Counteracting Rowhammer by Data Alternation

Lung, Stefan A.; Gaydadjiev, Georgi; Hamdioui, Said; Taouil, Mottaqiallah

DOI
10.1109/ETS61313.2024.10567079

Publication date
2024

Document Version
Final published version

Published in
Proceedings - 2024 29th IEEE European Test Symposium, ETS 2024

Counteracting Rowhammer by Data Alternation

Stefan A. Lung, Georgi Gaydadjiev, Said Hamdioui, Mottaqiallah Taouil
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Delft, The Netherlands

2024 IEEE European Test Symposium (ETS) | 979-8-3503-4932-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/ETS61313.2024.10567079

***Abstract*—Modern DRAMs are vulnerable to Rowhammer attacks, demanding robust protection methods to mitigate these attacks. Existing solutions aim at increased resilience by improving design and/or adjusting operation parameters, limit row access count by throttling and prevent bit flips by timely row refreshing. However, scaling these methods for future DRAM technologies may incur significant costs in terms of area, power and/or latency. This study analyses the impact of the values of the neighbouring cells on victim cells and introduces a row alternation protection method, which is a novel approach that alternates the data of attacker rows on each access to lower the chance of bit flips in victim rows. Our analysis show that the minimum Rowhammer count to cause a bitflip in a particular cell does not only depend on vertical neighbours from the attacker row, but also on the value of the horizontal neighbours from the victim row as well as diagonal cells from the attacker row. Row alternation is able to protect the majority of the vulnerable cells (i.e., with 65%) for the DRAM used in our case study and in cases where unsuccessful it significantly increases the average minimum required Rowhammer account by 18%.**

I. Introduction

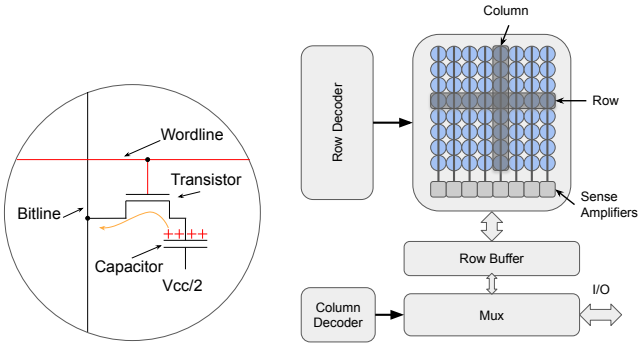
DRAM has become an essential component in modern computer systems and evolved over the many years to increase it's density and capacity each generation [1]. However, pushing the technology to its limits has come at a cost as modern DRAM memories suffer from electrical disturbances induced by specific access patterns causing unwanted flipping of bits, as discussed in one of the first studies on DRAM memory [2]. In 2015, Google's Project Zero [3] showed that it is possible to gain kernel privileges by means of a Rowhammer attack, i.e., repeatedly hammering certain rows in the DRAM. The results in [4] show that newer DRAM devices are becoming increasingly more vulnerable to Rowhammer as they require significantly fewer row accesses to inflict bit flips in neighbouring rows. To prevent such attacks, it is important to design countermeasures with negligible impact on performance, area, latency and power.

Several hardware-based protection methods have been developed to mitigate the effects of Rowhammer. These can be categorized in three groups: *1)* throttling the access rate of attacker rows in order to prevent reaching the required rowhammers within a refresh period; *2)* recharging victim rows before bit flips occur; and *3)* making a more robust design by adjusting operating parameters, altering the design or adding steps in the production process. An example of the first group is BlockHammer [5], which tracks row accesses and prevents rows from being accessed too frequently. In the second group, timely recharging of victim rows, several countermeasures exist. Examples are PARA [6], PRoHit [7] and MRLoc [8]; they all access victim rows with a certain probability after a neighbouring attacker row is accessed. A second subgroup uses counters to determine when to refresh rows. For example, CBT [9], TWiCe [10] and an ideal refresh-based method [11] take action depending on the neighbouring attacker row access count. In the third group, i.e., creating a more robust design, also several countermeasures exist. In [12], the wordline voltage is reduced to lower the effectiveness of a Rowhammer attack, which requires a higher access count to flip the victim row. In [2], the authors suggest increasing the refresh rate to the minimum rowhammers required by an attacker to perform an attack and remapping vulnerable cells to spare cells like performed for defective cells. In [13], hydrogen annealing as an extra step in the production process to increase the DRAM's resilience is proposed. All of the above countermeasures do not distinguish between the hammering frequency and the data values the attacker is applying. In [14], the authors showed that the minimum rowhammers to induce a successful bit flip is 50% lower when the attacker uses uncharged cells instead of charged ones.

In this paper, we analyse Rowhammer in-depth by taking the impact of both victim and attacker cell values into account. Thereafter, we propose a novel, low-cost protection method called *row alternation*. Data written back from the DRAM Row Buffer will be inverted. By alternating the data in the memory, the victim cell discharge is minimally impacted by the value of the aggressor cells. During a write operation, the row buffer cells are randomly written to either their true or complement value to prevent an attacker from anticipating the cell charges. The contributions of this paper are:

- A novel row alternation protection method that reduces the acceleration of victim cell discharge caused by Rowhammer.
- The validation and evaluation of row alternation.
- In-depth analysis of the impact of victim and aggressor cell values during Rowhammer.

The remainder of this paper is organized as follows. Section II provides a background on DRAM and Rowhammer. Section III introduces the row alternation protection method. Section IV describes the experimental setup and list of experiments. Section V presents the experimental results. Finally, Section VI discusses and concludes this paper.



(a) DRAM Cell (b) DRAM Architecture

Fig. 1: DRAM Cell and Architecture

II. Background

This section provide the basics of DRAM and Rowhammer. First a general DRAM architecture is discussed followed by the principle behind Rowhammer.

A. DRAM

DRAMs store their information in cells by means of capacitive charges. A cell consists of a transistor and a capacitor as shown in Figure 1a. As cells lose charge over time due to leakage, a refresh command is periodically issued to restore the capacitor charge to its initial value. Figure 1b shows a basic DRAM architecture. Its cells are arranged in a two dimensional matrix; cells on a row are connected to a wordline and on a column to a bitline. A wordline can be activated at a time by the row decoder depending on the selected address. When a row is active, the transistor allows access to the capacitor, i.e., the cell and the bitline will exchange charge. In case the capacitor is charged, charge will flow from the capacitor to the bitline. In case discharged, charge will flow from the bitline to the capacitor. Once a row is activated, the charge on the cells in this row are read out via the bitlines using *sense amplifiers*; they measure the voltage on the bitlines to determine the DRAM cell content. Depending on whether the cell is initially charged or not the sense amplifier will output a bit value of 1 or 0. True-cells will have a bit value of 1 and anti-cells a bit value 0 when charged, and opposite a 0 and 1 respectively when discharged. The output of the sense amplifiers are subsequently stored in the *Row Buffer*. Finally, a *Column Decoder* is used to select the desired column bits from or to the *Row Buffer*.

B. Rowhammer

Rowhammer is a security exploit in DRAM memories where frequently accessed attacker rows cause bit flips in neighbouring victim rows. In [2], the authors attribute bit flips in victim rows due to frequently issuing $activate \rightarrow precharge$ command sequences on a single row. As read and write commands are limited to interaction with the *Row Buffer*, a Rowhammer attack is performed by accessing at least two distinctive rows consecutively to ensure the clearing of the Row Buffer.

During a successful Rowhammer attack, the victim's cell discharge rate is accelerated causing the cell charge to drop below the read reference voltage within the refresh period. Studies have shown that the induced bit error rate of DRAM devices as a consequence of Rowhammer differs between manufacturers, generations and types, and that the effectiveness of Rowhammer on these DRAM modules varies [2, 4]. Unintended bit flips only occur in victim cells that are charged [14]; this can be a $1 \rightarrow 0$ transition for true-cells and $0 \rightarrow 1$ transition for anti-cells. The same study showed that uncharged attacker cells have a higher potential to flip victim cells as compared to charged attacker cells. The speed at which the discharging of the victim cells takes place during Rowhammer depends on two factors, namely, electron de-trapping [14, 15] and wordline-crosstalk [16]. A high temperature and longer wordline activation time may increase the vulnerability of cells to Rowhammer [11].

III. Row Alternation Protection Method

This section discusses the proposed row alternation protection method. First it discusses the concept followed by design and implementation choices.

A. Concept

As demonstrated in [14], the minimum Rowhammer count depends strongly on the applied attacker data. Uncharged attacker cells have a higher potential to flip victim cells. In this work we target to understand this impact and propose a countermeasure against it. The aim is to inhibit the acceleration of the victim cell's discharge rate by having some control on the attacker's cell data by altering this data throughout the Rowhammer attack; as a result, attacker cells will be 50% charged and 50% uncharged during the Rowhammer attack and hence improving the worst case where the attacker cells are 100% uncharged. Figure 2 shows this principle. The idea behind the row alternation protection method is to reduce the victim cell's voltage drop such that a refresh command successfully can restore the cell to its initial state. To realize this, the Row Buffer can be used. Data in the Row Buffer is written back to the memory in case another operation is performed on a different row. By writing the complemented value of the Row Buffer back to memory, the attacker has no control anymore of the values written back to the cells. However, the attacker can still control the write data to the Row Buffer through a write operation and counteract the inversions from the row buffer. To prevent that, written data should also be complemented with a 50% probability.

B. Design and Implementation Choices

Figure 3 shows the modifications made to the DRAM array to implement the row alternating protection method. This method does not distinguish between malicious and non-malicious row accesses. A single column is added to the DRAM array to store the state of each row (i.e., true or complement value). This state bit is also loaded into the *Row Buffer* during a row activation.

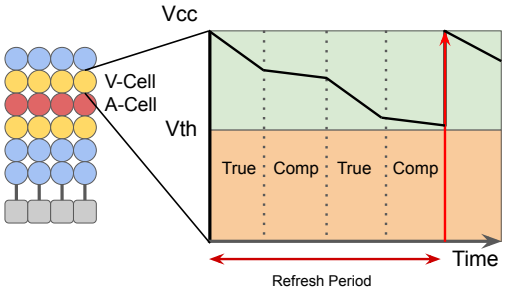


Fig. 2: Impact of Row Alternation on Discharge Rate

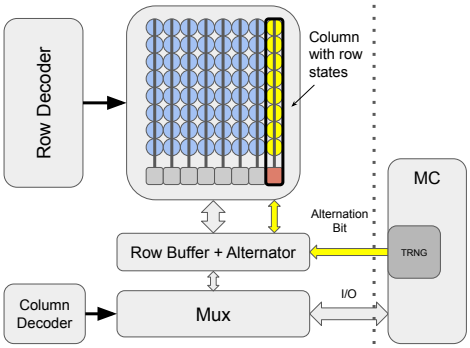


Fig. 3: DRAM Architecture with Status Bit

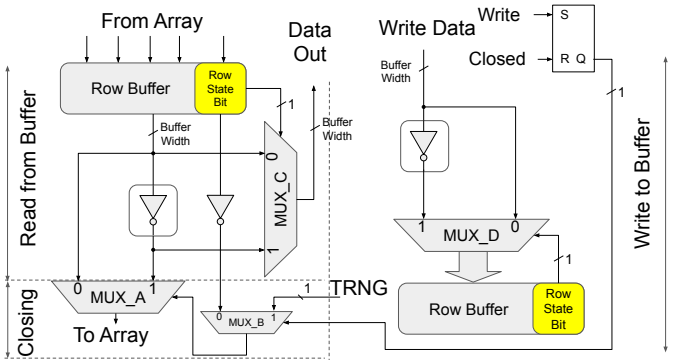


Fig. 4: Row Alternation Logic

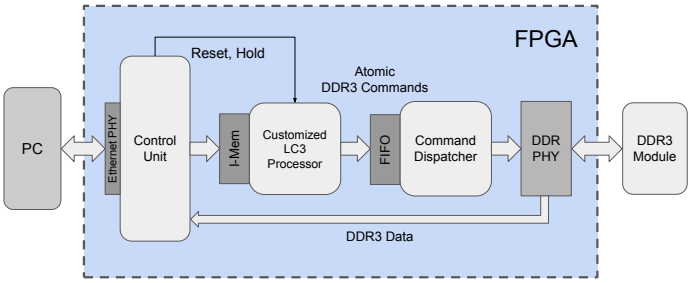


Fig. 5: Experimental Platform

Figure 4 shows a potential implementation of the logic needed for the *row alternation* protection method. Reading from a DRAM bank is shown in the left side. A row is accessed with the corresponding status bit. The true or complement data is subsequently forwarded to the output (Data Out) depending on the *Row State Bit*. Writing the Row Buffer back to the memory, i.e., when the row is closed, depends on whether a write operations has been performed since the previous close operation. We use a latch to keep track of that as shown in the top right of the figure. In case no write operation has been performed, MUX_B is used to forwarded the inverted value of the status bit, i.e, the inverted values of the Row Buffer will be written back to the memory. In case a write operation did take place, a random bit value will be used to determine the new value of the Row State Bit and whether true or complementary data is written to the cells. For simplicity, we duplicated the Row Buffer in the figure. The random bit value is denoted by TRNG in the figure which can be provided by the memory controller. After the row closes, the latch is reset. The random factor from the TRNG ensures that an attacker cannot anticipate row alternation during write operations.

IV. Experimental Setup

This section discusses the experimental platform followed by the performed experiments.

A. Experimental Platform

Figure 5 shows our experimental platform. It consists of 3 main components: host PC, FPGA and DRAM3 modules. The host PC is used to create programs that perform experiments that are run on the FPGA. A program consists of assembly instructions that are converted by a custom assembler to binary instructions. These binary instructions are transmitted by an Ethernet connection to the FPGA. The host PC is also responsible for data analysis that is collected on the FPGA, e.g., the results of a Rowhammer experiment.

We have used a Digilent Genesys 2 [17] FPGA development board as testing platform. The FPGA receives programs from the PC, executes them and send the results back. The control unit on the FPGA is responsible for receiving a program from the host PC and starting the experiment. It places all instructions inside an instruction memory. The instruction memory is read out by a customized processor which is based on the LC3 instruction set [18]. The LC3 instruction set is extended with custom atomic DRAM instructions such as activate, read, write and precharge. These DRAM related instructions are forwarded to a Command Dispatcher unit that is responsible for their forwarding to the DDR3 Physical Interface (PHY) [19]. The DDR3 PHY receives the memory commands at specific timings according to the spec, allowing the execution of DDR3 commands with maximum data throughput. The DDR3 PHY is part of A *DRAM controller* interface generated from the Xilinx memory interface generation [20]. The DDR3 PHY interfaces with the two SDRAM DDR3 modules on the development board. They have a width of 16-bit each and a maximum operating frequency of 800 MHz. Each module has 8 banks, i.e., 2^{15} rows and 2^{10} column addresses. The exact operating conditions can be found in [21].

B. Performed Experiments

There are three sets of experiments; they are: (1) DRAM characterization, (2) cell vulnerability analysis and (3) evaluating the row alternation protection method.

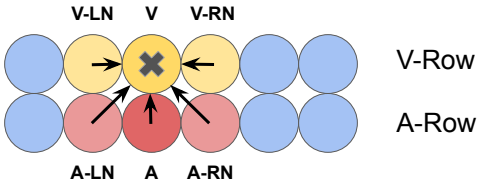


Fig. 6: Directions of influence on the victim cell

Set 1: DRAM Characterization - The target of this experiment is to understand the architecture of the DRAM. First, we identify the true and anti cells using a retention test with a solid 1 and 0 data pattern, respectively. Rows that have bit errors after a solid 1 retention test are true cells; these are cells that are initially charged and lose their charge after a long time. Similarly, anti cells can be identified by applying a solid 0 data pattern. Using this information, we can identify the neighbour rows of each row by performing Rowhammer attacks. First, all cells are charged by writing ones to true cells and zeros to anti cells. Thereafter, Rowhammering is performed on a single row. The rows that contain bit errors after the Rowhammer attack are identified as neighbours of the attacker row. Finally, the column layout can be derived by performing a Rowhammer attack with a walking uncharged cell pattern on an attacker row, while having the victim row cells all charged. This allows us to identify whether the column indices of the attacker cells and victim cells are vertically aligned. We assume that the column indices in general increase linear in order from LSB to MSB, e.g., cells in column 5 are to the left of cells in column 4 and to the right of cells in column 6.

Set 2: Cell Vulnerability Analysis - The target of this experiment is to identify the vulnerable cells. They are found by performing a Rowhammer sequence of $activate \rightarrow precharge$ within a refresh period (64 ms). Figure 6 shows how a victim cell could be influenced by its neighbours during a Rowhammer attack. The aggressor cell in red is denoted by A and the victim cell in yellow by V. We consider the left neighbour (LN) and right neighbour (RN) of the aggressor and victim cell as well during the attack. The three cells on the victim and attacker row can both take 8 different values. Hence, 8*8=64 patterns for each victim cell will be analyzed.

Set 3: Row Alternation Evaluation - The target of this experiment is to understand how well row alternation works as a protection method. We replace the $activate \rightarrow precharge$ hammer sequence with actual read and/or write commands. As the row alternation protection method requires the flipping of data in the row buffer, we can only simulate the attacker behaviour using write operations. This experiment will give an indication of how well the protection method works.

V. Results

This section provides the results of the experiments.

Set 1: DRAM Characterization - The DRAM contains true and anti-cells that are alternated each 680 or 688 rows. Each row contains 16,384 cells. We observed for this DRAM that an attacker row only influences one neighbouring victim at a time and hence that rows are isolated every two rows. This fits well with the *Open Memory Array Structure* described in [1]. We also verified that two rows are vertically aligned, meaning that the attacker and victim cell column indices are the same.

TABLE I: Impact of Background patterns on Bit Flipping

Data Pattern	Total Bit Errors	Bit Error Rate
Solid 1	9973	0.0037%
Solid 0	11020	0.0040%
Rowstripe	354929	0.1313%
∼Rowstripe	327193	0.1209%

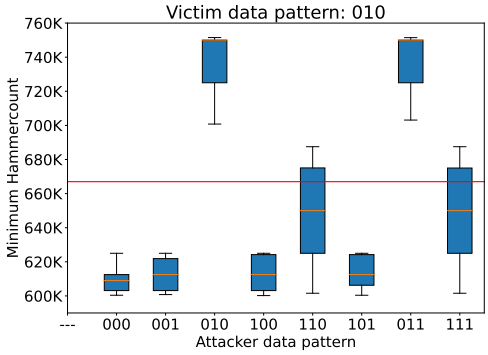
Set 2: Cell Vulnerability Analysis - Table I shows the total number of bit flips caused by rowhammering by maximizing the number of hammering (i.e., an amount of 667k) within a refresh period for different background patterns. For the true cells, the solid 1 and rowstripe data patterns are relevant. In solid 1 both the victim and aggressor cells are charged. In the rowstripe data pattern the victim cells are charged, while the aggressor cells discharged. Similarly, solid 0 and ∼rowstripe (i.e., inverted rowstripe) are applied to the anti cells. From the table, we conclude that that the rowstripe and ∼rowstripe cases (where the victim cells are charged and aggressors uncharged) lead to the largest amount of bitflips, justifying a countermeasure like row alternation.

We have selected *40* victim cells to do further analysis using the 64 patterns as shown in Figure 6. A binary search method is used to determine the resilience of the victim cell in terms of minimal hammercount needed to flip the victim cell. Each specific attacker/victim data pattern is repeated 100 times.

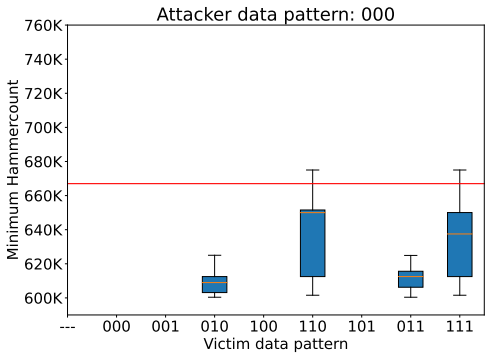
Similarly as previous work, we observed bit-flips only in the cases where the victim is charged. Moreover, most victim cells are only vulnerable when the attacker data is uncharged.

Figure 7(a) shows for a true victim cell the impact of the aggressor cells applied as shown in Figure 6. The victim pattern is fixed to 010 (i.e., left and right cell of the victim are 0 and hence uncharged, while the victim is one and hence charged) and 8 patterns for the attacker have been evaluated. The red line represent the maximum hammer count within a refresh period. When the direct aggressor bit is 0, the attack has the highest chance to be successful as they need the least number of rowhammers. The left bit of the main aggressor cell also has some impact; for example the minimum Rowhammer count is reasonably higher for attacker pattern **0**10 as compared to **1**10. However, the impact of the left bit is marginal when the main aggressor cell is 0 (e.g., there is less difference between 000 and 100). The right bit has no impact as can be seen for example by comparing attack patterns 000 and 001 and attack patterns 010 and 011.

For the particular cell in Figure 7(a), successful rowhammers where also possible when the aggressor cell was charged. We did not observe this behaviour frequently elsewhere in the DRAM. We also did not observe a lot of impact from the attacker cell's left or right neighbours. However when we did observe such impact, the left or right neighbours of the aggressor cell should always be charged for maximum impact. We do not exactly understand this behaviour.



(a) Attacker Impact



(b) Victim Impact

Fig. 7: Successful Bit Flips on a True Victim Cell

Figure 7(b) shows for the same cell the impact of the victim neighbours. This cell is only vulnerable to Rowhammer attacks when the victim is charged. Again we observe some impact of the victim's left neighbour cell. Opposite to the left cell of its aggressor, a zero value in the victim's left neighbour reduces the minimum Rowhammer count for a successful bit flip.

To speed up the experiments, we actually loaded multiple half words (16-bits) in the same column address region with the same aggressor-victim data patterns. In general, although not occurring that frequently, we observed that some even column bits where only impacted by left neighbours and odd column bits by right neighbours. From this we conclude that our assumption that all column indices increase linear from LSB to MSB is false. Most likely, the column indices are physically mapped in another way for example to reduce the amount of error correction in case of soft errors. Actually the neighbours turned out to be part of another half word. It is possible to identify the neighbours in the following way. First, a victim row is charged and attacker row uncharged. A potential horizontal neighbour of the victim is uncharged and rowhammering takes place. If the cell is actually a neighbour, the minimum Rowhammer count of the cell will reduce. We found that cells with a column addresses 0-63 had neighbours with column addresses 64-128 and the opposite is also true. To obtain the neighbour cell c_n of cell c, the formulas below can be used. If the column address is between 0-63 the top formula is applicable and for 64-128 the bottom one.

TABLE II: Impact of Row Alternation on 40 Victim Cells

Hammer Sequence	Max HC	Failures (No prot.)	Failures (Prot.)	Imp. rate
$(act \rightarrow rd \rightarrow pre)$	640,000	38	-	-
$(act \rightarrow wrt \rightarrow pre)$	609,523	26	9	65.38%

$$c_n = c + 64 + (3 - 2 \cdot (c \mod 4))$$

$$c_n = c - 64 + (3 - 2 \cdot (c \mod 4))$$

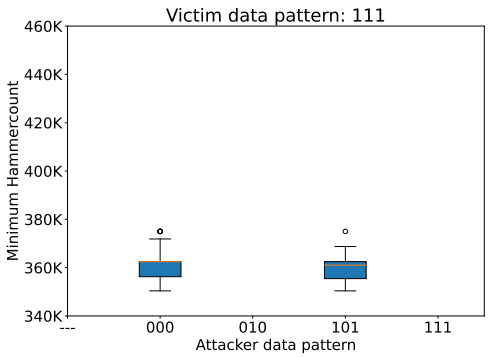
There was no need to further analyze the neighbours as their impact can easily be guaranteed by giving them the same value, e.g., victim and aggressor cell charged and uncharged, respectively and keeping the remaining cells on the victim and attacker rows charged or uncharged. Drawback is that in Figure 6 only 16 patterns remain of the originally 64.

Set 3: Row Alternation Evaluation - In this experiment, we evaluate the row alternation protection method using read and write operations. The Rowhammer sequences are shown in Table II. The table shows for each sequence the maximum hammercount (HC) within a refresh period of 64 ms, the number of failing cells with no protection, and finally (when applicable) the number of cells with protection enabled with the improvement rate. Here we performed the analysis only on the same 40 vulnerable cells used in the second experiment. However, as the hamming sequence is longer here the max HC account is lower and hence not all 40 cells where vulnerable for this sequence. Due to not being able to modify the DRAM architecture, we could only evaluate write sequences. With the proposed method we were able to protect around 65% of the cells. Without protection, the most vulnerable cell required 390k rowhammers and the least vulnerable one 607k. With protection, we increased the minimum hammer count on average by 18% for the cells that we could not protect.

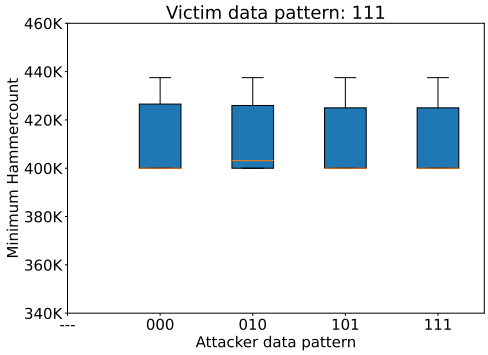
Figure 8 shows how the row alternation protection method works for a particular true cell without (part (a)) and with the protection method (part (b)). Part (a) of the figure shows a cell that is vulnerable to Rowhammer when the aggressor cell contains a 0. By applying row alternation, the attacker data always get balanced. As a result, the impact of Rowhammering will be independent of the attacker data pattern and all of them have similar impact. For the considered cell, the minimum Rowhammering count for a success bitflip has been increased from 350k to 400k. Note that for this particular cell it was not enough. This method does have a drawback, as row hammering is also possible for this particular cell when the aggressor data is 1. Note that to perform a successful Rowhammer attack that the attacker only needs to have access to a small part of the row, as the complete Row Buffer is written back in its entirety. The Row buffer is typically very wide (e.g., 16 kbits), this might introduce data dependent vulnerabilities on other parts in the memory.

VI. Discussion & Conclusion

In this paper, we have characterized a DRAM using reverse engineering. True and anti cells have been found together with the neighbouring rows. Our proposed method focused



(a) ($act \rightarrow write \rightarrow pre$) No protection



(b) ($act \rightarrow write \rightarrow pre$) With Row Protection

Fig. 8: Impact of Row Alternation on a True-Victim Cell

on altering the data in the attacker row. Results showed that 65% of vulnerable cells have been successfully protected. It is worth noting the following:

- **Overhead:** As Figure 3 and Figure 4 showed, the required modifications to the DRAM are minimal. Adding a column to the memory to store the status bits causes a 0.0006% overhead. The logic needed to invert the Row Buffer data is also very small and barely impacts the performance. In case the DRAM uses ECC, the protection method requires recalculating the ECC after inverting a row. To prevent such calculations, we could always save the ECC based on the non-inverter true values.
- **Comparison with state of the art:** Row alternation has purely been proposed against countering the data value of aggressor cells and not as a full protection scheme against Rowhammer. It is hard to compare our method against other types such as throttling the access rate, recharging victim rows on time and having a more robust design, as they use different principles. Moreover, the selected DRAM technology and operating conditions have a huge impact on the results. A fair comparison can only be made using the same platform and conditions. For example, row refreshing methods PARA [6] and PRoHit [7] have been compared to MRLoc in [8]. They show that PARA reduces the impact of Rowhammer the least and that PRoHit and MRLoc perform similar. Nevertheless, row data alternation is an independent method that can be combined with other schemes. Our low-cost solution could help reduce overhead of other schemes, as our scheme already reduces the impact of Rowhammer. More research is needed in this direction.
- **Applicability:** Altering the row data as a protection method is applicable for any type of DRAM and memory controller. The method works even when data scrambling is used as data scrambling is performed by the memory controller and the row alternation protection method in the DRAM device.
- **Limitations:** Row alternation works well against leakage due to electron de-trapping [14, 15] (as its impact is mostly determined by the data of neighbour cells), but is less effective against wordline-crosstalk [16] (as its impact is mostly determined by the Rowhammer amount).

REFERENCES

[1] B. Keeth *et al.*, *DRAM Circuit Design*. Wiley, 2007.

[2] Y. Kim *et al.*, "Flipping Bits in Memory without Accessing Them: An Experimental Study of DRAM Disturbance Errors," *ISCA*, 2014.

[3] M. Seaborn *et al.*, "Exploiting the DRAM rowhammer bug to gain kernel privilege," https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html, 2015, [Online; accessed 14-December-2023].

[4] J.S. Kim *et al.*, "Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques," *ISCA*, 2020.

[5] A.G. Yağlikçi *et al.*, "BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows," in *HPCA*, 2021, pp. 345–358.

[6] Y. Wang *et al.*, "Discreet-PARA: Rowhammer Defense with Low Cost and High Efficiency," in *(ICCD)*, 2021, pp. 433–441.

[7] M. Son *et al.*, "Making DRAM stronger against row hammering," in *(DAC)*, 2017, pp. 1–6.

[8] J.M. You *et al.*, "MRLoc: Mitigating Row-hammering based on memory Locality," in *(DAC)*, 2019, pp. 1–6.

[9] S.M. Seyedzadeh *et al.*, "Counter-Based Tree Structure for Row Hammering Mitigation in DRAM," *IEEE Computer Architecture Letters*, vol. 16, pp. 18–21, 2017.

[10] E. Lee *et al.*, "TWiCe: Preventing Row-hammering by Exploiting Time Window Counters," in *ISCA*, 2019, pp. 385–396.

[11] L. Orosa *et al.*, "A Deeper Look into RowHammer's Sensitivities: Experimental Analysis of Real DRAM Chips and Implications on Future Attacks and Defenses," *CoRR*, vol. abs/2110.10291, 2021.

[12] A.G. Yağlıkçı *et al.*, "Understanding RowHammer Under Reduced Wordline Voltage: An Experimental Study Using Real DRAM Devices," in *DSN*, 2022, pp. 475–487.

[13] S.W. Ryu *et al.*, "Overcoming the reliability limitation in the ultimately scaled DRAM using silicon migration technique by hydrogen annealing," in *IEDM*, 2017, pp. 21.6.1–21.6.4.

[14] K. Park *et al.*, "Experiments and root cause analysis for active-precharge hammering fault in ddr3 sdram under 3 × nm technology," *Microelectronics Reliability*, vol. 57, p. 39–46, 2016.

[15] S. Baeg *et al.*, "Estimation of the Trap Energy Characteristics of Row Hammer-Affected Cells in Gamma-Irradiated DDR4 DRAM," *IEEE Transactions on Nuclear Science*, vol. 69, pp. 558–566, 2022.

[16] A.J. Walker *et al.*, "On DRAM Rowhammer and the Physics of Insecurity," *IEEE Transactions on Electron Devices*, vol. 68, pp. 1400–1410, 2021.

[17] Digilent Inc., "Genesys 2," https://digilent.com/reference/programmable-logic/genesys-2/start, [Online; accessed 14-Dec-2023].

[18] C. Vonk, "Instruction set: Introduces a simplified LC-3 instruction set," https://coertvonk.com/inquiries/how-cpu-work/instruction-set-30971, [Online; accessed 14-December-2023].

[19] *51204 - MIG 7 Series DDR2/DDR3 - PHY Only Design Guide*, https://support.xilinx.com/s/article/51204?language=en_US, Xilinx, Inc., 2014.

[20] *Zynq-7000 SoC and 7 Series Devices Memory Interface Solutions v4.2*, https://docs.xilinx.com/v/u/en-US/ug586_7Series_MIS, Xilinx, Inc., 2015, v4.2.

[21] *Micron: 4Gb: x4, x8, x16 DDR3 SDRAM Features*, https://digilent.com/reference/programmable-logic/genesys-2/start, Micron Technology, Inc., 2009, rev. M 4/13 EN.