Flipping Bits Like a Pro: Precise Rowhammering on Embedded Devices

Anandpreet Kaur, Pravin Srivastav, and Bibhas Ghoshal

***Abstract*—In this article, we introduce *Flip-On-Chip*, the first end-to-end tool that thoroughly examines the vulnerability of embedded DRAM against rowhammer bit flips. Our tool, Flip-On-Chip, utilizes DRAM address mapping information to efficiently and deterministically perform a double-sided RowHammer test. We evaluated Flip-On-Chip on two DRAM modules: 1) LPDDR2 and 2) LPDDR4. It is found that our proposed tool increases the number of bit flips by *7.34%* on LPDDR2 and by *99.97%* on LPDDR4, as compared to state-of-the-art approaches provided in the literature. Additionally, Flip-On-Chip takes into account a number of system-level parameters to evaluate their influence on triggering Rowhammer bit flips.**

***Index Terms*—ARM, bit flips profiling, DRAM, Rowhammer.**

I. Introduction

SECURING memory from unauthorized modifications has been a major concern for computer systems. Researchers have put forth a number of strategies to guard memory against unauthorized modifications. However, the problem still persists and data corruption in memory is still possible. A primary reason is due to vulnerabilities which get exposed during operation and allow adversaries to mount sophisticated attacks that can bypass the memory protection techniques. For example, Mutlu and Kim [1] identified a new method of corrupting data kept in memory which the users do not have access to. Rowhammer is the name given to this phenomenon in which the same DRAM rows are purposefully accessed/struck repeatedly during a refresh interval in order to corrupt data by flipping bits in nearby cells. Technology scaling is the primary reason for this vulnerability, as it increases cell-to-cell interaction, thus reducing reliability. Though initially this bug was identified in DRAM of large computing systems, however, later, it was revealed that the memory of embedded and mobile devices is not resistant to this vulnerability either [2], [3].

Within a few years, a myriad of research studies have illustrated the devastating effects of this vulnerability by mounting attacks on various types of computing systems [1], [3], [4], [5], [6], [7]. These publications primarily offer two methods for mounting Rowhammer attacks: 1) probabilistic [5] and 2) deterministic [4], [7]. In probabilistic approaches, such as a single-sided Rowhammer attack, two or more rows are randomly chosen, and are accessed alternately causing bit flips in victim rows. Picking a random address is a simpler method since it does not require the knowledge of physical address mapping to DRAM's row, column, and bank bits [5], [8]. On the other hand, due to lack of mapping knowledge, applying these techniques does not guarantee the presence of target address at the vulnerable location of the DRAM [3]. Thus, they are unreliable, and may lead to alteration of undesired data. Deterministic methods are preferable because they allow strategic placement of critical data at vulnerable locations. This ensures deterministic bit flips, which are significantly more effective. Researchers have also shown that hammering two neighboring rows of a specific DRAM row (say $(i-1)$th and $(i+1)$th neighbor rows of an ith row) increases the likelihood of bit flips rather than hammering just one neighbor and a more distant row. Also, for many machines, double-sided hammering is necessary for triggering bit flips [5] in a reasonable time. In order to explore and launch an effective and deterministic Rowhammer attack on DRAMs, it is imperative to reveal the hidden DRAM address mapping.

The mapping information between user-level locations (such as virtual addresses) and physical locations on DRAM (such as row and bank), however, is not made available by the vendors. Therefore, access of the upper and lower neighboring rows in the same bank is not straightforward. Thus, the research community either resorts to hardware probing [4] or software-based methods [4], [6], [7], [8] to reveal this information. However, these techniques have mainly been restricted to the $\times 86$ platforms, and not much explored for embedded devices. To the best of our knowledge, only two software-based techniques have been proposed so far for embedded devices [9], [10]. Though the knowledge about bank bits was uncovered in [10], however, this article implemented the approach of DRAMA [4], and it is ineffective and does not give deterministic address mapping as indicated in [7]. Similarly, the proposal of [9], when executed, (source code was made publicly available[1]) it was found that only a single row bit location was identified, and the procedure did not reveal the bank bits because of which some of the collected group of three addresses were not situated in the same bank different row (SBDR). Moreover, since all row bits were not discovered, thus the number of aggressor and victim rows were left unidentified. As a result, the whole memory was not profiled and thus lot of vulnerable locations were remained

Manuscript received 14 July 2023; accepted 22 July 2023. This work was supported by GoI. This manuscript was recommended for publication by P. R. Panda. *(Corresponding author: Anandpreet Kaur.)*

The authors are with the Systems Laboratory, Indian Institute of Information Technology Allahabad, Prayagraj 211015, India (e-mail: phc2017001@iiita.ac.in).

Digital Object Identifier 10.1109/LES.2023.3298737

[1]https://github.com/0x5ec1ab/rowhammer armv8

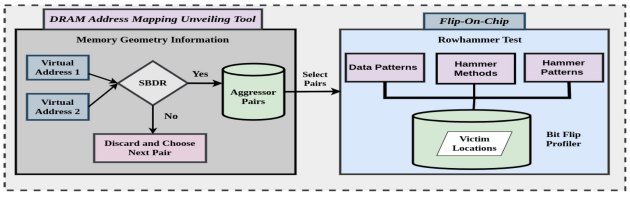

Fig. 1. Overall workflow.

uncovered. Thus, further research is needed to uncover the complete mapping information and verify this newly acquired knowledge by conducting rowhammer test on ARM devices.

Our Contributions: We present to the embedded research community.

1) Tools for DRAM address mapping and bit-flip profiling.
2) Analysis of bit-flip locations.
3) Examining the impact of various factors on the embedded DRAM chips.

To the best of the author's knowledge, there is no such tool that facilitates such an extensive study of RowHammer attacks on embedded devices, specially memory modules of ARM-based experimental boards.

II. Methodology

The main goal of our approach is to help a user to detect whether the DRAM of an embedded device is susceptible to bit flips and, if yes, analyze those vulnerable memory locations.

Fig. 1 illustrates the high-level view of our proposed methodology. It consists of two components-*DRAM address mapping unveiling* tool which infer the detailed memory geometry information [11] and *Flip-On-Chip* tool, which utilizes the provided information to perform the deterministic rowhammer test. Further, our Flip-On-Chip tool consists of two components: 1) a Rowhammer Test tool and 2) a bit-flip profiler (BFP). It also profiles the memory to determine data alterations that happened during the Rowhammer test and stores them. The detailed explanation is provided in the following sections.

A. DRAM Address Mapping Unveiling Tool

To trigger rowhammer in a deterministic manner, addresses that lie in the SBDR should be identified. To fulfil this precondition, we developed a tool that reverse-engineers DRAM chip of the target device and provides underlying hardware mapping information. We make use of the target device information to judiciously select physical addresses in order to provide deterministic DRAM address mapping. The device information includes *total number of banks, rows, columns, and physical memory size*. To find the hidden DRAM address mapping for ARM devices, two major technological challenges must be addressed.

1) *Determination of Device Information:* On any $\times$86 architecture, the device information can be obtained by reading the EEPROM kernel module. The kernel module is read via the *decode-dimms* system command [7], [8]. However, for ARM-based devices such as RPI 4B and RPI 3B+, no such EEPROM could be found and thus *decode-dimms* did not work. To find the device information, the hardware specification sheet of the device provided by the memory manufacturer was consulted.
2) *Development of Dedicated Module to Determine Row-Buffer and Nonrow-Buffer Conflicts:* To compare the timing between row-buffer and nonrow-buffer conflicts $\times$86 architecture supports an unprivileged instruction called *rdtsc* [4]. Unlike $\times$86, on ARM architecture, performance monitors cycle count register (PMCCNTR) instruction is privileged. Thus, to determine these conflicts for ARM, we had to implement our own kernel module.

Our address mapping tool consists of two major components: 1) address generation and translation unit and 2) a unit to map physical address bits to DRAM address bits. The *Main* Module, the *Communication* Module, and the *Latency* Module are the three modules which implement both the components. DRAM address mapping refers to mapping of physical address to a 3-tuple: $\langle$*Row, Column, Bank*$\rangle$ (DIMM, channel, and rank bits are included in the bank bits).

The mapping is constructed using the *timing channel* concept [6] which distinguishes *row-buffer conflicts* from non rowbuffer conflicts. The *row-buffer conflicts* happen when the two addresses are located in SBDR, resulting in higher latency as than the case that these addresses lie either within the same row or in different banks. Using this principle, the position of row, column, and bank bits is determined.

Finding Row and Column Bits We implemented the algorithm described in [6] that reverse-engineers mapping of $\times$86 platform. In contrast to their method, which is implemented as a user process, our approach consists of two modules: 1) the Main module, a user module and 2) the Latency module, a kernel module. We modify the following w.r.t. ARM architecture.

1) The total access time between addresses is measured using the *PMCCNTR* instruction, and it is privileged.
2) Unlike $\times$86, ARMv8 has different flush instruction, *DC CIVAC* flush instruction to clean and invalidate data cache so that next access can reach to DRAM and also memory barrier instructions *DSB 0XF, ISB* is used.

The Row bit (R) is a set of bits, that is determined, if address pairs with a one-bit difference have latency greater than the threshold. In similar manner, column bits (C) are detected, except that addresses that differ at two-bit positions (R and nonrow bit) are taken into account. Once the R and C bits have been revealed, the next step involves determination of bank bits. If a physical address has n bits, then $n-(R+C)$ remaining bits are regarded as "probable" bank bits, since they may contain some unrevealed column and row bits.

Identifying Bank Bits for Construction of Bank Address Functions (BAFs) To index DRAM banks, an exclusiveor logic functions on bank bits are computed and are called as BAFs. According to [12], ARM platform also employ XOR functions, which can be one-bit or two-bit functions. We made modifications to the approach in [7] specific to ARM, as their work identified BAF for $\times$86. Unlike their algorithm, that considers physical pages into account. The input to our algorithm is the physical address, as we discovered that our probable bank bits are present in both the offset and the physical page. Once BAF is revealed, the leftover bits are analyzed. By examining the

memory sheet of the target device, we were able to determine that the row and bank bits sets required to index banks and rows, respectively, are complete. As a result, leftover bits are part of an incomplete set and thus are assigned to the column bit set.

B. Proposed DRAM Profiler Tool: Flip-on-Chip

The factors our proposed tool takes into account, as well as information that BFP collects, is discussed next.

1) Rowhammer Test Tool: We assess the device vulnerability to bit flips by performing the rowhammer test. To perform deterministic rowhammer, accurate address mapping information is utilized. Given that current DRAM modules include millions of different rows and that the addresses are not chosen at random for hammering, thus, using the revealed information helps to narrow the search space. Additionally, it takes a reasonable amount of time to cause bit flips in victim rows. In addition to the DRAM address mapping, three other essential factors are taken into account by our tool: 1) the hammer methods; 2) the hammer patterns; and 3) the data patterns for effectively triggering bit flips [13]. To the best of our knowledge, there has not been an investigation that looks into the ways these important factors affect bit flips on DRAM modules of embedded devices.

a) Hammer methods: To make sure that a memory access is serviced from the DRAM, instead of cache, the following eight ARM-compatible instructions provided by [9] are examined.

1) Data Cache Clean by Virtual Address to point of coherency (PoC) {DC CVAC} with and without DSB.
2) Data Cache Clean and Invalidate by Virtual Address to PoC {DC CIVAC} with and without DSB.
3) Data Cache zero by virtual address (DC ZVA) instructions with and without DSB.

b) Data patterns: This parameter sets the data values for both the victim and the aggressor rows, and 8 such data patterns are taken into account [13].

c) Hammer patterns: This indicates the number of aggressor rows to be hammered, and our tool considers double-sided and many-sided patterns [13].

2) BFP: After hammering each aggressor rows, memory is scanned and a bit-flip table storing physical addresses of both aggressor pairs and victim locations is created. Since most of the victim locations are repeatable, it is quite likely that the same value of a cell can be corrupted in the future [1]. Thus, the information gathered in the bit-flip table becomes a valuable resource for an attacker, and it can be used to plant security-sensitive data at a target area and then exploit it. Also, the defence mechanisms can guard these vulnerable areas.

III. Evaluation

In this section, we present the Proof-of-Concept of both tools by evaluating them on two memory modules. We compare our proposed method with the technique described by [9] in terms of induced #bit-flips, and also examine the data gathered by BFP.

Experimental Setup: We validated our proposed approach on two LPDRAM modules: 1) LPDDR4 memory of RPI 4B device and 2) LPDDR2 memory of RPI 3B+ devices using a *Linux raspberry-pi 5.15.32-v8+ kernel*. In order to ensure that we do not overlook any addresses that are required to locate the mapping information, and also to discover all vulnerable memory locations, we *mmap* a large chunk of memory (70%–80%).

TABLE I
Reverse Engineered Dram Mapping

DRAM Type	Device Name	DRAM Configuration	Row Bits	Column Bits	Bank Address Functions
LPDDR4	RaspberryPi 4B	2,1,1,8	15-31	0-10	(11,12), (12), (13), (14)
LPDDR2	RaspberryPi 3B+	1,1,1,8	16-29	0-12	(13,14), (14), (15)

TABLE II
Average #Bit Flips Induced on LPDRAM Modules

DRAM	**LPDDR2**	**LPDDR4**
Round No.	***State-of-the-art* [9] / *Flip-On-Chip***	***State-of-the-art* [9] / *Flip-On-Chip***
R1	338088 / 370549	1455888 / 2910768
R2	339128 / 371664	1455528 /2911536
R3	349336 / 363936	1455232 / 2910768
R4	338936 / 369768	1455616 / 2910944
R5	345581 / 360757	1456112 / 2910704
Average	342213.8 / 367334.8	1455675.2 /2910944
% Increment	**7.340732606**	**99.97208168**

A. DRAM Address Mapping Unveiling Tool

To generate deterministic mapping, the data sheet of *RPI 4B* and *RPI 3B+* corresponding to memory part [14] and [15], respectively, were consulted to obtain the memory configuration. We successfully uncovered DRAM address mapping as shown in Table I for these two devices. A specific DRAM setup is shown in a quadruple by the DRAM Configuration column: (number of channels, # DIMMs per channel, # ranks per DIMM, and # banks per rank). Here, BAF (BAF0) for RPI 3B+ represents XOR-ed combination of two bank bits 13th and 14th.

B. Flip-on-Chip

Here, we assess our tool effectiveness in terms of induced #bit-flips.

1) Comparison With State-of-the-Art: Using the DRAM mapping data provided by our mapping tool and state-of-the-art work [9], respectively, we executed the Rowhammer test on both devices and found that our mapping results were as expected. We obtain a greater number of bit flips as compared to a number of bit flips induced by [9]. The double-sided rowhammer technique with the *DC ZVA* hammering approach was specifically used in this experiment. We executed five rounds of rowhammer tests for each setting, and results are shown in Table II. The data stored at both aggressor rows and victim rows was *0xffff/0xffff/0xffff* and is 64-bits wide. R1–R5 represent the five rounds, and from the average column it can be observed that our tool considerably induced more bit flips compared to state-of-the art. We noticed that not all the victims produced by the [9] code followed the ideal description of a victim, which is a bit-flip location that is situated in the same bank as the aggressor rows. We ran their code on RPI 3B+ and found a 25.40% error in the victim's location. Thus, these results justify the correctness of knowing full information of physical-DRAM address mapping and highlights the importance of bank bits information knowledge.

TABLE III
#BIT-FLIPS INDUCED ON LPDDR2 AND LPDDR4 MODULES

DRAM	Data Patterns	Best Hammer Methods (HM)		#Bit-Flips	
		2-sided	3-sided	2-sided	3-sided
LPDDR2	(0xff/0x00/0xff)	DC CIVAC+STR +DSB	DC CIVAC+STR+DSB	13304	17408
	(0x00/0xff/0x00)	DC CIVAC+STR+DSB, DC CVAC + STR + DSB, DC CIVAC + STR	DC CIVAC+LDR+DSB, DC CIVAC +STR +DSB, DC CVAC+STR+DSB	10240	13312
	(0x550/0x550/0x55)	DC CVAC+ STR+DSB	DC CVAC+STR	1128	1128
	(0xaa/0xaa/0xaa)	DC CVAC+STR+DSB	DC CIVAC + STR + DSB	1192	1128
	(0xaa/0x550/0xaa)	DC CVAC + STR +DSB	DC CVAC + STR +DSB	13312	16384
	(0x550/0xaa/0x55)	DC CIVAC+STR+DSB	DC CIVAC +STR +DSB	15360	17408
	(0x6d/0xb6/0xdb)	DC CVAC + STR + DSB	DC CIVAC + STR	12528	12536
	(0x92/0x49/0x24)	DC CIVAC+STR+DSB	DC CIVAC + STR + DSB	8192	12472
LPDDR4	(0x6d/0xb6/0xdb)	6 HM except DC ZVA & DC ZVA +DSB	6 HM except DC ZVA & DC ZVA +DSB	1002	1016
	(0x92/0x49/0x24)	DC CVAC + STR	6 HM except DC ZVA & DC ZVA +DSB	1008	1016

2) Analysis: A look at the literature related to bit flips on embedded devices, reveals that the research has been restricted to causing bit flips using rowhammer. No further analysis has been done on exploring either the location and pattern of victim addresses or exploring bit flips on various devices. However, such an analysis is of utmost importance since prior knowledge of vulnerable locations in a DRAM will enable attackers to mount RowHammer attacks in a more efficient and precise way. In this work, we tried to provide an analysis of the bit flips that were recorded using the Flip-on-Chip tool. The aim is to answer the following research questions.

1) *RQ1:* Do the victim addresses stays the same, if bit-flip procedure is carried out on a different experimental board?
2) *RQ2:* If bit flips occur in some victim locations of an embedded DRAM, do they occur consecutively in the DRAM, or they occur in some region?

To address these questions, four LPDDR2 modules of the RPI3B+ boards running *Flip-On-Chip* and state-of-the-art approach were used as experimental set-up. Both approaches were executed repeatedly on each of these devices, collecting the common victim memory locations in a bit-flip table. Results of the experiments were produced when there is little variation in the common number of bit flips between rounds. The results show that when our tool and the state-of-the-art approach were executed, they were able to determine almost 99% (%) Common Victims Across RPI's in three rounds and 70% in five rounds respectively.

Further, we found that these aggressor rows that corrupt victim locations are consecutive and the longest sequence length is 16. We found, 1175 such small clusters. Next, we tried to find how distant these found clusters are from each other. Is there any specific pattern they follow? We discovered that a total of 14 such clusters or zones are there and each of them are, evenly spaced from each other. The largest memory cluster included 6464 bit flip causing aggressor rows. Using this information, an adversary will have prior knowledge of the space they may utilise for placing security sensitive data.

3) Examining Key Parameters: Our Flip-On-Chip considers three primary configurable parameters, and the effects of those parameters on two DRAM modules are shown here. Table III shows maximum *#bit-flips* triggered on LPDDR2 and LPDDR4 DRAM in response to various data patterns, along with the hammer method that caused those bit flips. The data pattern *0x55/0xaa/0x55: checkered board* is most effective in triggering bit flips on LPDDR2. On LPDDR4 modules, only *killer pattern: 0x6d/0xb6/0xdb, 0x92/0x49/0x24* induced the flips among all data patterns.

IV. CONCLUSION

In this work, we examined, in great detail, how different parameters, such as knowledge of DRAM address mapping, hammer patterns, data patterns, and hammer methods affects bit flips on embedded devices. Our study demonstrates that having knowledge of underlying memory geometry information have a significant impact, as the attackers can cause significantly more bit flips by choosing target addresses. To support our work, we developed *DRAM Address Mapping Unveiling Tool* to reverse-engineer the DRAM address mappings for *ARMv8*-based devices. Further, we also tried to get analysis result on the likelihood of getting repeated bit flips on a memory module and found interesting patterns in victim locations using *Flip-On-Chip* tool.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their valuable feedback as well as ESWEEK-CASES team.

REFERENCES

[1] O. Mutlu and J. S. Kim, "RowHammer: A retrospective," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 8, pp. 1555–1571, Aug. 2020.
[2] A. P. Fournaris, L. P. Fraile, and O. Koufopavlou, "Exploiting hardware vulnerabilities to attack embedded system devices: A survey of potent microarchitectural attacks," *Electronics*, vol. 6, no. 3, p. 52, 2017.
[3] L. P. Fraile, A. P. Fournaris, and O. Koufopavlou, "Revisiting Rowhammer attacks in embedded systems," in *Proc. DTIS*, 2019, pp. 1–6.
[4] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "DRAMA: Exploiting DRAM addressing for Cross-CPU attacks," in *Proc. USENIX Security*, 2016, pp. 565–581.
[5] M. Seaborn and T. Dullien, "Exploiting the dram rowhammer bug to gain kernel privileges," presented at Black Hat, 2015.
[6] Y. Xiao, X. Zhang, Y. Zhang, and R. Teodorescu, "One bit flips, one cloud flops: Cross-VM row hammer attacks and privilege escalation," in *Proc. USENIX Security*, 2016, pp. 19–35.
[7] M. Wang, Z. Zhang, Y. Cheng, and S. Nepal, "DRAMDig: A knowledgeassisted tool to uncover dram address mapping," in *Proc. DAC*, 2020, pp. 1–6.
[8] M. Seaborn. "How physical addresses map to rows and banks in dram." May 2015. [Online]. Available: http://lackingrhoticity.blogspot.com/2015/05/how-physical-addresses-map-to-rows-and-banks.html
[9] Z. Zhang, Z. Zhan, D. Balasubramanian, X. Koutsoukos, and G. Karsai, "Triggering rowhammer hardware faults on ARM: A revisit," in *Proc. ASHES*, 2018, pp. 24–33.
[10] M. Bechtel and H. Yun, "Memory-aware denial-of-service attacks on shared cache in multicore real-time systems," *IEEE Trans. Comput.*, vol. 71, no. 9, pp. 2351–2357, Sep. 2022.
[11] A. Kaur, P. Srivastav, and B. Ghoshal, "Work-in-progress: Dram-MaUT: Dram address mapping unveiling tool for ARM devices," in *Proc. CASES*, 2022, pp. 41–42.
[12] M. Schwarz, "DRAMA: Exploiting DRAM buffers for fun and profit," M.S. thesis, Dept. Comput. Sci., Graz University of Technol., Graz, Austria, 2016.
[13] Z. Zhang et al., "BitMine: An end-to-end tool for detecting rowhammer vulnerability," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 5167–5181, 2021.
[14] "200b: x16/x32 LPDDR4/LPDDR4X SDRAM features," Data Sheet LPDDR4, Micron, Boise, ID, USA, 2017. [Online]. Available: https://datasheet.lcsc.com/szlcsc/2011180207Micron-Tech-MT53D1024M32D4DT-046-WT-D C907715.pdf/
[15] "Embedded LPDDR2 SDRAM features," Data Sheet LPDDR2, Micron, Boise, ID, USA, 2014. [Online]. Available: https://datasheetspdf.com/pdf-file/1202705/MicronTechnology/EDB8132B4PB-8D-F/1/