REFault: A Fault Injection Platform for Rowhammer Research on DDR5 Memory

Stefan Gloor[0009−0002−0555−2210], Patrick Jattke[0000−0003−2574−907X], and Kaveh Razavi[0000−0002−8588−7100]

Computer Security Group, ETH Zurich
{stgloor,pjattke,kaveh}@ethz.ch

Abstract. DDR5 is showing increased resistance to Rowhammer attacks compared to previous generations. The minimum hammer count (HC_{min}) is a metric to assess the susceptibility of the DRAM substrate to Rowhammer. Due to the lack of a generic platform that allows disabling refresh commands, there is currently no way to determine the HC_{min} of DDR5 UDIMMs. We address this gap with *REFault*, a low-cost fault injection system that allows altering DDR5 commands on-the-fly. REFault is made from a configurable DRAM fault injection interposer and a custom-designed injection controller. We leverage REFault to temporarily disable refresh commands on a commodity system, and determine, for the first time, the HC_{min} of two DDR5 devices from major DRAM manufacturers. We show that the HC_{min} is as low as 16 k activations which has not improved compared to DDR4 devices. We conclude that the increased resistance to Rowhammer in DDR5 devices comes from improved mitigations rather than the DRAM substrate itself.

Keywords: Rowhammer · DDR5 · DRAM · Fault Injection.

1 Introduction

Since the initial discovery of Rowhammer in 2012 on DDR3 devices [15], researchers have investigated the susceptibility of DRAM to Rowhammer. A substantial body of research has demonstrated that Rowhammer can be practically exploited in real-world scenarios and has detrimental implications on system security [4, 6–9, 22, 23, 25, 16]. In response, DRAM manufacturers have introduced in-DRAM mitigations with DDR4 devices. However, these measures were proven to be ineffective in protecting against Rowhammer [9, 8].

In 2024, the first DDR5 Rowhammer bit flips were reported [10] on a DDR5 DRAM device. In addition to enhanced Rowhammer mitigations, DDR5 incorporates on-die error-correcting codes (ECC) and a two-fold increase of the refresh rate (32 ms) compared to DDR4, both of which have implications for Rowhammer. Furthermore, advancements in the manufacturing process may also influence the inherent Rowhammer susceptibility. Consequently, it is yet to be shown how susceptible DDR5 devices are compared to their predecessors.

A common method to quantify the susceptibility of a DRAM device to Rowhammer is to measure the minimum number of activations required to induce the first bit flip, which is referred to as *minimum hammer count* (HC_{min}) [14]. As in-DRAM Rowhammer mitigations are typically assisted by refresh commands [6, 8], it is required to disable them while determining HC_{min}. However, we do not have the capability to disable automatic refreshes in the memory controller of off-the-shelf desktop CPUs. To make this possible, one approach that has been proven viable on DDR4 involves injecting faults into the DRAM command bus using an interposer such that the parity mismatch invalidates the targeted refresh commands [3]. As this approach requires manipulating high-speed signals, it is only further complicated by the increased clock frequency present on DDR5. Furthermore, two-cycle commands in DDR5 and the lack of the command parity signal that DDR4 fault injection [3] relied on, make designing a fault injection system for DDR5 substantially more challenging.

In this work, we introduce *REFault*, a low-cost fault injection system that enables altering bits of the DDR5 command bus. REFault consists of a configurable DDR5 interposer with both general-purpose and high-frequency solid-state switches with a careful design to minimize parasitic effects and to ensure reliable operation. Further, our REFault platform comprises a microcontroller mounted on a custom-designed fault injection controller and a software stack to automate long-running experiment campaigns. To identify a suitable bit line for fault injection, we conducted a thorough analysis of constraints and side effects when injecting faults in all possible DRAM commands that can result in suppression of refreshes. We demonstrate REFault's potential for studying DDR5 by suppressing refresh commands for multiple milliseconds to characterize the Rowhammer susceptibility (HC_{min}) of two DDR5 devices over 128 tested rows. We determined that the HC_{min} of both devices is as low as 16 k activations. We conclude that the increased resistance to Rowhammer in DDR5 devices as measured by recent work [10] comes from improved mitigations rather than the DRAM substrate itself.

Contributions. In summary, we make the following contributions:

- We design and build *REFault*, a low-cost fault injection system that allows to alter command bits of the high-speed DDR5 bus at the hardware level.
- We analyze the DDR5 command bus encoding as provided by JEDEC [12] to determine which bit manipulations are usable to disable refresh commands, and therefore, Rowhammer mitigations.
- We build an automated system that orchestrates the fault injection, monitors the progress to recover the system in case of crashing faults, and gathers experiment results.
- We demonstrate REFault's usability by, for the first time, determining the minimum hammer count (HC_{min}) of two DDR5 devices.

Open Sourcing. We publish[1] our hardware design files, software, and documentation to enable reproducibility of our results.

[1] https://github.com/comsec-group/refault

2 A Primer on DRAM

DRAM (dynamic random access memory) has evolved to become the *de facto* standard for today's computing devices, for example, in the form of dual in-line memory modules (DIMMs) for desktops and servers. In its core, DRAM is composed of a set of two-dimensional arrays of memory cells, where each cell stores a single bit of information in a capacitor.

Architecture. Like in previous generations, the architecture of DDR5 memory is structured hierarchically: individual memory arrays are called *banks*, which are grouped together to form *bank groups*. DRAM address bits encode bank (BA), bank group (BG), *row*, and *column* addresses. A single bank spans several physical chips, because the data bus width of the DIMM bus is wider (32 bits) than the one of individual DRAM chips (8 or 16 bits). Multiple chips can be grouped together to form a *rank*, where all chips share the same *Chip-Select* (CS) signal. This allows for increased memory capacity by having multiple ranks that are operated independently of each other. A *channel* is the set of all command, data, and control lines of a memory controller required to interface DRAM. Modern CPUs commonly offer multiple independent channels. DDR5 introduces the new concept of *subchannels*. Two entirely independent channels are now present on one DIMM, which has the benefit of increased concurrency and support for better scheduling by the memory controller, but also implies that the available 288 bus pins must now be shared between the two subchannels. As a consequence of this, there are fewer command/address (CA), data, and control lines available to a DDR5 subchannel than on regular channels in previous generations. Because of this reduced pin count, DDR5 introduced two-cycle commands, utilizing two subsequent clock cycles to encode a single command [11, 12, 21].

Command Set. DDR5 DRAM operates with a given set of commands (see Table 1), encoded by dedicated CA bits. These commands are used for calibration and initialization, as well as normal operation.

Refresh. To prevent data loss due to charge leakage, DRAM cells need to be refreshed periodically. Typically, the recommended refresh window to ensure proper operation of a DRAM cell is in the order of milliseconds (e.g., 32 ms for DDR5). However, past experiments [19] have shown that in practice, memory cells can hold their charge over the span of multiple seconds without being refreshed at room temperature [2]. DDR5 supports two modes of refresh operations: Normal Refresh mode and Fine Granularity Refresh (FGR) mode. In normal refresh mode, all bank refreshes (REFab) are issued to refresh all banks in each bank group. In FGR mode, additionally same-bank refreshes (REFsb) can be issued to only refresh a specific bank in each bank group.

Rowhammer Mitigations. DRAM manufacturers have incorporated a variety of on-die mitigations, commonly known as Target Row Refresh (TRR). In general, these mitigations attempt to identify potential victim rows and refresh them prematurely (i.e., before the next periodic refresh) to avoid that any disturbance errors can be induced. In DDR4, such mitigations have not prevented Rowham-

Table 1: Simplified DDR5 command set as defined by JEDEC [12]. V=valid (defined logic level, either high or low), CID=chip id (for die stacking), X=don't care (can be floating), AP=auto precharge, CW=control word, BL=burst length, WRP=write partial, ODT=on-die termination, RFU=reserved for future use, meaning that it may be assigned a new command in the future. The behavior of unassigned command encodings is undefined.

Command	**Abbr.**	**CS**	**Command/address bits**													
			0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Activate (Open a row)	ACT	L	L	L	Row R0–3				Bank		Bank group			Chip CID0–2		
		H	Row R4–16													R17/ CID3
RFU		L	H	L	L	L	V									
		H	V													
Write pattern	WRP	L	H	L	L	H	L	H	Bank		Bank group			Chip CID0–2		
		H	V	Column C3–10								V	AP	H	V	CID3
RFU		L	H	L	L	H	H	V								
		H	V													
Mode register write	MRW	L	H	L	H	L	L	Address MRA0–7								V
		H	Opcode OP0-7								V		CW	V		
Mode register read	MRR	L	H	L	H	L	H	Address MRA0–7								V
		H	V										CW	V		
Write	WR	L	H	L	H	H	L	BL	Bank		Bank group			Chip CID0–2		
		H	V	Column C3–10								V	AP	WRP	V	CID3
Read	RD	L	H	L	H	H	H	BL	Bank		Bank group			Chip CID0–2		
		H	V	Column C3–10								V	AP	V		CID3
Vref CA		L	H	H	L	L	L	Opcode OP0-6							L	V
Vref CS		L	H	H	L	L	L	Opcode OP0-6							H	V
Refresh all	REFab	L	H	H	L	L	H	CID3	V			H	L	Chip CID0–2		
Ref. mgmt. all b.	RFMab	L	H	H	L	L	H	CID3	V			L		Chip CID0–2		
Ref. same bank	REFsb	L	H	H	L	L	H	CID3	Bank		V	H		Chip CID0–2		
Ref. mgmt. s. b.	RFMsb	L	H	H	L	L	H	CID3	Bank		V	L	H	Chip CID0–2		
Precharge all	PREab	L	H	H	L	H	L	CID3	V				L	Chip CID0–2		
Precharge same b.	PREsb	L	H	H	L	H	L	CID3	Bank		V		H	Chip CID0–2		
Precharge	PREpb	L	H	H	L	H	H	CID3	Bank		Bank group			Chip CID0–2		
RFU		L	H	H	H	L	L	V								
Self-refresh entry	SREF	L	H	H	H	L	H	V					L	V		
Power-down entry	PDE	L	H	H	H	L	H	V					H	ODT	V	
Multi-purpose	MPC	L	H	H	H	H	L	Opcode OP0–7								V
Power-down exit, No operation		L	H	H	H	H	H	V								
Deselect	NOP	H	X													

mer attacks in the past [6, 8–10]. DDR5 introduced Refresh Management (RFM) commands to provide extra time for DRAM to manage TRR internally.

Previous work [3] demonstrated a cost-effective, hardware fault injection system for DDR4 memory that disconnects bus signals to manipulate DRAM commands. It consists of an interposer that sits between the motherboard's DIMM slot and a standard DDR4 DIMM. While more flexible FPGA-based tools exist (e.g., [1]), live fault injection has several advantages: (i) it does not rely on the implementation of a custom memory controller; (ii) it can be used in conjunction with any standard CPU, allowing the investigation of memory controller-dependent aspects; and finally, (iii) it is very inexpensive and easy to manufacture. With the advent of DDR5 memory, the question arises: *can a similar fault injection platform be designed for DDR5?* Since DDR5 introduces some novelties that complicate this simple fault injection approach, this is a non-trivial problem that requires solving three major challenges we explain next.

3 Challenges

DDR5 commands are encoded differently than DDR4 commands. Due to the introduction of subchannels, there are less CA pins available, which is likely why DDR5 lacks a parity signal for the CA bus. This means that all commands, even the ones corrupted by fault injection, will be interpreted by DRAM instead of being safely discarded. Also, this reduction in pin count per channel led to the introduction of two-cycle commands, which use two clock cycles to convey a single memory access or control instruction. With these types of commands, forcing a single CA line to a fixed level will inevitably affect two bits in the command encoding. This limits the number of possible CA candidates to fault and potentially increases the negative side effects of fault injection, as some corrupted commands may lead to protocol violations that induce undefined behavior.

Challenge 1. Determining *usable* faults that transform a targeted DRAM command into another valid one while minimizing negative side effects.

By comprehensively analyzing the command set and the fault injection implications as shown in Section 4, we minimize the negative side effects. In Section 7, we show that in practice, any remaining effects such as potential timing violations, do not inhibit the ability to perform Rowhammer.

Commands on the DDR5 bus are issued at a high rate with nanosecond precision. Electrically, this requires that DDR5 hardware must be carefully designed with regard to trace impedance, propagation delay, reflections, cross talk and other parasitic effects. In our application, we must not only passively carry the high speed signals, but actively switch them and change their logic level. Specifically, we need to first *disconnect* the appropriate CA line and then *force* the respective DIMM input to the desired logic level. This requires a solid state switch suitable for high speed signals which we can integrate in our design.

Challenge 2. Designing a PCB and electronic circuit capable of switching DDR5 signals while maintaining an acceptable level of signal integrity.

We tackled this challenge by paying close attention to high-speed electronics design principles and by improving the design iteratively over multiple versions. We describe the applied PCB design techniques as well as the chosen semiconductor switches in Section 5.

Unlike traditional Rowhammer experiments which operate the DRAM device within the specifications, hardware fault injection, by definition, stresses the hardware beyond its limits. It is unclear how DRAM and the experiment machine will behave to a condition violating the DDR5 specification. As resulting crashes might render the experiment machine unusable, the fault injection system should still be able to autonomously perform data recovery and rebooting of the machine, regardless of its condition and without manual intervention. This is especially important for long-term experiments, such as determining hammer counts of many rows.

Challenge 3. Automatically collecting experiment data and recovering the system to continue the experiment if fault injection renders the experiment machine unresponsive.

We solved this challenge by having a separate control server that orchestrates the experiment autonomously, as shown in Section 6. If the experiment machine becomes unresponsive, the control server has the ability to power cycle the machine remotely.

4 Interposer Design

The fault injection interposer is the system's main component. The interposer actively forces signals on the DDR5 bus to a high or low logic level in order to manipulate memory commands. It consists of a custom PCB that slots in between a standard UDIMM slot (mainboard) and a DDR5 UDIMM. We focus on UDIMMs (unbuffered DIMMs) in this project, but the design should be easily transferable to RDIMMs (registered DIMMs).

Command Manipulation. As DRAM commands can be issued with only a few nanoseconds in between them, it is not feasible to suppress a single command in time without a sophisticated and costly high-speed electronics design. Therefore, we focus on modifying entire *types* of commands (e.g., REFs) for a (comparatively) long time period.

In contrast to DDR4, DDR5 does not offer a parity signal for the CA bus anymore. This implies that all commands issued by the memory controller will be interpreted by the DIMM, even if they were malformed by fault injection. Consequently, direct command *suppression*, as in mFIT [3], is not possible with DDR5 anymore. Instead, we *transform* one command into another one by alter-

ing its CA bit encoding (see Table 1). This new command must be "safe", in the sense that it must not interfere with the continuation of normal operation.

Suppressing REF Commands. To select CA lines of interest, we focused on the suppression of REFab and REFsb commands, as we identified them to be the most useful for Rowhammer research [6, 8]. There are several possibilities to transform REFs, according to the command encoding (Table 1). Table 2 shows the implications for the most important commands when faulting individual CA lines. As it can be seen in Table 2, there are always some restrictions when performing fault injection on a single CA bit (indicated by "X"). Ideally, we would only want a single implication (i.e., "X") for a single fault. We considered the three most promising possibilities:

- **Forcing CA3 High:** This would transform all REFs to PREpb commands, but it would also set column bit 5 during RD/WR and row bits 1 and 7 during ACT. With the newly created PREpb command, a particular bank would be precharged. The specification, however, permits precharging a bank which has an empty row buffer (i.e., has not been activated before).
- **Forcing CA10 High or Low:** This would remap REFsb to REFab and vice versa. According to the specification, REFsb is only allowed in FGR mode. As we cannot check (or manipulate) the current refresh mode, it is unclear at which point of the experiment the specification would be violated and how DRAM will react to such a condition. Additionally, timing violations may occur as the minimum time between two consecutive REFab commands (tRFC) in normal refresh mode is larger than the minimum interval between REFab and REFsb or two consecutive REFsb commands in FGR mode. CA10 also controls auto-precharge for read/write and sets bank group bit 2.
- **Forcing CA4 Low:** This would remap all REFs to "Vref CA/CS" commands, which are used for calibration and may be illegal during normal operation. We do not know how the DRAM would react to this. Additionally, it would fix row bits 2 and 8 during ACT, column bit 6 during write, and not allow reading from memory while the fault injection is active. This fault would also map PREpb to either PREab or PREsb, depending on the bank group.

For our final Rowhammer evaluation, we forced CA3 high (Section 7.2 and Section 7.3), as we believe this to be the fault that causes the least disturbance on DRAM operation while still effectively disabling all refresh commands.

5 Interposer Implementation

The design of the fault injection interposer presents two major challenges. First, it must be able to dynamically force the desired CA lines to either a high or low voltage level without further disturbing the DIMM operation beyond the desired fault. Namely, it must maintain a tolerable level of signal integrity and therefore some high-speed electronics design practices apply. It must also be physically compatible with both the DIMM slot and the memory module itself.

Table 2: Overview of CA candidates for fault injection and their implications. A plain "X" in the table indicates that the respective command will be transformed into another one, while "X^x" represents implications to command bits which do not change the type of command. We highlighted the faults (gray rows) we considered to be potentially usable.

Fault		Command									
		ACT	WR	RD	**REFab**	**REFsb**	RFMab	RFMsb	PREab	PREsb	PREpb
CA0	H	X									
	L	X^a	X	X	**X**	**X**	X	X	X	X	X
CA1	H	X	X	X							
	L	X^a	X^b	X^b	**X**	**X**	X	X	X	X	X
CA2	H	X^a	X^b	X^b	**X**	**X**	X	X	X	X	X
	L	X^a	X	X							
CA3	H	X^a	X^b	X^b	**X**	**X**	X	X			
	L	X^a	X	X					X	X	X
CA4	H	X^a	X	X^b					X	X	
	L	X^a	X^b	X	**X**	**X**	X	X			X
CA9	H	$X^{a\,c}$	X^c	X^c			X	X			X^c
	L	$X^{a\,c}$	X^c	X^c	X	X					X^c
CA10	H	$X^{a\,c}$	$X^{c\,d}$	$X^{c\,d}$	**X**		X		X		X^c
	L	$X^{a\,c}$	$X^{c\,d}$	$X^{c\,d}$		**X**		X		X	X^c

[a] Opcode encoding intact, row bits are affected.
[b] Opcode encoding intact, column bits are affected.
[c] Opcode encoding intact, bank group bits are affected.
[d] Opcode encoding intact, auto-precharge bit is affected.

Switching Devices. To achieve the desired goal of injecting faults to transform DRAM commands (see Section 4), our switching circuit depicted in Figure 1 is comprised of two integrated semiconductor switches:

- a high-frequency switch (A, TMUX136 [24]) to disconnect the CA line targeted from the DIMM bus, and
- a general-purpose analog switch (B, PI5A3157 [5]) to pull the CA line to either a high or low voltage level.

This configuration was chosen mainly for availability reasons, as a solid state, 3-way (SP3T) switch suitable for high frequencies and DC is challenging to find. We chose different switching devices than previous work [3], as those would not allow for pulling a CA line high, which is an important requirement for us.

The high-frequency switch introduces a propagation delay (approx. 100 ps [24]) to the signal. This delay is an inherent property of the switched channel and is, to our knowledge, independent of the switch's current position. If we incorporated switching stages on only a few selected lines, we would expect the interposer to

not work properly because of the introduced propagation delays. For example, an (unimpeded) clock edge would reach the DIMM earlier than a CA signal that is routed through a switch, potentially leading to a corrupted command. Additionally, different impedance or capacitive load properties among switched and unswitched CA lines may reduce signal integrity. Therefore, we added switching cascades to *all CA lines* of subchannel A, including chip select (CS) and clock signals, to affect the bus uniformly.

PCB Design. We managed to fit everything on a 6-layer PCB without the need for buried/blind vias, which keeps the manufacturing costs relatively low. The control signals of the switches are routed to an onboard flat-flex cable connector, to which the injection controller can be connected. Due to size constraints of the connector, we decided to only connect the control signals of CA{0-8, 10} and CS to the injection controller. The remaining control signals are exposed via test pads that can be statically wired to achieve a constant pass-through behavior. Figure 2 shows our interposer PCB.

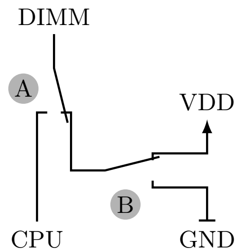
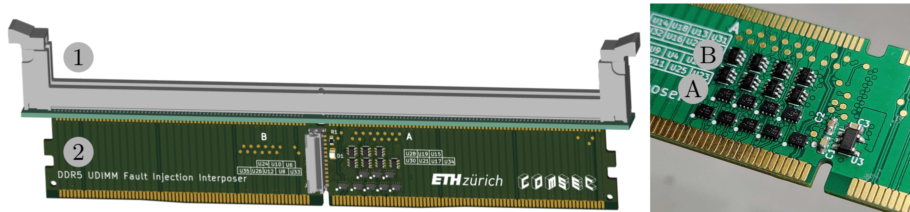


Fig. 1: Diagram of cascaded switches to pull a single CA line either high or low. This circuit is replicated for every relevant control/address signal of subchannel A to ensure equal propagation delays. For the clock signals, we omitted switch B, as there is no point in disconnecting and faulting this signal.



(a) 3D rendering.

(b) Close-up photo of the switching devices.

Fig. 2: Our fault injection interposer.

The PCB stackup, i.e., the specification of the used materials and their thicknesses for the individual conducting and dielectric layers of a PCB, affects critical electrical properties, such as impedance. For a high-speed application like this, using an impedance-controlled stackup, i.e., a PCB design with well-defined and known impedances, is crucial. Although JEDEC does provide a recommended

stackup [13], it is not manufacturable by our supplier. Instead, we use an available stackup and adjust the trace widths accordingly to match the impedance requirements. It is noteworthy that the specification defines multiple target impedances for differential signal pairs. Also, we carefully length-matched all relevant PCB traces to ensure an equal propagation delay.

Unlike DIMM slots on off-the-shelf mainboards, in which the DIMM sits perpendicular to the mainboard, we require a "straddle" mount connector (Figure 2a-(1)) for our interposer. This connector allows the UDIMM to sit in the same plane as the interposer PCB (Figure 2a-(2)), requiring less space and easier signal routing on the PCB. Sourcing DDR5 UDIMM straddle mount connectors in small quantities turned out to be a difficult endeavor at our usual distributors. We reached out to a manufacturer of DDR5 pass-through interposers that was willing to sell the connectors in small quantities.

We soldered the fault injection interposers in-house by using a custom stencil and a hot air gun. The straddle mount connector was soldered by hand. With the fault injection disabled (i.e., interposer is transparent), our systems equipped with this interposer successfully passed power-on self-test (POST) and runs of Memtest86+.

Costs. In Table 3, we provide an overview of the total hardware costs for the interposer and the injection controller. These costs are based on the discounted price for acquiring components for ten fault injections systems. We note that these costs do not include the different design revisions we built before arriving at the final design, the manual labor of assembling the PCBs, and the costs of other hardware used in our setup (e.g., networking gear, compute nodes).

Table 3: Overview of the total costs for building our fault injection interposer and injection controller.

	Item	**Supplier**	**Price [$]**
Interposer	PCB with stencil	JLCPCB	20
	Parts, e.g., switches	Mouser/Digikey	30
	Misc., cables, consumables		5
Injection Controller	PCB	JLCPCB	4
	Parts, e.g., connectors	Mouser/Digikey	40
	Teensy 4.1 μ-controller	PJRC	40
	Misc., cables, consumables		5

6 System Architecture

In this section, we explain our system architecture and the components surrounding the fault injection interposer. Figure 3 gives an overview of our fault injection system. At the heart of the system is our fault injection interposer (1)

which manipulates the bus signals. The interposer's switches are controlled by the injection controller (2), which in turn is triggered by the host software (3) running on the experiment machine. A control server (4) orchestrates the whole experiment by providing the host software and retrieving the experiment data.

We focus on the design of the injection controller in Section 6.1, the host software in Section 6.2, and the control server in Section 6.3.

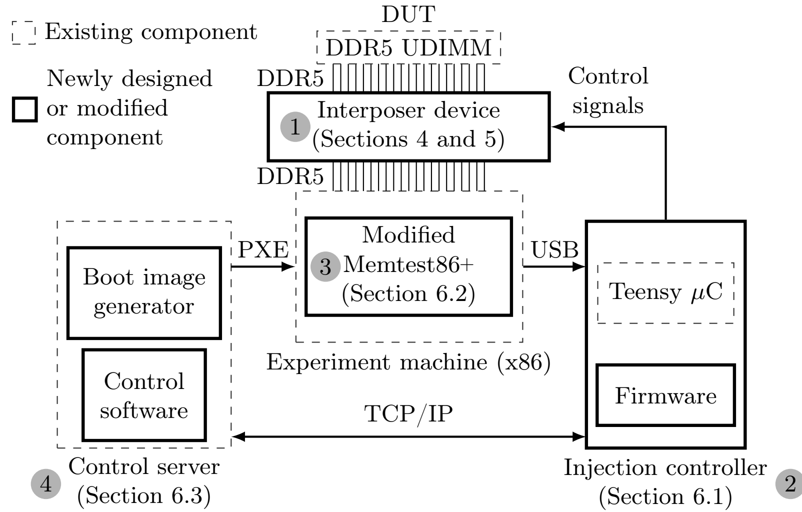


Fig. 3: Overview of our fault injection system, also indicating which components were designed and implemented during this work.

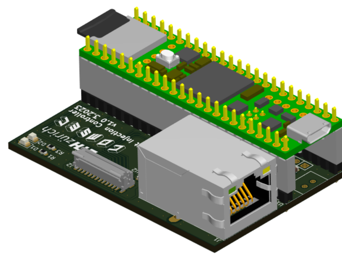
6.1 Injection Controller

As the fault injection interposer is restricted in size, we decided to place its driver logic on a separate PCB, the injection controller. To synchronize the fault injection with the workload, the experiment machine running the Rowhammer code must be able to communicate with this injection controller. For this, we decided to use USB, as it is universally supported and is comparably easy to implement. Although there might be other interface types that are easier to implement, the fact that modern platforms do not offer native serial (RS-232) or parallel ports anymore, renders this possibility infeasible[2].

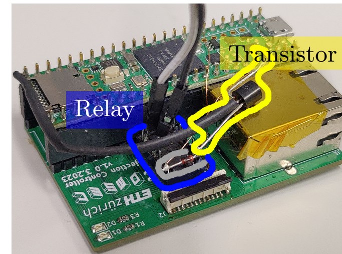
The injection controller, depicted in Figure 4a, consists of a custom baseboard and a Teensy 4.1 microcontroller board [20]. This microcontroller board is a commercially available, low-cost prototyping module with a powerful 600 MHz, 32-bit

[2] Using an adapter, which uses USB in many cases, would not be useful as it still requires suitable USB drivers.

ARM Cortex M7 CPU. Our custom baseboard provides the necessary connector for it to attach to the fault injection interposer using a flat-flex cable (FFC) which carries the control signals for the individual switches. The switches are directly driven by the 3.3 V GPIO pins of the Teensy microcontroller. Additionally, it incorporates an Ethernet port to connect to a network, as well as some status LEDs. As an afterthought, we added a switching relay to the injection controller board to trigger the physical power switch of the experiment machine (Figure 4b). This allows for remote power cycling even if the system is completely unresponsive. This mechanical relay is driven by a transistor that directly connects to a GPIO pin of the Teensy microcontroller board.



(a) 3D rendering.



(b) Manually added transistor and relay (flyback diode highlighted in gray).

Fig. 4: Injection controller.

Firmware. The firmware of our injection controller is kept simple and has the following three main tasks:

- Listen for USB packets from the experiment machine and either store the received data in a general-purpose data buffer or initiate the fault injection.
- When a fault injection trigger packet is received over USB, activate the corresponding GPIO pins to drive the switches on the interposer.
- Provide a HTTP server as a mean to communicate with the control server and to retrieve the experiment data.

We implemented the firmware in C, using existing TCP/IP, USB, and multithreading libraries from both the Arduino and Teensy framework.

6.2 Host Software

Software running on the experiment machine should ideally only perform memory accesses when allowed by the experiment code, to not interfere with the fault injection and to keep noise to a minimum. This categorically excludes standard operating systems like Linux or Windows. One possibility is to develop a specialized, native UEFI application from scratch, which requires additional work

to implement the necessary startup and initialization code, USB host controller driver, and display drivers before the actual fault injection can be implemented.

Instead, we decided to adapt Memtest86+[3][26] for this purpose, as done by previous work [15]. Memtest86+ is an open-source DRAM testing utility running directly on top of UEFI, without any operating system in between. More importantly, we decided to use Memtest86+ as a base system because we can reuse its framework to build our system, as it already implements the display, USB, and SMBus/SPD bus drivers, as well as memory management.

Our host software must be able to send commands to the injection controller, access (or hammer) memory, check the memory for bit flips, and report the results back to the injection controller. For this, it needs to be able to allocate a sufficiently large contiguous piece of physical memory and communicate with the injection controller via USB. We patched Memtest86+ to include our Rowhammer experiment code as well as support for sending data (e.g., bitflip data) and commands (e.g., fault trigger) to the injection controller via USB. For this, we leveraged the existing USB keyboard driver framework in Memtest86+. We noticed that the unmodified USB stack of Memtest86+ has issues detecting keyboards on some USB ports of some systems, which also translates to our fault injection system. This issue seems to be unrelated to the presence of internal USB hubs, as external USB hubs do not pose a problem (on QEMU and on physical machines). We also leveraged the existing SPD driver of Memtest86+ to read out metadata of the DIMM under test. This data is sent to the injection controller such that it can be retrieved by the control server via its HTTP server. By using the existing display driver, status information can be displayed on an attached monitor.

6.3 Control Server

As fault injection might render the experiment machine unresponsive because of corrupted memory accesses due to the fault injection side effects (see Table 2), Rowhammer bit flips, or any other undefined behavior caused by violating the specification, a control server is needed to supervise and control the experiment. Figure 5 shows the control server components and their interactions.

The control server compiles (1) a bootable image for the experiment machine, containing the necessary framework and experiment code. It also hosts a PXE server to send (2) that image to the experiment machine. While the experiment is running, the control server periodically polls (3) the injection controller for collecting data, or in case of a crash, rebooting the machine. Both connections to the injection controller and experiment machine are running over TCP/IP on a dedicated, isolated network. With this setup, a researcher performing experiments with our system only needs (remote) access to the control server.

The control server consists of a standard GNU/Linux distribution, a DHCP server with PXE capabilities and a built-in TFTP server, as well as shell scripts for the automation logic. Depending on the experiment, customized Memtest86+

[3] Not to be confused with the proprietary Memtest86.

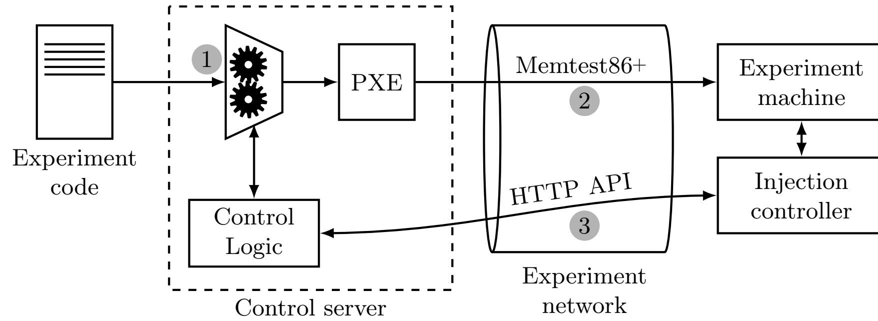


Fig. 5: Components of the control server.

images (e.g., with different hammering patterns) are compiled and autonomously booted on the experiment machine via PXE. Once the experiment finishes, the control server fetches the data from the injection controller. If the experiment fails or times out, the control server will automatically power-cycle the experiment machine and try again, or continue with the next experiment. We designed the system in such a way that the experiment can be run fully remotely and autonomously, with minimum user interaction: a researcher only needs remote access to the control server to start the procedure and collect the log files at the end. The control server can also oversee multiple injection controllers at once.

7 Evaluation

We first focus on basic functional validation (Section 7.1), followed by demonstrating a simple Rowhammer experiment (Section 7.2). We then provide a more elaborate hammer count estimation (Section 7.3), and finally, explain the issue of occasional data corruption that we encountered (Section 7.4).

7.1 Functional Validation

To test whether all switches are working, we ran a script on the control server that consecutively generates Memtest86+ images that inject a fault on each CA line with each fault level. Without proper control and synchronization of the workload, a crash of the system upon fault injection is very likely. In case the machine does not crash immediately, the fault injection is repeated in a loop. A crash usually manifests itself as a corrupted screen image, an exception (e.g., illegal instruction, page fault), or unresponsiveness. We use these crashes as a proxy to verify that all switches are properly working. In addition to this, we manually verified proper voltage levels and timings of single switches using a high-speed oscilloscope.

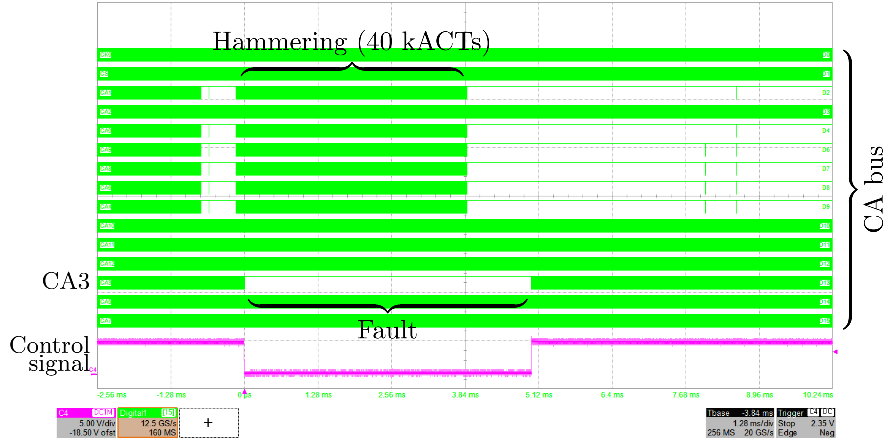


Fig. 6: Mixed-signal oscilloscope measurement of the DDR5 bus during a basic Rowhammer experiment. Due to the high data rate, the transitions of the digital DDR5 signals blend together at the depicted zoom level, such that they are displayed as solid bars.

7.2 Basic Rowhammer Experiment

To validate the system's basic functionality, we set up a simple Rowhammer experiment with disabled refreshes where we hammer a double-sided aggressor pair for 20 k activations (i.e., 40 k activations in total). For disabling refreshes, we force CA3 to be high during hammering, which we identified before as candidate for suppressing REFs in Section 4. This transforms all REF commands into PREpb. We monitored the DDR5 bus using a high-speed, mixed signal oscilloscope, as depicted in Figure 6. Using this experiment, we successfully triggered Rowhammer bit flips on DDR5. Bit errors happened only next to the aggressor rows and did not occur without hammering, which confirms that they are not a side effect of the fault or the absence of refreshes (i.e., retention failures).

Row Remapping. When forcing CA3 to a high level, this also forces row bit 1 to be set during activation. In general, this makes double-sided hammering impossible. However, we can circumvent this limitation by exploiting internal row remapping, which is employed by two out of three major DRAM manufacturers [17, 18]. By hammering rows 7 and 15 (as seen by the memory controller), a double-sided hammering of row 8 is achieved as row 15 is remapped to row 9 in these devices, as shown in Figure 7. As expected, we start observing bit flips at relatively low hammer counts when hammering rows 7 and 15, confirming that our devices actually employ this row remapping scheme.

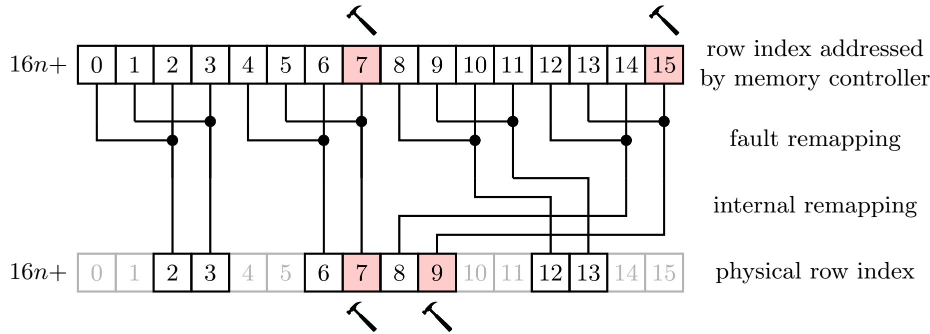


Fig. 7: In general, the row remapping caused by the fault makes double-sided hammering impossible. However, on devices with internal row remapping, this limitation can be circumvented.

7.3 Hammer Count Estimation

To demonstrate the usability of our platform, we estimated the minimal hammer count (HC_{min}) of devices from the two major DRAM vendors that employ internal row remapping. The details of the DIMMs we used are shown in Table 4.

Table 4: Details of DIMMs used for the hammer count estimation. We report the frequency (Freq.) in MHz, the data bus width (I/O) in bits, and further the number of ranks (R), bank groups (BG), and banks per bank group (B).

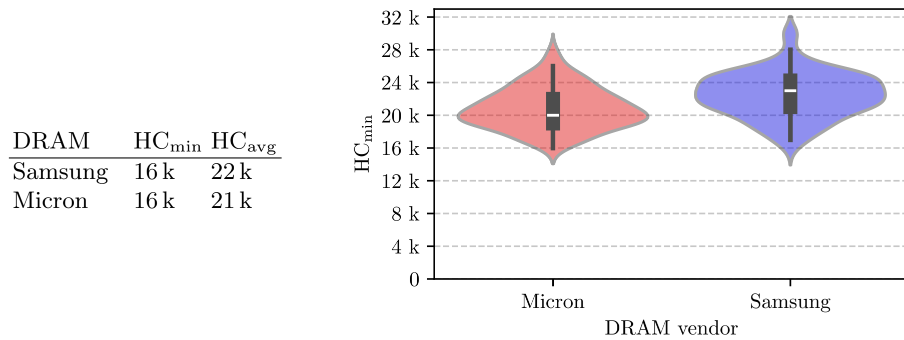
DRAM Vendor	DIMM Model	Mfg. Date	Size	Freq.	I/O	R	BG	B
Micron	CT16G48C40U5.M8A1	W02/22	16 G	4800	x8	1	8	4
Samsung	CMK32GX5M2B5200Z40	W38/22	16 G	4800	x8	1	8	4

Experiment. For this, we repeated the previous Rowhammer experiment (Section 7.2) using the CA3 fault with a sweeping number of activations over a span of 128 rows. We count activation of both aggressors as a single hammer. As we noticed during the previous experiment that the initial hammer count was not always sufficient to trigger bit flips, we start the experiment with a hammer count of 40 k (i.e., 80 k activations in total) and decrease it if a row yields bit flips. If a row does not yield bit flips three consecutive times, we continue with the next victim row, 16 rows further due to the row remapping (Figure 7). We decrease the hammer count by 1 k, 2 k, 3 k, or 5 k, depending on the previous hammer count and the number of bit flips we encountered in the previous round. Our final hammer count update, however, is always $\pm$ 1 k from the previously tested hammer count, which guarantees consistent accuracy of our reported HC_{min}. We found this optimization technique empirically, and we use it to speed up the

experiment run time. We did not employ a standard binary search, as a negative outcome (i.e., no bit flips) is more expensive (3 retries) than a successful one. Hence, we opted for a search technique that does not overshoot the true value.

Setup. The experiment was performed on a AMD Zen 4 machine (Ryzen 7 7700X) using a fault duration of 10 ms. Collecting data for 128 rows of a single DIMM required approximately 54 hours of runtime.

Results. The results are given in Figure 8 and its associated table. Our results show a minimum hammer count of 16 k activations, for both devices, with DRAM chips from Samsung and Micron. Single-sided bit flips on devices without internal row remapping (i.e., from the third major DRAM vendor) could not be obtained. It is unclear whether this is due to an error in our setup, or just simply because the real single-sided hammer count is higher than the maximal value we tested for (90 k), due to on-die ECC, which is available on DDR5 [12].



DRAM	HC_{min}	HC_{avg}
Samsung	16 k	22 k
Micron	16 k	21 k

Fig. 8: Minimal hammer counts for tested DIMMs. We report for both DIMMs the minimum (HC_{min}) and the average hammer count (HC_{avg}) over all 128 tested rows. The plot depicts the hammer count distribution across rows.

7.4 Reliability and Data Corruption

We observed a pronounced difference in reliability of the fault injection depending on the machine type (Intel Alder/Raptor Lake, AMD Zen 4), fault duration, and hardware configuration. We achieved the most reliable results on AMD Zen 4 using an external GPU and a fault duration of 10 ms. In this configuration, we observed a success rate (i.e., a successful hammering without the machine crashing) during the experiment in Section 7.3 of 33–41%.

Faults active for a prolonged duration (in the range of tens of milliseconds), drastically increase the probability of the machine crashing during the experiment, especially on Intel machines. In some of these cases, *all* data returned

by the DIMM appeared to be random. As it appears to occur suddenly (with respect to increasing the fault duration) and uniformly over time, we can rule out retention errors. Interestingly, the corruption of a 64 bit double-word seems to occur byte-wise, i.e., bytes are either corrupted or intact, which suggests that this might be an effect on a chip level. However, x16 chips still show a byte-grouped corruption, which may be either a consequence of the chip's internal architecture, or it may not be related to the chips at all, but rather an effect of other hard- or software. We leave further investigation of this problem to future work.

8 Discussion

There are some inherent limitations with the way we interpreted DDR5 fault injection that could be a promising direction for future work.

Command Transformation. Since single-command, or even single-bit manipulation would require complex high-speed circuitry capable of monitoring the DDR5 bus for nanosecond-precision fault injection, we must accept that the fault introduced by our interposer (which does not have any bus monitoring capabilities) will be present for the duration of multiple commands. This is due to both the coarse synchronization caused by the lack of bus monitoring and the fact that the solid-state switches are not fast enough to force only a single bit in time. The command encoding (see Table 1), defined by the JEDEC specification [12], severely limits the possibilities of manipulating a single CA line for a comparatively long period of time, since a single command bit always affects a multitude of different commands (see Table 2). The introduction of two-cycle commands further limits the possibilities of fault injection, as a CA line represents two bits of information during a single command. Hence, it is not possible to suppress or transform a single command of choice without affecting other commands. The challenge is to find a trade-off between suppressing the command of interest while still allowing basic memory access. We have focused on faulting CA3 bit high, as we believe this fault inflicts the least disturbance on DRAM operation while still achieving the goal of disabling REFs.

Data Corruption. As we described in Section 7.4, prolonged fault durations (larger than approximately 10 ms) drastically increase the probability of a crash of the experiment machine. This limits the flexibility of our platform, as some experiments, such as investigating retention failures, can not be performed due to requiring longer fault durations. As we were unable to locate the root cause of this problem yet, we can not make any speculation on how to circumvent this issue. It might be worthwhile to investigate whether the data corruption issue is related to certain DRAM commands being suppressed (e.g., multi-purpose commands), or due to some unknown side effect of the fault injection (e.g., interference). We think that further investigation is very promising to greatly increase the applicability of our fault injection platform.

Implications for Double-Sided Hammering. When suppressing REFs by faulting CA3 high, this also sets row bit 1 and 7. This means that a double-sided attack is generally not possible in this mode, as this would require access to two rows which are both next to the victim row and therefore differ in row address bit 1. However, because some DIMMs use internal remapping schemes, we can circumvent this limitation on selected DIMMs.

Other Faults. Exploring the usability of other faults, such as forcing CA10 high or low, or forcing CA4 low (Section 4), could be an interesting direction for future work.

9 Conclusion

We designed and built REFault, a custom fault injection system for flexible manipulation of the command/address (CA) bus in DDR5 memory systems. By analyzing the DDR5 command encoding [12], we identified the most suitable CA lines for fault injection. The REFault interposer, positioned between a standard computer mainboard and a DDR5 UDIMM, uses high-frequency switches to precisely impose a high or low level on any CA bit. These switches are controlled by a custom injection controller, triggered by a modified version of Memtest86+ running on the experiment machine. Our workflow integrates a control server to automate and monitor experiments, ensuring efficiency and reliability.

We confirmed the functionality of our platform by suppressing refresh commands for several milliseconds and successfully triggered DDR5 Rowhammer bit flips. For the first time, we determined the minimum hammer count (HC_{min}) of DDR5 DRAM devices from two major DRAM manufacturers. With this work, we have built a novel DDR5 testing platform that lays the foundation for future DDR5 Rowhammer research.

References

1. Antmicro: Extending the Open Source Rowhammer Testing Framework to DDR5. https://antmicro.com/blog/2022/08/extending-the-open-source-rowhammer-testing-framework-to-ddr5/, last accessed 2025/01/09 (2022)
2. Cojocar, L., Kim, J., Patel, M., Tsai, L., Saroiu, S., Wolman, A., Mutlu, O.: Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers. In: S&P '20. pp. 712–728. IEEE (2020). https://doi.org/10.1109/SP40000.2020.00085
3. Cojocar, L., Loughlin, K., Saroiu, S., Kasikci, B., Wolman, A.: mFIT: A Bump-in-the-Wire Tool for Plug-and-Play Analysis of Rowhammer Susceptibility Factors. No. MSR-TR-2021-25, Microsoft (2021), https://www.microsoft.com/en-us/research/publication/mfit-a-bump-in-the-wire-tool-for-plug-and-play-analysis-of-rowhammer-susceptibility-factors/, last accessed 2025/01/09
4. de Ridder, F., Frigo, P., Vannacci, E., Bos, H., Giuffrida, C., Razavi, K.: SMASH: Synchronized Many-sided Rowhammer Attacks From JavaScript. In: USENIX Security '21. USENIX Association (2021)
5. Diodes Inc.: PI5A3157. https://www.diodes.com/assets/Datasheets/PI5A3157.pdf, last accessed 2025/01/13 (2023)

6. Frigo, P., Vannacci, E., Hassan, H., van der Veen, V., Mutlu, O., Giuffrida, C., Bos, H., Razavi, K.: TRRespass: Exploiting the Many Sides of Target Row Refresh. In: S&P '20. IEEE (2020)
7. Gruss, D., Maurice, C., Mangard, S.: Rowhammer.js: A remote software-induced fault attack in JavaScript. In: DIMVA '16. pp. 300–321. Springer (2016). https://doi.org/10.1007/978-3-319-40667-1_15
8. Hassan, H., Tugrul, Y., Kim, J., van der Veen, V., Razavi, K., Mutlu, O.: Uncovering in-DRAM RowHammer protection mechanisms:a new methodology, custom RowHammer patterns, and implications. In: MICRO '21. ACM (2021)
9. Jattke, P., Van Der Veen, V., Frigo, P., Gunter, S., Razavi, K.: BLACKSMITH: Scalable rowhammering in the frequency domain. In: S&P '22. pp. 716–734. IEEE, United States (2022). https://doi.org/10.1109/SP46214.2022.9833772
10. Jattke, P., Wipfli, M., Solt, F., Marazzi, M., Bölcskei, M., Razavi, K.: Zenhammer: Rowhammer Attacks on AMD Zen-based Platforms. In: USENIX Security '24. USENIX Association (2024)
11. JEDEC Solid State Technology Association: JEDEC Standard DDR4 SDRAM. https://www.jedec.org/standards-documents/docs/jesd79-4a, last accessed 2025/01/09 (2014)
12. JEDEC Solid State Technology Association: JEDEC Standard DDR5 SDRAM. https://www.jedec.org/standards-documents/docs/jesd79-5b, last accessed 2025/01/09 (2020)
13. JEDEC Solid State Technology Association: DDR5 Unbuffered Dual Inline Memory Module (UDIMM) Common Standard. https://www.jedec.org/standards-documents/docs/jesd308a, last accessed 2025/01/13 (2024)
14. Kim, J.S., Patel, M., Yaglikci, A.G., Hassan, H., Azizi, R., Orosa, L., Mutlu, O.: Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques. In: ISCA '20. pp. 638–651. IEEE, Valencia, Spain (May 2020). https://doi.org/10.1109/ISCA45697.2020.00059, https://ieeexplore.ieee.org/document/9138944/
15. Kim, Y., Daly, R., Kim, J., Fallin, C., Lee, J., Lee, D., Wilkerson, C., Lai, K., Mutlu, O.: Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In: ISCA '14. pp. 361–372. ACM/IEEE (2014). https://doi.org/10.1109/ISCA.2014.6853210
16. Marazzi, M., Razavi, K.: RISC-H: Rowhammer Attacks on RISC-V. In: DRAM-Sec '24 (2024)
17. Nam, H., Baek, S., Wi, M., Kim, M.J., Park, J., Song, C., Kim, N.S., Ahn, J.H.: DRAMScope: Uncovering DRAM Microarchitecture and Characteristics by Issuing Memory Commands. In: ISCA '24. ACM/IEEE (2024)
18. Orosa, L., Rührmair, U., Giray Yağlikçi, A., Luo, H., Olgun, A., Jattke, P., Patel, M., Kim, J.S., Razavi, K., Mutlu, O.: SpyHammer: Understanding and Exploiting RowHammer Under Fine-Grained Temperature Variations. IEEE access **12**, 80986–81003 (2024). https://doi.org/10.1109/ACCESS.2024.3409389
19. Patel, M., Kim, J.S., Shahroodi, T., Hassan, H., Mutlu, O.: Bit-Exact ECC Recovery (BEER): Determining DRAM On-Die ECC Functions by Exploiting DRAM Data Retention Characteristics. In: MICRO '20. pp. 282–297. IEEE, Athens, Greece (2020). https://doi.org/10.1109/MICRO50266.2020.00034
20. PJRC, LLC.: Teensy® 4.1 development board. https://www.pjrc.com/store/teensy41.html, last accessed 2025/01/14 (2021)
21. Schlachter, S., Drake, B.: Introducing micron® DDR5 SDRAM: More than a generational update. https://www.micron.com/content/dam/micron/

global/public/products/white-paper/ddr5-more-than-a-generational-update-wp.pdf, last accessed 2025/01/09 (2019)
22. Seaborn, M., Dullien, T.: Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges. https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html, last accessed 2025/01/09 (2015)
23. Tatar A., Konoth, R., Athanasopoulos, E., Giuffrida, C., Bos, H., Razavi, K.: Throwhammer: Rowhammer Attacks Over the Network and Defenses. In: USENIX ATC '18. pp. 213–226. USENIX Association, Boston, MA (2018)
24. Texas Instruments: TMUX136. https://www.ti.com/lit/ds/symlink/tmux136.pdf, last accessed 2025/01/13 (2023)
25. Van Der Veen, V., Fratantonio, Y., Lindorfer, M., Gruss, D., Maurice, C., Vigna, G., Bos, H., Razavi, K., Giuffrida, C.: Drammer: Deterministic Rowhammer attacks on mobile platforms. In: CCS '16. vol. 24-28-October-2016, pp. 1675–1689. ACM (2016). https://doi.org/10.1145/2976749.2978406
26. Whitaker, M.: Memtest86+. https://memtest.org/, last accessed 2025/01/10 (2024)