Dr. Robert Dennard was granted the patent for one-transistor DRAM cell.

The first commercially available DRAM IC:
- Intel 1103 (1Kbit) - [1970]

The first SDRAM:
- Samsung KM48SL2000 (16 Mbit) - [1992]

The first DDR memory prototype:
- Samsung DDR SDRAM (64 Mbit) - [1997]
DRAM in 2022

- GPUs
- Safety-critical systems
- Servers
- Personal Computers
- Mobile Devices

(32Gbit)

DRAM Scaling Challenges

SAFARI
Scaling Challenges of DRAM Technology

**DRAM Cell**

**Data Integrity**

**Latency**

**DRAM Scaling Problems**

**System-Level Problems**

- Performance
- Energy Efficiency
- Reliability
- Security

SAFARI
Our Goals

To combat the **system-level implications** of the **DRAM scaling challenges**:

1. Build an **infrastructure** for characterization, analysis, and understanding of real DRAM chips

2. Enable new **mechanisms** for improving DRAM performance, energy consumption, reliability, and security
Contributions

A flexible and easy-to-use FPGA-based DRAM characterization infrastructure

SoftMC
HPCA’17

U-TRR
MICRO’21

A new methodology for uncovering in-DRAM RowHammer Protection Mechanisms

A low-cost substrate for improving DRAM performance, energy efficiency, and reliability

Self-Managing DRAM: Enabling autonomous and efficient in-DRAM maintenance operations

SMD
Ongoing arXiv’22

CROW
ISCA’19
DRAM Background

DRAM Technology, Organization, and Operation
DRAM Organization

CPU

Memory Controller

Memory Bus

DRAM Bank

Row

Sense Amplifier

DRAM Cell

DRAM

Memory

Controller

CPU
DRAM Operation

- Precharge
- Activate
- Read/Write

DRAM Bank

Sense Amplifier
A single bit is encoded in a small capacitor and leaky.

Stored data is corrupted if too much charge leaks.
Periodic refresh operations preserve stored data
Contributions

A flexible and easy-to-use FPGA-based DRAM characterization infrastructure

SoftMC
HPCA’17

Self-Managing DRAM: Enabling autonomous and efficient in-DRAM maintenance operations

SMD
Ongoing arXiv’22

A new methodology for uncovering in-DRAM RowHammer Protection Mechanisms

U-TRR
MICRO’21

A low-cost substrate for improving DRAM performance, energy efficiency, and reliability

CROW
ISCA’19

SAFARI
Reliability Effects of DRAM Timing Parameters

Many of the factors affecting DRAM **reliability** and **latency** cannot be properly modeled.
Factors Affecting DRAM Reliability and Latency

We need to perform experimental studies of real DRAM chips
Goals of a DRAM Characterization Infrastructure

• **Flexibility**
  • Ability to test *any* DRAM operation
  • Ability to test *any combination* of DRAM operations and *custom* timing parameters

• **Ease of use**
  • *Simple* programming interface (C++)
  • *Minimal* programming effort and time
  • *Accessible* to a wide range of users
    • *who may lack experience in hardware design*
SoftMC: High-Level View

The first publicly-available FPGA-based DRAM characterization infrastructure

Easily programmable using the SoftMC C++ API
Key Components

- SoftMC API
- PCIe Driver
- SoftMC Hardware
Writing data to DRAM:

```c
InstructionSequence iseq;
iseq.insert(genACT(bank, row));
iseq.insert(genWAIT(tRCD));
iseq.insert(genWR(bank, col, data));
iseq.insert(genWAIT(tCL + tBL + tWR));
iseq.insert(genPRE(bank));
iseq.insert(genWAIT(tRP));
iseq.insert(genEND());
iseq.execute(fpga);
```
Key Components

- SoftMC API
- PCIe Driver
- SoftMC Hardware

SAFARI

SoftMC (HPCA'17) U-TRR (MICRO'21) SMD (Ongoing) CROW (ISCA'19)
SoftMC Hardware

- PCIe Controller
- Instruction Receiver
- Instruction Queue
- Instruction Dispatcher
- DDR PHY
- Auto-refresh Controller
- Calibration Controller
- Read Capture

Wait (Ready to access Latency)

Host Machine

Instructions

Data

SoftMC Hardware (FPGA)

20
Use Cases

1. Retention Time Distribution Study

2. Evaluating the Effectiveness of New DRAM Latency Reduction Techniques
Use Case 1: Retention Time Distribution Study

Can be implemented with just ~100 lines of C++ code

```cpp
InstructionSequence iseq;
iseq.insert(genACT(bank, row));
iseq.insert(genWAIT(tRCD));
for(int col = 0; col < COLUMNS; col++){
    iseq.insert(genWR(bank, col, data));
    iseq.insert(genWAIT(tBL));
}
iseq.insert(genWAIT(tCL + tWR));
iseq.insert(genPRE(bank));
iseq.insert(genWAIT(tRP));
iseq.insert(genEND());
iseq.execute(fpga));
```
Use Case 1: Results

Validates the correctness of the SoftMC Infrastructure

Number of Erroneous Bytes

@ ~20°C (room temperature)

Module A
Module B
Module C

SoftMC (HPCA'17) U-TRR (MICRO'21) SMD (Ongoing) CROW (ISCA'19)
Use Case 2: Accessing Highly-Charged Cells Faster

**NUAT**  
*(Shin+, HPCA 2014)*

**ChargeCache**  
*(Hassan+, HPCA 2016)*

A highly-charged cell can be accessed with low latency
Use Case 2: How a Highly-Charged Cell Is Accessed Faster?

- **Activation Latency**
- **Precharge Latency**

**Ready-to-access Latency**

- **Activate**
- **Read**
- **Precharge**
- **Activate**

- **DRAM Cell**
- **Sense Amplifier**

Time: 0 (refresh) - 64 ms
Use Case 2: Ready-to-Access Latency Test

- Longer wait ➔ Lower cell charge
- Shorter wait ➔ Higher cell charge

With custom ready-to-access latency parameter

Can be implemented with just ~150 lines of C++ code
Use Case 2: Results

We do not observe the expected latency reduction effect in existing DRAM chips.
Use Case 2: Why Don’t We See the Latency Reduction Effect?

The memory controller cannot externally control when a sense amplifier gets enabled in existing DRAM chips.

- **Ready to Access**
- **Charge Level**
- **Potential Reduction**
- **Fixed Latency!**
- **Enabling the Sense Amplifier**

**Data 0**

**Data 1**

**Cell**

**Sense Amp**

**time**

**charge**

2) [DSN’22] Yaglikci+, “Understanding RowHammer Under Reduced Wordline Voltage: An Experimental Study Using Real DRAM Devices”

3) [MICRO’21] Orosa+, “A Deeper Look into RowHammer’s Sensitivities: Experimental Analysis of Real DRAM Chips and Implications on Future Attacks and Defenses”

4) [MICRO’21] Hassan+, “Uncovering In-DRAM RowHammer Protection Mechanisms: A New Methodology, Custom RowHammer Patterns, and Implications”


6) [ISCA’21] Orosa+, “CODIC: A Low-Cost Substrate for Enabling Custom In-DRAM Functionalities and Optimizations”

7) [ISCA’20] Kim+, “Revisiting RowHammer: An Experimental Analysis of Modern Devices and Mitigation Techniques”

8) [S&P’20] Frigo+, “TRRespass: Exploiting the Many Sides of Target Row Refresh”

9) [HPCA’19] Kim+, “D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput”


13) [MICRO’17] Khan+, “Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content”

14) [SIGMETRICS’16] Chang+, “Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization”
## Research Enabled by SoftMC (others)

2) [ETS’21] Farmani+, “RHAT: Efficient RowHammer-Aware Test for Modern DRAM Modules”
3) [HOST’20] Talukder+, “Towards the Avoidance of Counterfeit Memory: Identifying the DRAM Origin”
4) [MICRO’19] Gao+, “ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs”
6) [ICCE’18] Talukder+, “Exploiting DRAM Latency Variations for Generating True Random Numbers”
Summary

**SoftMC**

The first **publicly-available** DRAM characterization infrastructure

- **Flexible** and Easy to Use
- **Source code** available on GitHub: [github.com/CMU-SAFARI/SoftMC](https://github.com/CMU-SAFARI/SoftMC)

SoftMC enables many **studies, ideas, and methodologies** in the design of future memory systems
Contributions

A flexible and easy-to-use FPGA-based DRAM characterization infrastructure

A new methodology for uncovering in-DRAM RowHammer Protection Mechanisms

Self-Managing DRAM: Enabling autonomous and efficient in-DRAM maintenance operations

A low-cost substrate for improving DRAM performance, energy efficiency, and reliability
Repeatedly **opening** (activating) and **closing** (precharging) a DRAM row causes **RowHammer bit flips** in nearby cells.
Target Row Refresh (TRR)

DRAM vendors equip their DRAM chips with a *proprietary* mitigation mechanisms known as **Target Row Refresh (TRR)**

**Key Idea:** TRR refreshes nearby rows upon detecting an aggressor row.

![Diagram of TRR-equipped DRAM Chip]

- **Memory Controller**
  - REF
  - Aggressor detected: **Row 2**
  - Refresh neighbor rows

![TRR-induced refreshes]

- **TRR-equipped DRAM Chip**
  - Row 0
  - Row 1
  - **Aggressor Row: Row 2**
  - Row 3
  - Row 4
The Problem with TRR

TRR is **obscure**, **undocumented**, and **proprietary**

We **cannot** easily study the **security properties** of TRR
Study **in-DRAM TRR** mechanisms to

1. **understand** how they operate
2. **assess** their security
3. **secure** DRAM completely against RowHammer
A new methodology to uncover the inner workings of TRR

Key Idea:

Use data retention failures as a side channel to detect when a row is refreshed by TRR
U-TRR: High-Level Overview

U-TRR has two main components

**Row Scout (RS)**

Finds a set of DRAM rows that meet certain requirements as needed by TRR-A and identifies the data retention times of these rows.

**TRR Analyzer (TRR-A)**

Uses RS-provided rows to distinguish between TRR-induced and regular refreshes, and thus builds an understanding of the underlying TRR mechanism.

---

**Profiling Configuration**
- row group layout
- row group count
- bank
- range
- ...

**Row Scout (RS)**
- Retention Profiled Rows (RPR)

**Experiment Configuration**
- aggressor (A) row addr.
- dummy (D) row addr.
- hammering mode
- number of rounds
- A/D hammer counts
- REF count
- ...

**TRR Analyzer (TRR-A)**

**Analysis**

RPRs refreshed by TRR-induced refresh
We perform many experiments to understand the operation of different TRR mechanisms.
We implement U-TRR using SoftMC,

<table>
<thead>
<tr>
<th>Module</th>
<th>Date (yy-ww)</th>
<th>Chip Density (Gbit)</th>
<th>Organized</th>
<th>HCfirst</th>
<th>Version</th>
<th>Aggressor Detection</th>
<th>Aggressor Capacity</th>
<th>Per-Bank TRR</th>
<th>TRR-to-REF Ratio</th>
<th>Neighbors Refreshed</th>
<th>% Vulnerable DRAM Rows†</th>
<th>Max. Bit Flips per Row per Hammer†</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>19-50</td>
<td>8</td>
<td>1 16 8</td>
<td>16K</td>
<td>ATRRI</td>
<td>Counter-based</td>
<td>16</td>
<td>✓</td>
<td>1/9</td>
<td>4</td>
<td>73.3%</td>
<td>1.16</td>
</tr>
<tr>
<td>A1-5</td>
<td>19-36</td>
<td>8</td>
<td>1 8 16</td>
<td>13K-15K</td>
<td>ATRRI</td>
<td>Counter-based</td>
<td>16</td>
<td>✓</td>
<td>1/9</td>
<td>4</td>
<td>99.2% - 99.4%</td>
<td>2.32 - 4.73</td>
</tr>
<tr>
<td>A6-7</td>
<td>19-45</td>
<td>8</td>
<td>1 8 16</td>
<td>13K-15K</td>
<td>ATRRI</td>
<td>Counter-based</td>
<td>16</td>
<td>✓</td>
<td>1/9</td>
<td>4</td>
<td>99.3% - 99.4%</td>
<td>2.12 - 3.86</td>
</tr>
<tr>
<td>A8-9</td>
<td>20-07</td>
<td>8</td>
<td>1 16 8</td>
<td>12K-14K</td>
<td>ATRRI</td>
<td>Counter-based</td>
<td>16</td>
<td>✓</td>
<td>1/9</td>
<td>4</td>
<td>74.6% - 75.0%</td>
<td>1.96 - 2.96</td>
</tr>
<tr>
<td>A10-12</td>
<td>19-51</td>
<td>8</td>
<td>1 8 16</td>
<td>12K-13K</td>
<td>ATRRI</td>
<td>Counter-based</td>
<td>16</td>
<td>✓</td>
<td>1/9</td>
<td>4</td>
<td>74.6% - 75.0%</td>
<td>1.48 - 2.86</td>
</tr>
<tr>
<td>A13-14</td>
<td>20-31</td>
<td>8</td>
<td>1 8 16</td>
<td>11K-14K</td>
<td>ATRRI</td>
<td>Counter-based</td>
<td>16</td>
<td>✓</td>
<td>1/9</td>
<td>2</td>
<td>94.3% - 98.6%</td>
<td>1.53 - 2.78</td>
</tr>
<tr>
<td>B0</td>
<td>18-22</td>
<td>4</td>
<td>1 16 8</td>
<td>44K</td>
<td>BTRRI</td>
<td>Sampling-based</td>
<td>1</td>
<td>✗</td>
<td>1/4</td>
<td>2</td>
<td>99.9%</td>
<td>2.13</td>
</tr>
<tr>
<td>B1-4</td>
<td>20-17</td>
<td>4</td>
<td>1 16 8</td>
<td>159K-192K</td>
<td>BTRRI</td>
<td>Sampling-based</td>
<td>1</td>
<td>✗</td>
<td>1/4</td>
<td>2</td>
<td>23.3% - 51.2%</td>
<td>0.06 - 0.11</td>
</tr>
<tr>
<td>B5-6</td>
<td>16-48</td>
<td>4</td>
<td>1 16 8</td>
<td>44K-50K</td>
<td>BTRRI</td>
<td>Sampling-based</td>
<td>1</td>
<td>✗</td>
<td>1/4</td>
<td>2</td>
<td>99.9%</td>
<td>1.85 - 2.03</td>
</tr>
<tr>
<td>B7</td>
<td>19-06</td>
<td>8</td>
<td>2 16 8</td>
<td>20K</td>
<td>BTRRI</td>
<td>Sampling-based</td>
<td>1</td>
<td>✗</td>
<td>1/4</td>
<td>2</td>
<td>99.9%</td>
<td>31.14</td>
</tr>
<tr>
<td>B8</td>
<td>18-03</td>
<td>4</td>
<td>1 16 8</td>
<td>43K</td>
<td>BTRRI</td>
<td>Sampling-based</td>
<td>1</td>
<td>✗</td>
<td>1/4</td>
<td>2</td>
<td>99.9%</td>
<td>2.57</td>
</tr>
<tr>
<td>B9-12</td>
<td>19-48</td>
<td>8</td>
<td>1 16 8</td>
<td>42K-65K</td>
<td>BTRRI</td>
<td>Sampling-based</td>
<td>1</td>
<td>✗</td>
<td>1/4</td>
<td>2</td>
<td>36.3% - 38.9%</td>
<td>16.83 - 24.26</td>
</tr>
<tr>
<td>B13-14</td>
<td>20-08</td>
<td>8</td>
<td>1 16 8</td>
<td>11K-14K</td>
<td>BTRRI</td>
<td>Sampling-based</td>
<td>1</td>
<td>✓</td>
<td>1/2</td>
<td>4</td>
<td>99.9%</td>
<td>16.20 - 18.12</td>
</tr>
<tr>
<td>C0-3</td>
<td>16-48</td>
<td>4</td>
<td>1 16 x8</td>
<td>137K-194K</td>
<td>CTRRI</td>
<td>Mix</td>
<td>Unknown</td>
<td>✓</td>
<td>1/17</td>
<td>2</td>
<td>10.0% - 23.2%</td>
<td>0.05 - 0.15</td>
</tr>
<tr>
<td>C4-6</td>
<td>17-12</td>
<td>8</td>
<td>1 16 x8</td>
<td>130K-150K</td>
<td>CTRRI</td>
<td>Mix</td>
<td>Unknown</td>
<td>✓</td>
<td>1/17</td>
<td>2</td>
<td>7.8% - 12.0%</td>
<td>0.06 - 0.08</td>
</tr>
<tr>
<td>C7</td>
<td>20-31</td>
<td>8</td>
<td>1 8 x16</td>
<td>40K-44K</td>
<td>CTRRI</td>
<td>Mix</td>
<td>Unknown</td>
<td>✓</td>
<td>1/17</td>
<td>2</td>
<td>39.8% - 41.8%</td>
<td>9.66 - 14.56</td>
</tr>
<tr>
<td>C9-11</td>
<td>20-31</td>
<td>8</td>
<td>1 8 x16</td>
<td>42K-53K</td>
<td>CTRRI</td>
<td>Mix</td>
<td>Unknown</td>
<td>✓</td>
<td>1/9</td>
<td>2</td>
<td>99.7%</td>
<td>9.30 - 32.04</td>
</tr>
<tr>
<td>C12-14</td>
<td>20-46</td>
<td>16</td>
<td>1 8 x16</td>
<td>6K-7K</td>
<td>CTRRI</td>
<td>Mix</td>
<td>Unknown</td>
<td>✓</td>
<td>1/8</td>
<td>2</td>
<td>99.9%</td>
<td>4.91 - 12.64</td>
</tr>
</tbody>
</table>

Table 1 in the paper provides more information about the analyzed modules.
Case Study: Understanding Vendor A’s TRR

Refresh Types:
• Regular Refresh (RR)
• TRR-capable Refresh (TREF\textsubscript{1} and TREF\textsubscript{2})

Observation: TRR tracks potentially aggressor rows using a Counter Table

TREF\textsubscript{1}: Refreshes the victims of row ID with the largest counter value

TREF\textsubscript{2}: Refreshes the victims of row ID that TREF\textsubscript{2} pointer refers to
Case Study: Circumventing Vendor A’s TRR

**Approach:** Ensure an aggressor row is **discarded** from the Counter Table **prior** to a REF command.

**Counter Table**

<table>
<thead>
<tr>
<th>row ID</th>
<th>counter value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>row ID</th>
<th>counter value</th>
</tr>
</thead>
</table>

(A1, A2)

**RR**

**TREF<sub>1</sub>**

**TREF<sub>2</sub>**

**REF**

---

**REF** → **ACT** ([A<sub>1</sub>, A<sub>2</sub>]) → **ACT**(D<sub>1</sub>) → **ACT**(D<sub>2</sub>) → ... → **ACT**(D<sub>16</sub>) → **REF**

\[N \text{ times} \quad N+1 \text{ times} \quad N+1 \text{ times} \quad N+1 \text{ times}\]

A<sub>i</sub>: aggressor row

D<sub>i</sub>: dummy row

This RowHammer access pattern requires **synchronizing accesses** with **REF commands**.

---

Circumventing Vendor A’s TRR by **discarding** the actual **aggressor rows** from the Counter Table.
We craft **new RowHammer access patterns** that circumvent TRR of three major DRAM vendors.

On the **45** DDR4 modules we test, the **new access patterns** cause a large number of RowHammer bit flips.
Effect on Individual Rows

All 45 modules we tested are **vulnerable** to our new RowHammer access patterns.

For many modules, our RowHammer access patterns cause bit flips in more than **99.9% of the rows**.

Why are some modules less vulnerable?

1) Fundamentally less vulnerable to RowHammer
2) Different TRR mechanisms
3) Unique row organization
Effect on Individual Rows

All 45 modules we tested are vulnerable to our new RowHammer access patterns.

For many modules, our RowHammer access patterns cause bit flips in more than 99.9% of the rows.

Our access patterns successfully circumvent the TRR implementations of all three major DRAM vendors by unique row organization.
Other Observations and Results in the Paper

- More observations on the TRRs of the three vendors
- Analysis on the effectiveness of ECC against our RowHammer access patterns
- Detailed description of the crafted access patterns
- Hammers per aggressor row sensitivity analysis
- Observations and results for individual modules
- …
Target Row Refresh (TRR):
a set of obscure, undocumented, and proprietary RowHammer mitigation techniques

We cannot easily study the security properties of TRR

Is TRR fully secure? How can we validate its security guarantees?

A new methodology that leverages data retention failures to uncover the inner workings of TRR and study its security

U-TRR

15x Vendor A DDR4 modules
15x Vendor B DDR4 modules
15x Vendor C DDR4 modules

U-TRR

New RowHammer access patterns

All 45 modules we test are vulnerable
99.9% of rows in a DRAM bank experience at least one RowHammer bit flip
Up to 7 RowHammer bit flips in an 8-byte dataword, making ECC ineffective

U-TRR can facilitate the development of new RowHammer attacks and more secure RowHammer protection mechanisms

SAFARI

SoftMC (HPCA'17) U-TRR (MICRO'21) SMD (Ongoing) CROW (ISCA'19)
Contributions

A flexible and easy-to-use FPGA-based DRAM characterization infrastructure

SoftMC
HPCA’17

U-TRR
MICRO’21

A new methodology for uncovering in-DRAM RowHammer Protection Mechanisms

Self-Managing DRAM: Enabling autonomous and efficient in-DRAM maintenance operations

SMD
Ongoing
arXiv’22

CROW
ISCA’19

A low-cost substrate for improving DRAM performance, energy efficiency, and reliability

SAFARI
Problem: The Rigid DRAM Interface

The **Memory Controller** manages DRAM maintenance operations

Changes to **maintenance operations** are often reflected to the memory controller design, DRAM interface, and other system components

Implementing new maintenance operations (or modifying the existing ones) is **difficult-to-realize**
A Prime Example: New Features of DDR5

**DRAM Refresh**

**Same Bank Refresh** - simultaneously refreshes one bank in each bank group

The new **REFsb** command requires changes in DRAM interface and memory controller.

**Refresh Management (RFM)** — memory controller issues the new RFM command to allow DRAM chips more time to perform victim row refresh.

The new **RFM** command requires changes in DRAM interface and memory controller.

**In-DRAM Scrubbing**

**DDR5** changes are **difficult-to-implement** as they were **only** possible after **multiple years** required to develop a new DRAM standard.

**Memory Scrubbing**

**In-DRAM Scrubbing** – DDR5 uses the on-die ECC to perform periodic scrubbing.

The new **scrub** command requires changes in DRAM interface and memory controller.
Self-Managing DRAM (SMD) enables autonomous in-DRAM maintenance operations.

Key Idea:
Prevent the memory controller from accessing DRAM regions that are under maintenance by rejecting row activation (ACT) commands.

Leveraging the ability to reject an ACT, a maintenance operation can be implemented completely within a DRAM chip.
A DRAM bank is divided into configurable-size **Lock Regions**.

SMD marks regions *locked* for maintenance in the **Lock Region Table (LRT)**.

SMD rejects *any* ACT command that targets a *locked region* by sending the memory controller an **ACT_NACK** signal.
SMD: High-Level Operation

LRT

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>locked</td>
</tr>
<tr>
<td>1</td>
<td>available</td>
</tr>
<tr>
<td>2</td>
<td>available</td>
</tr>
<tr>
<td>3</td>
<td>available</td>
</tr>
</tbody>
</table>

ACT Retry Interval (ARI)

T_{ACK\_NACK}

bank 0
lock region 0

ACT

bank 0
lock region 1

ACT

ACT\_NACK

ACT\_NACK

RD

PRE

tRCD

tRAS

time

LRT

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>available</td>
</tr>
<tr>
<td>1</td>
<td>available</td>
</tr>
<tr>
<td>2</td>
<td>available</td>
</tr>
<tr>
<td>3</td>
<td>available</td>
</tr>
</tbody>
</table>

SAFARI

SoftMC (HPCA'17)  U-TRR (MICRO'21)  SMD (Ongoing)  CROW (ISCA'19)
### SMD-Based Maintenance Mechanisms

<table>
<thead>
<tr>
<th>DRAM Refresh</th>
<th>Fixed Rate (SMD-FR)</th>
<th>Variable Rate (SMD-VR)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>uniformly</strong> refreshes all DRAM rows with a <strong>fixed</strong> refresh period</td>
<td><strong>skips</strong> refreshing rows that can <strong>retain their data for longer</strong> than the default refresh period</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RowHammer Protection</th>
<th>Probabilistic (SMD-PRP)</th>
<th>Deterministic (SMD-DRP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performs <strong>neighbor row refresh</strong> with a <strong>small probability</strong> on every row activation</td>
<td><strong>keeps track</strong> of most frequently activated rows and performs <strong>neighbor</strong> row refresh when activation count threshold is exceeded</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory Scrubbing</th>
<th>Periodic Scrubbing (SMD-MS)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>periodically scans the entire</strong> DRAM for errors and corrects them</td>
</tr>
</tbody>
</table>

**SAFARI**

- SoftMC (HPCA’17)
- U-TRR (MICRO’21)
- SMD (Ongoing)
- CROW (ISCA’19)
Methodology

• Simulators
  • Ramulator [Kim+, CAL’15]  
    https://github/CMU-SAFARI/ramulator
  • DRAMPower [Chandrasekar+, DSD’11]  
    https://github.com/tukl-msd/DRAMPower

• Workloads
  • 44 single-core workloads  
    SPEC CPU2006, TPC, STREAM, MediaBench
  • 60 multi-programmed four-core workloads  
    By randomly choosing from single-core workloads

• System Parameters
  • 4-channel dual-rank DDR4 DRAM
  • 32ms default refresh period
Single-Core Results

**Speedup**
- SMD-FR
- SMD-VR
- SMD-PRP
- SMD-DRP
- SMD-MS

**Normalized DRAM Energy**
- SMD-FR
- SMD-VR
- SMD-PRP
- SMD-DRP
- SMD-MS
- SMD Combined
- NoRefresh

- 9.2% increase
- 9.0% normalized DRAM energy

**Techniques**
- SoftMC (HPCA'17)
- U-TRR (MICRO'21)
- SMD (Ongoing)
- CROW (ISCA'19)
SMD-based maintenance mechanisms have significant performance and energy efficiency benefits
Sensitivity to Refresh Period

SMD-based refresh mechanisms will be even more beneficial in future DRAM chip with low refresh periods.
Hardware Overhead

Interface Modifications
• A single ACT_NACK pin per DRAM chip

DRAM Chip Modifications
• Lock Region Table incurs only:
  • 32um² area overhead (0.001% of a 45.4mm² DRAM chip)
  • 0.053ns access latency overhead

Memory Controller Modifications
• Changes in request scheduling to handle ACT_NACK signals

• No further changes needed for new maintenance operations
• The memory controller no longer manages DRAM maintenance
Other Results in the Paper

• Lock region size sensitivity

• Comparison to memory controller-based RH protection

• Comparison to memory controller-based scrubbing

• SMD-DRP maximum activation threshold sensitivity

• Victim row window sensitivity
The three major DRAM maintenance operations:
- Refresh
- RowHammer Protection
- Memory Scrubbing

Implementing new maintenance mechanisms often requires difficult-to-realize changes.

Our Goal
1. Ease the process of enabling new DRAM maintenance operations
2. Enable more efficient in-DRAM maintenance operations

Self-Managing DRAM (SMD)
Enables implementing new in-DRAM maintenance mechanisms with no further changes in the DRAM interface and memory controller.

SMD-based refresh, RowHammer protection, and scrubbing achieve 9.2% speedup and 6.2% lower DRAM energy vs. conventional DRAM.

Source code will be available soon: [github.com/CMU-SAFARI/SelfManagingDRAM](github.com/CMU-SAFARI/SelfManagingDRAM)
Contributions

A flexible and easy-to-use FPGA-based DRAM characterization infrastructure

SoftMC
HPCA'17

U-TRR
MICRO'21

A new methodology for uncovering in-DRAM RowHammer Protection Mechanisms

Self-Managing DRAM: Enabling autonomous and efficient in-DRAM maintenance operations

SMD
Ongoing
arXiv'22

CROW
ISCA'19

A low-cost substrate for improving DRAM performance, energy efficiency, and reliability
Scaling Challenges of DRAM Technology

DRAM Cell

DRAM Scaling Problems

- Data Integrity
- Latency

System-Level Problems

- Performance
- Energy Efficiency
- Reliability
- Security

SAFARI

SoftMC (HPCA'17) U-TRR (MICRO'21) SMD (Ongoing) CROW (ISCA'19)
Observation

Modifying the underlying *density-optimized* DRAM cell array structure may incur non-negligible *area overhead*
Our Goal

1. Lower DRAM latency
2. Reduce refresh overhead
3. Improve DRAM reliability

- **No changes** to the DRAM cell structure
- **Only minimal** changes to the DRAM chip

SAFARI

SoftMC (HPCA'17) → U-TRR (MICRO'21) → SMD (Ongoing) → CROW (ISCA'19)
A **flexible substrate** that enables new mechanisms for **improving** DRAM:

- **Performance**
- **Energy Efficiency**
- **Reliability**
- **Security**

**Key Idea:**

1) **efficiently** duplicate select rows in DRAM
2) exploit **duplicated rows** in new mechanisms
CROW: High-Level Overview

CROW enables:

1) **Row Copy**: efficiently duplicating data from a regular row to a copy row
2) **Two-row Activation**: quick access to a duplicated row
CROW Operation 1: Row Copy

DRAM Subarray

regular rows

copy rows

ACT-c (copy)

Memory Controller

SA
SA
SA
SA
SA
SA

regular row decoder

CROW decoder

SA
SA
SA
SA
SA
SA

CROW (ISCA’19)

SoftMC (HPCA’17) U-TRR (MICRO’21) SMD (Ongoing)
Row Copy: Steps

Enables quickly copying a regular row into a copy row

1. Activation of the source row
2. Charge sharing
3. Beginning of restoration
4. Activation of the destination row
5. Restoration of both rows

source row:

destination row:
CROW Operation 2: Two-Row Activation

DRAM Subarray

regular rows

copy rows

ACT-t (two row)

Memory Controller

SAFARI

SoftMC (HPCA'17) U-TRR (MICRO'21) SMD (Ongoing) CROW (ISCA'19)
Two-Row Activation: Steps

1. Activation of two rows
2. Charge sharing  **fast**
3. Restoration

Enables fast access to data that is duplicated across a regular row and a **copy row**.
CROW-based Mechanisms

CROW-cache

CROW-ref

Mitigating RowHammer
CROW-cache

**Problem:** High access latency

**Key idea:** Use *copy rows* to enable low-latency access to most-recently-activated regular rows in a subarray

CROW-cache combines:

- **row copy** → copy a newly activated regular row into a *copy row*
- **two-row activation** → activate the regular row and *copy row* together on the next access

**Reduces** activation latency by *38%*
CROW-cache Operation

Request Queue

1. CROW-table miss
2. Allocate a copy row
3. Issue ACT-c (copy)

Second activation of row X is faster
CROW-based Mechanisms

- CROW-cache
- CROW-ref

Mitigating RowHammer
**Problem:** Refresh has high overheads. Weak rows lead to high refresh rate

- **weak row:** at least one of the row’s cells cannot retain data correctly when refresh rate is decreased

**Key idea:** Safely reduce refresh rate by remapping a **weak** regular row to a **strong** copy row

CROW-ref uses:

- **row copy** → copy a weak regular row to a strong copy row

CROW-ref **eliminates more than half of the refresh** requests
CROW-ref Operation

1. Perform retention time profiling
2. Remap weak rows to strong copy rows
3. On ACT, check the CROW-table
4. If remapped, activate a copy row

How many weak rows exist in a DRAM chip?
Identifying Weak Rows

Weak cells are rare [Liu+, ISCA’13]
weak cell: retention < 256ms
~1000/2^{38} (32 GiB) failing cells

DRAM Retention Time Profiler
• REAPER [Patel+, ISCA’17]
  PARBOR [Khan+, DSN’16]
  AVATAR [Qureshi+, DSN’15]

A few copy rows are sufficient to halve the refresh rate
CROW-based Mechanisms

CROW-cache

CROW-ref

Mitigating RowHammer
Mitigating RowHammer

Key idea: remap victim rows to copy rows
Methodology

• Simulator
  • DRAM Simulator (Ramulator [Kim+, CAL’15])
    https://github.com/CMU-SAFARI/ramulator

• Workloads
  • 44 single-core workloads
    • SPEC CPU2006, TPC, STREAM, MediaBench
  • 160 multi-programmed four-core workloads
    • By randomly choosing from single-core workloads
  • Execute at least 200 million representative instructions per core

• System Parameters
  • 1/4 core system with 8 MiB LLC
  • LPDDR4 main memory
  • 8 copy rows per 512-row subarray

SAFARI

SoftMC (HPCA’17) U-TRR (MICRO’21) SMD (Ongoing) CROW (ISCA’19)
CROW-cache results

CROW-cache improves single-/four-core performance and energy

* with 8 copy rows and a 64Gb DRAM chip (sensitivity in paper)
CROW-ref Results

CROW-ref significantly reduces the performance and energy overhead of DRAM refresh.
Combining CROW-cache and CROW-ref

CROW-(cache+ref) provides more performance and DRAM energy benefits than each mechanism alone.

Speedup

<table>
<thead>
<tr>
<th></th>
<th>0.70</th>
<th>0.80</th>
<th>0.90</th>
<th>1.00</th>
<th>1.10</th>
<th>1.20</th>
<th>1.30</th>
</tr>
</thead>
<tbody>
<tr>
<td>CROW-(cache+ref)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ideal CROW-cache + no refresh</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Normalized DRAM Energy

<table>
<thead>
<tr>
<th></th>
<th>0.72</th>
<th>0.74</th>
<th>0.76</th>
<th>0.78</th>
</tr>
</thead>
<tbody>
<tr>
<td>CROW-(cache+ref)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ideal CROW-cache + no refresh</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hardware Overhead

For 8 copy rows and 16 GiB DRAM:
• 0.5% DRAM chip area
• 1.6% DRAM capacity
• 11.3 KiB memory controller storage

CROW is a low-cost substrate
Other Results in the Paper

• Performance and energy sensitivity to:
  • Number of copy-rows per subarray
  • DRAM chip density
  • Last-level cache capacity

• CROW-cache with prefetching

• CROW-cache compared to other in-DRAM caching mechanisms:
  • TL-DRAM \cite{Lee13_HPCA}
  • SALP \cite{Kim12_ISCA}
Summary

Copy-Row DRAM (CROW)

- Introduces copy rows into a subarray
- The benefits of a copy row:
  - Efficiently duplicating data from regular row to a copy row
  - Quick access to a duplicated row
  - Remapping a regular row to a copy row

Use cases of CROW:
- CROW-cache & CROW-ref
- Mitigating RowHammer
- We hope CROW enables many other use cases going forward

Source code available: github.com/CMU-SAFARI/CROW
Contributions

A flexible and easy-to-use FPGA-based DRAM characterization infrastructure

SoftMC  
HPCA’17

U-TRR  
MICRO’21

A new methodology for uncovering in-DRAM RowHammer Protection Mechanisms

Self-Managing DRAM: Enabling autonomous and efficient in-DRAM maintenance operations

SMD  
Ongoing arXiv’22

CROW  
ISCA’19

A low-cost substrate for improving DRAM performance, energy efficiency, and reliability

SAFARI
Future Research Directions

• **Deeper DRAM Characterization**
  • Analyzing cell characteristics of new devices
  • Impact of aging
  • Low temperature operation

• **Extending SoftMC**
  • Supporting other DRAM and NVM standards

• **Improving RowHammer Attacks & Defenses**
  • Studying the security properties of RowHammer protection mechanisms

• **New DRAM Maintenance Mechanisms**
  • Profiling-based maintenance operations
  • Memory controller and in-DRAM processing interoperability

• **Exploiting in-DRAM Data Movement**
  • More mechanisms that exploit low-overhead in-DRAM data migration