manufacturers should also disclose how they re-map faulty physical rows. It also could be stored in SPD.
PARA results

Adversarial access pattern opens and closes a row just enough times ($N_{th}$) during a refresh interval but no more ($p=0.001$).

<table>
<thead>
<tr>
<th>Duration</th>
<th>$N_{th}=50K$</th>
<th>$N_{th}=100K$</th>
<th>$N_{th}=200K$</th>
</tr>
</thead>
<tbody>
<tr>
<td>64ms</td>
<td>$1.4 \times 10^{-11}$</td>
<td>$1.9 \times 10^{-22}$</td>
<td>$3.6 \times 10^{-44}$</td>
</tr>
<tr>
<td>1 year</td>
<td>$6.8 \times 10^{-3}$</td>
<td>$9.4 \times 10^{-14}$</td>
<td>$1.8 \times 10^{-35}$</td>
</tr>
</tbody>
</table>

Table 7. Error probabilities for PARA when $p=0.001$
PARA performance impact

Impact on 29 single-threaded workloads from SPEC CPU2006, TPC, and memory-intensive microbenchmarks evaluated on a cycle-accurate DRAM simulator.

Averaged across all 29 benchmarks, there was only a 0.197% degradation in instruction throughput during the simulated duration of 100ms. In addition, the largest degradation in instruction throughput for any single benchmark was 0.745%.

Sooooo it looks lightweight.
Bibliography

- Onur Mutlu et al. - *Flipping bits in memory without accessing them: an experimental study of DRAM disturbance errors*
- Bruce Jacob, David T. Wang, Spencer Ng - *Memory Systems: Cache, DRAM, Disk*
- Mark Seaborn, Thomas Dullien - *Exploiting the DRAM rowhammer bug to gain kernel privileges. How to cause and exploit single bit errors*