Towards Trustworthy Neuromorphic Computing: An Analysis of Hardware Security and Reliability Risks

Von der Fakultät für Elektrotechnik und Informationstechnik
der Rheinisch–Westfälischen Technischen Hochschule Aachen
zur Erlangung des akademischen Grades
eines Doktors der Ingenieurwissenschaften
genehmigte Dissertation

vorgelegt von
M.Sc.(CVUT) M.Sc.(RWTH) Felix Staudigl
aus Ingolstadt, Deutschland

Berichter: Universitätsprofessor Dr. rer. nat. Rainer Leupers
Universitätsprofessor Dr.-Ing. Miloš Krstić

Tag der mündlichen Prüfung: 20.02.2025

Abstract

The von Neumann bottleneck poses a significant barrier, limiting the computing performance and energy efficiency of conventional computing systems. Neuromorphic computing seeks to overcome this bottleneck by leveraging the intrinsic parallelism and efficiency of the brain-inspired Computing-in-Memory (CIM) paradigm. Nonetheless, the immature state of the foundational building block, memristive devices, introduces substantial challenges in terms of their reliability and vulnerability to hardware security threats, necessitating thorough analysis to ensure the development of secure and reliable computing systems for future applications.

To provide an in-depth investigation of the reliability concerns, a fault injection platform is introduced evaluating the robustness of digital CIM operations. This platform marks a significant contribution to understanding and mitigating reliability issues in memristor-based systems by providing a holistic simulation framework operating on the crossbar and operational level.

Moreover, the work presents NeuroHammer, a novel hardware security attack exploiting the unique properties of memristive crossbar arrays to compromise the integrity of neuromorphic computing systems. This attack underscores the susceptibility of Resistive Random-Access Memories (ReRAMs) to hardware security threats, highlighting the critical need for effective countermeasures.

To perform a detailed reliability evaluation on actual devices, this dissertation unveils the NeuroBreakoutBoard (NBB)—a highly versatile instrumentation platform designed to examine the effects of memristive device nonidealities across various abstraction levels. The NBB's ability in executing CIM operations on real memristive crossbar arrays sets it apart from previously proposed platforms, emphasizing its essential role in facilitating thorough analyses for the advancement of dependable neuromorphic computing platforms.

In conclusion, this thesis delivers a comprehensive evaluation of memristive devices, spanning from reliability issues to hardware security threats, thereby paving the way for their broad adoption and integration into the next era of computing systems.

Acknowledgements

Completing this dissertation has been an extraordinary journey for both my academic and personal growth which would have not been possible without the invaluable support and encouragement I have received along the way. Therefore, I would like to express my deepest gratitude to several key individuals and groups whose contributions led to this dissertation.

First and foremost, I would like to express my sincerest thanks to Prof. Rainer Leupers for his support, guidance, and mentorship throughout this process. Through his unwavering trust, I was able to found the NeuroLab which strengthened not only my personal research but also benefits the overall neuromorphic research group at Institute for Communication Technologies and Embedded Systems (ICE).

Next, I am deeply grateful to have worked together with both scientific and non-scientific colleagues at the ICE. Their camaraderie, support, and assistance have not only enriched my research experience but also made my time at the institute truly enjoyable and fulfilling. Furthermore, I would like to thank all students and co-authors contributing to this thesis in form of countless hours discussing, implementing, and challenging my ideas ultimately leading to the publications summarized in this thesis.

To my family—Sissi, Stefan, Amelie, and Stephanie—your continuous encouragement and confidence in me have been foundational for pursuing this PhD and finalizing my thesis. I cannot thank you enough for always being there, and I will always be grateful for it.

I would like to express my heartfelt appreciation to Anna-Lena Becker. Anna-Lena, the past five years would have not been possible without your ability to listen, offer advice, and simply be there for me. Thank you for being my partner, my confidante, and my source of joy and balance.

To all mentioned and those unmentioned who have contributed to my journey in any kind of way, I am truly grateful. This dissertation does not only reflect my personal effort but also the support and guidance I've received from each one of you.

Lastly, I would like to acknowledge the usage of Grammarly and ChatGPT for their assistance in proofreading this thesis.

Felix Staudigl, April 2024

Contents

Chapter 1

Introduction

The von Neumann bottleneck has long been recognized as a significant impediment to the performance of conventional computing systems. This bottleneck arises from the strict separation between memory and processing units, necessitating frequent data transfers and resulting in suboptimal system performance and energy efficiency. To overcome this challenge, emerging computing paradigms, such as Computing-in-Memory (CIM), have garnered considerable attention. By shifting computational operations inside the memory, CIM endeavors to mitigate the limitations imposed by the von Neumann bottleneck. The concept of relocating computational operations to memory is derived from the mammalian brain, giving rise to the term *"neuromorphic computing."*

Memristors, initially postulated as the fourth fundamental circuit element by Leon Chua [44] in 1971, serve as the foundational building blocks for neuromorphic computing systems. Leveraging the resistive switching characteristics of memristors, these devices offer promising advantages, including high density, non-volatility, and low static power consumption. Based on these unique characteristics, memristors allow the implementation of two flavors of CIM: analog CIM and Logic-in-Memory (LIM).

Analog CIM leverages the parallelism inherent in memristive devices to perform computational operations directly within the memory arrays. This approach offers the potential for higher throughput and overall computing performance compared to conventional system architectures but requires the use of Analog-to-Digital Converters (ADCs) and Digital-to-Analog Converters (DACs). On the other hand, LIM takes a different approach by implementing binary logic gates directly within the memristive crossbar arrays. This eliminates the need for ADCs and DACs, simplifying the system architecture but offers lower computing performance.

However, the reliability of memristors remains a critical concern, exerting a substantial influence on the overall reliability of CIM architectures. For analog CIM, the variability of memristive devices, caused by fabrication process variations and aging effects, introduces uncertainties and affects accuracy. LIM may offer a more robust solution compared to analog CIM, as it operates in the digital domain, which is less affected by the inherent variability of memristive devices.

Alongside performance limitations, conventional computing systems also suffer from inherent hardware security vulnerabilities. Hardware security focuses on identifying specific design characteristics that may be exploited to gain unauthorized control over the entire system [167]. Due to the rigid structure of Integrated Circuits (ICs), these vulnerabilities pose considerable challenges in terms of remediation, persisting throughout the entire operational lifespan of the IC. The emergence of hardware-

enabled attacks, including notable examples such as Rowhammer [105], Spectre [106], and Meltdown [130], has underscored the potential for significant disruptions to critical computing infrastructure. Consequently, the investigation of hardware security vulnerabilities and the development of corresponding countermeasures have become a vital area of research [158]. These vulnerabilities pose a grave risk to the confidentiality and integrity of sensitive data, posing an interesting question: *Is neuromorphic computing susceptible to hardware security attacks?*

Addressing both reliability concerns and hardware security vulnerabilities is of paramount importance in advancing neuromorphic systems based on memristive devices. This thesis seeks to investigate the reliability and hardware security aspects of neuromorphic systems, with a specific focus on memristor-based CIM architectures. By comprehensively understanding and effectively mitigating these challenges, this research aims to contribute to the development of secure and reliable computing systems for future applications. This thesis contributes to the state of the art as follows:

Fault Injection Platform: The implementation of the first fault injection framework for LIM operations, capable of investigating reliability from the memristor level to actual full-fledged workloads. The framework includes a memristor-level fault injection simulator named X-Fault and an operational-level simulator called Faulty Logic-in-Memory (FLIM). The former excels at emulating the impact of faults on individual logic gates with high precision, while the latter offers exceptional simulation speed for executing realistic workloads using abstracted fault models. Together, these simulators allow for a comparison of logic families in terms of fault resilience and enable an investigation at the application level to understand which parameters most significantly influence potential workloads. While both simulators use fault models from the literature, verifying our simulation results with actual hardware is beyond the scope of this work.

NeuroHammer: A novel hardware security attack termed NeuroHammer, which threatens the integrity of the entire neuromorphic system. This attack specifically targets the memristive crossbar arrays utilized in neuromorphic computing systems, intentionally inducing bit-flip faults to undermine the foundational principle of modern computing systems—memory separation. By exploiting the distinct properties of Resistive Random-Access Memory (ReRAM) to alter the switching kinetics through thermal crosstalk, NeuroHammer unveils an attack surface, similar to the Rowhammer attack in Dynamic Random-Access Memory (DRAM). We provide a comprehensive case study showcasing the profound impact of NeuroHammer by leaking an Rivest–Shamir–Adleman (RSA) key from a computing system using memristive memory.

NeuroBreakoutBoard (NBB): The design and implementation of the NBB, a versatile and adaptable instrumentation platform designed to investigate the characteristics of memristive devices at the device, crossbar, and operational levels. Equipped with custom-designed signal generation and sensing circuitry, the NBB enables precise control over memristive cell programming and supports the execution of both analog CIM and LIM operations. In this thesis, we use the NBB to characterize a taped-out ReRAM crossbar array and execute analog CIM and LIM operations.

1.1 Publications

- **Staudigl, F.**, Merchant, F. and Leupers, R., 2021. *A Survey of Neuromorphic Computing-in-Memory: Architectures, Simulators, and Security*. IEEE Design & Test (*journal proceedings*).

- **Staudigl, F.**, Al Indari, H., Schön, D., Sisejkovic, D., Merchant, F., Joseph, J.M., Rana, V., Menzel, S. and Leupers, R., 2022. *NeuroHammer: Inducing Bit-Flips in Memristive Crossbar Memories*. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE) (*conference proceedings*).

- **Staudigl, F.**, Sturm, K.J., Bartel, M., Fetz, T., Sisejkovic, D., Joseph, J.M., Pöhls, L.B. and Leupers, R., 2022. *X-Fault: Impact of Faults on Binary Neural Networks in Memristor-Crossbar Arrays with Logic-in-Memory Computation*. In Proceedings of the IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS) (*conference proceedings*).

- **Staudigl, F.**, Fetz, T., Pelke, R., Sisejkovic, D., Joseph, J.M., Pöhls, L.B. and Leupers, R., 2023. *Fault Injection in Native Logic-in-Memory Computation on Neuromorphic Hardware*. In Proceedings of the Annual Design Automation Conference (DAC) (*conference proceedings*).

- **Staudigl, F.**, Fetz, T., Pelke, R., Sisejkovic, D., Joseph, J.M., Pöhls, L.B. and Leupers, R., 2023. *A Holistic Fault Injection Platform for Neuromorphic Hardware*. In Proceedings of IEEE Latin American Test Symposium (LATS) (*conference proceedings*).

- **Staudigl, F.**, Hossein, M., Ziegler, T., Al Indari, H., Pelke, R., Siegel, S., Wouters, D.J., Sisejkovic, D., Joseph, J.M., and Leupers, R., 2023. *Work-in-Progress: A Universal Instrumentation Platform for Non-Volatile Memories*. In Proceedings of the IEEE International Conference on Hardware/Software Codesign and System Synthesis (CODES/ISSS) (*conference proceedings*).

- **Staudigl, F.**, Al Indari, H., Schön, D., Chen, H.-Y., Sisejkovic, D., Joseph, J. M., Rana, V., Menzel, S., Hagelauer, A., Leupers, R., 2024. *It's Getting Hot in Here: Hardware Security Implications of Thermal Crosstalk on ReRAMs*. In IEEE Transactions on Reliability (*journal proceedings*).

- **Staudigl, F.**, Thoma, J. P., Niesler, C., Sturm, K.J., Pelke, R., Sisejkovic, D., Joseph, J. M., Güneysu, T., Davi, L., Leupers, R., 2024. *NVM-Flip: Non-Volatile-Memory BitFlips on the System Level*. In Proceedings of the ACM Workshop on Secure and Trustworthy Cyber-Physical Systems (SaT-CPS) (*conference proceedings*).

1.2 Synopsis and Outline

This chapter highlights the emergence of neuromorphic computing as a response to the von Neumann bottleneck, which constrains the computing performance and energy efficiency of conventional computing systems. It introduces the memristor as the fundamental building block of this novel computing paradigm. Additionally, the chapter addresses the unresolved reliability and hardware security concerns and provides an overview of the contributions made by this thesis.

The remainder of this thesis is organized as follows. Chapter 2 provides an overview of memristive memory technology, the CIM paradigm, and reliability concerns associated with memristive devices. The related literature is discussed in Chapter 3. Chapter 4 introduces the fault injection framework, along with two comprehensive case studies investigating the fault resilience of logic families and Binary Neural Networks (BNNs). Following this, Chapter 5 presents NeuroHammer, accompanied by a case study illustrating its potential risks through the leakage of an RSA key. Chapter 6 explores the design and implementation of the NBB, followed by a characterization of a commercially available ReRAM technology node. Lastly, Chapter 7 concludes the thesis and outlines future research directions.

Chapter 2

Background

The objective of this chapter is to provide the essential background information that will facilitate a comprehensive understanding of the contributions made in this thesis. Section 2.1 explores the working principles of Emerging Non-Volatiles Memories (eNVMs). The Computing-in-Memory (CIM) paradigm is explained in Section 2.2. Moreover, Section 2.3 discusses the reliability aspects associated with memristor-based memories. The hardware security of both conventional and emerging computing systems is detailed in Section 2.4. Finally, Section 2.5 marks the conclusion of this chapter.

2.1 Emerging Non-Volatile Memories (eNVMs)

Memristive devices serve as fundamental building block of eNVMs. The following section introduces the working principles of these devices and discusses typical memory structures.

2.1.1 Memristive Devices

The data storage in memristive devices hinges upon altering resistance by applying specific voltage pulses across the device terminals. Varied resistive states are achievable based on the polarity, amplitude, and pulse duration, increasing memory density and positioning this technology at the forefront of memory applications and in-memory computing paradigms [179]. In the case of binary switching devices, resistive states are identified as the High Resistive State (HRS) and Low Resistive State (LRS). Memristive devices are categorized according to the underlying resistance switching mechanism.

Diverse mechanisms are employed by various types of memristive devices. Phase Change Memories (PCMs) transition from a non-conducting phase (amorphous) to a conducting phase (crystalline) via Joule heating processes [67]. Spin-Transfer Torque Random-Access Memories (STT-RAMs) utilize spin-polarized currents to switch the magnetization of the free layer in magnetic tunnel junctions, limiting tunneling current [152]. In Ferroelectric Tunnel Junctions (FTJs), the transport of electrons is modulated by the field-induced polarization switching of a ferroelectric layer [46].

However, this thesis centers on Resistive Random-Access Memories (ReRAMs), which modulate resistance through the redistribution of ionic defects [199]. Specifically, the Valence Change Material (VCM) is employed, typically utilizing oxygen vacancies as mobile defects to modify local conductivity. This alteration modifies

Table 2.1: Summary of commercial and academic prototypes using ReRAMs for memory and computing applications [61].

Institution	Node	Stack	Capacity	Cell	Endurance	Ref.
Crossbar Inc.	180 nm			1R		[92]
	45 nm			1T1R		[54]
Sony	180 nm	Cu CBRAM	4 MB	2T 1S1R	10^7	[151]
		CuTe CBRAM		1T1R		[213]
TSMC	40 nm	HfOx	11 MB	1T1R	10^4	[41]
	22 nm		13.5 MB			[42]
Weebit Nano	28 nm	HfO2	16 kB	1T1R	10^5	[73]
Tsinghua Uni.	130 nm	HfOx	16 MB	1T1R	10^6	[39]
Intel	22 nm	Cu CBRAM	4 MB	1T1R		[89]
Panasonic	40 nm	TaOx	1 Mbit		10^5	[215]
HP Labs		HfO2	16 kB	1T1R		[122]
IHP	130 nm	HfO2	4 kbit	1T1R		[201]

the electrostatic barriers at the metal/oxide interfaces by changing the width of the depletion layer [211, 200, 66]. This redistribution of ionic defects within a specific filamentary region allows for distinct resistive states. For instance, a high concentration of oxygen vacancies at the metal interface's disc region, known as the active electrode interface, places the device in the LRS [141]. Conversely, a low concentration of oxygen vacancies at this interface results in the device being in the HRS. The transition between these states, known as the SET transition (HRS to LRS) or RESET transition (LRS to HRS), is achieved by applying specific voltage polarities to the active electrode. The movement of oxygen vacancies, driven by their positive charge, governs these transitions. Joule heating can expedite ion migration, facilitating highly nonlinear switching dynamics, stable read operations at low voltages, and faster switching at slightly higher voltages [144, 194]. Theoretical models by Menzel et al. [144, 143] offer insights into relevant model parameters associated with these dynamics. Filamentary VCM cells are preferred for emerging computing paradigms due to their compatibility with fabrication processes and Complementary Metal-Oxide Semiconductor (CMOS) technology. Table 2.1 provides an overview of commercial and academic prototypes utilizing ReRAMs. Promising oxide materials like HfO_2 and Ta_2O_5, used by companies such as Panasonic and TSMC for ReRAM macros, highlight their in-memory computing capabilities [86, 28, 115, 196, 189, 215, 123, 41].

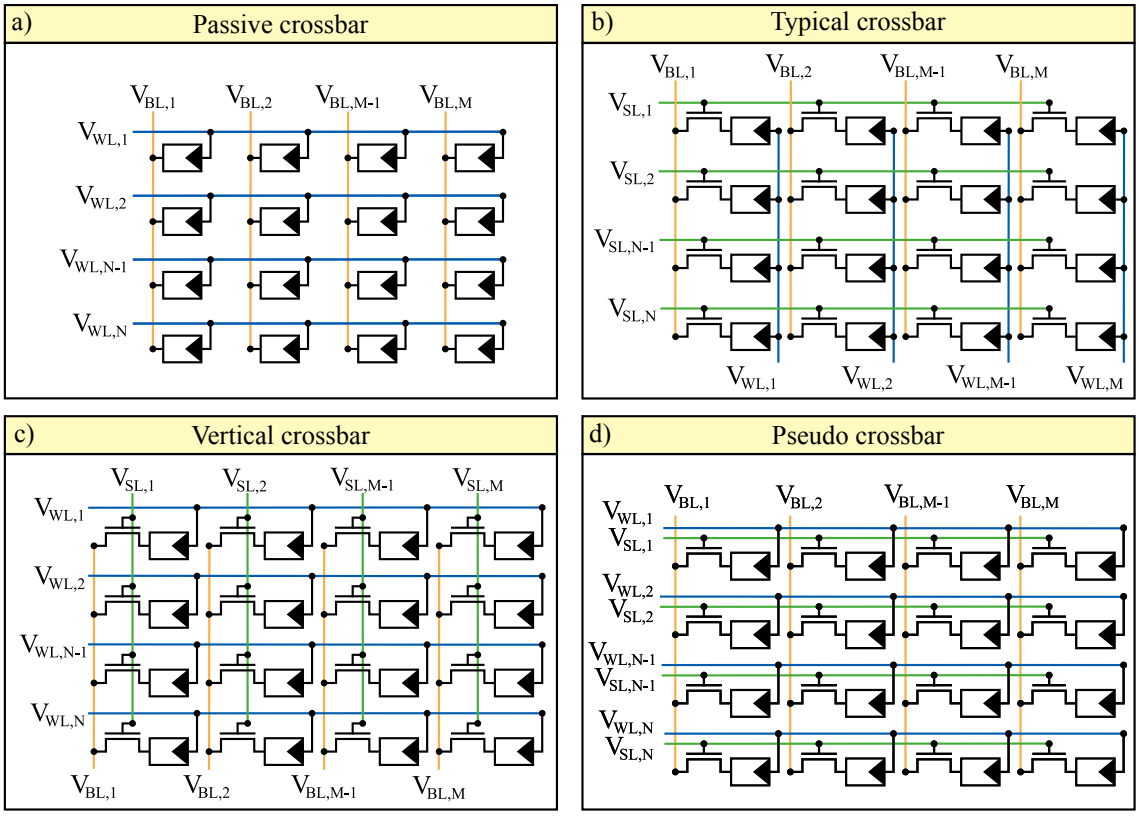


Figure 2.1: Overview of memristive crossbar structures: (a) passive crossbar, (b) typical crossbar, (c) vertical crossbar, and (d) pseudo crossbar.

2.1.2 Memristive Crossbar Structures

Crossbar structures stand out as the primary choice for achieving high-density memristive memories. At their core, these structures interconnect memristive cells through vertical Bit Line (BL) and horizontal Word Line (WL), as depicted in Figure 2.1 (a). To program a single memristive cell, a high voltage V_{write} is applied to the corresponding WL, while the corresponding BL is set to Ground (GND). Ensuring the non-selected cells remain protected against unintended programming is essential. All other lines are set to $V_{\text{write}}/2$ to adhere to the "$V/2$ scheme" [127]. Ideally, a voltage drop of V_{write} occurs across the selected device, while the non-selected devices along the WL and BL experience an absolute value of $|V_{\text{write}}/2|$. However, the ideal scenario, disregarding line resistances and capacitances, introduces parasitic sneak path currents. These currents not only impact the read operation's accuracy but also limit array size, induce undesired switching events, and increase power dissipation [129].

Hence, more sophisticated crossbar structures emerged designed to mitigate sneak path currents in passive crossbar arrays. Intel's 3D Xpoint memory, for instance, integrates threshold switches in series with the memristive devices, effectively reducing sneak path currents. Although this design offers high memory density, it necessitates increased switching voltages [88]. Another approach employs active crossbar arrays incorporating transistors as selectors. However, as transistors are three-terminal de-

vices, introducing an additional line connecting the transistor gates reduces memory density. Figure 2.1 (b-d) illustrates three distinct 1-Transistor 1-Resistor (1T1R) topologies—typical 1T1R array, vertical 1T1R array, and pseudo-crossbar array—each characterized by different arrangements involving exclusively horizontal and vertical lines [171]. In the following, the writing schemes for the active crossbar structures are detailed corresponding to Figure 2.1:

Typical 1T1R array: To write a cell (n,m), all transistors of the n-th line are selected by applying the respective gate voltage V_{Gate}, while all other select lines are set to GND. Naturally, only the devices along the selected row can be programmed. Due to the WLs and BLs are arranged in parallel, the device (n,m) can be programmed by applying voltages to the m-th WL and the m-th BL. All other WLs and BLs are set to 0 V, resulting in a voltage drop solely across the desired device (n,m) (see Figure 2.1 (b)).

Vertical 1T1R array: In the vertical 1T1R configuration, the transistor gates within a column share the same select line. Consequently, to program a cell (n,m), the m-th select line is set to V_{Gate}, the n-th WL to V_{Write}, the m-th BL is set to GND, and the other WLs are set to same potential as the m-th BL.

Pseudo 1T1R array: In the pseudo-crossbar array, the select line and the word line are parallel to each other. To program cell (n,m), the n-th select line is set to a high potential, the n-th WL is set to V_{Write}, the m-th BL is set to GND, and the remaining BLs need to be set to V_{Write} to prevent switching in other cells of the same row.

While the described writing schemes summarized in Table 2.2 effectively mitigate sneak path currents, active crossbar structures also suffer from shortcomings. For instance, the utilization of both negative and positive write voltages could cause leakage currents through the parasitic diodes to the transistor bulk.

Moreover, the voltage drop over WLs and BLs (IR drop) has the potential to impact the read and program pulses significantly. Irrespective of the 1T1R configurations, one terminal of each memristive cell is linked through a shared line with other devices in a row or column. Consequently, when a selected transistor permits the flow of current, it induces a potential shift at the shared line due to the IR drop. The extent of the IR drop is influenced by various factors, such as the length of the current path between the WL drivers and the BL drivers. This length, often termed the critical length, fluctuates depending on the position of the selected cell within the pseudo and the vertical 1T1R array. However, it remains constant regardless of the cell's placement in the typical 1T1R array, as shown in Figure 2.1 (b-d).

Similarly, the leakage currents from the transistor might result in an unintended voltage drop over an unselected cell if a potential is applied between the WL and BL. These leakage currents are anticipated to emerge in more advanced technology nodes, posing a threat to the reliability of active crossbar structures.

Table 2.2: Writing schemes for the SET/RESET operation on a typical, vertical, pseudo, and passive crossbar array.

Type	SET		RESET	
Typical	$V_{\mathrm{WL},m} = V_{\mathrm{SET}}$ $V_{\mathrm{SL},n} = V_{\mathrm{GATE}}$ $V_{\mathrm{BL},[0,M]} = \mathrm{GND}$	$V_{\mathrm{WL},[0,M]\setminus m} = \mathrm{GND}$ $V_{\mathrm{SL},[0,N]\setminus n} = \mathrm{GND}$	$V_{\mathrm{WL},m} = V_{\mathrm{RESET}}$ $V_{\mathrm{SL},n} = V_{\mathrm{GATE}}$ $V_{\mathrm{BL},[0,M]} = \mathrm{GND}$	$V_{\mathrm{WL},[0,M]\setminus m} = \mathrm{GND}$ $V_{\mathrm{SL},[0,N]\setminus n} = \mathrm{GND}$
Vertical	$V_{\mathrm{WL},n} = V_{\mathrm{SET}}$ $V_{\mathrm{SL},m} = V_{\mathrm{GATE}}$ $V_{\mathrm{BL},[0,M]} = \mathrm{GND}$	$V_{\mathrm{WL},[0,N]\setminus n} = \mathrm{GND}$ $V_{\mathrm{SL},[0,M]\setminus m} = \mathrm{GND}$	$V_{\mathrm{WL},n} = V_{\mathrm{RESET}}$ $V_{\mathrm{SL},m} = V_{\mathrm{GATE}}$ $V_{\mathrm{BL},[0,M]} = \mathrm{GND}$	$V_{\mathrm{WL},[0,N]\setminus n} = \mathrm{GND}$ $V_{\mathrm{SL},[0,M]\setminus m} = \mathrm{GND}$
Pseudo	$V_{\mathrm{WL},n} = V_{\mathrm{SET}}$ $V_{\mathrm{SL},n} = V_{\mathrm{GATE}}$ $V_{\mathrm{BL},m} = \mathrm{GND}$	$V_{\mathrm{WL},[0,N]\setminus n} = \mathrm{GND}$ $V_{\mathrm{SL},[0,N]\setminus n} = \mathrm{GND}$ $V_{\mathrm{BL},[0,M]\setminus m} = V_{\mathrm{SET}}$	$V_{\mathrm{WL},n} = V_{\mathrm{RESET}}$ $V_{\mathrm{SL},n} = V_{\mathrm{GATE}}$ $V_{\mathrm{BL},m} = \mathrm{GND}$	$V_{\mathrm{WL},[0,N]\setminus n} = \mathrm{GND}$ $V_{\mathrm{SL},[0,N]\setminus n} = \mathrm{GND}$ $V_{\mathrm{BL},[0,M]\setminus m} = V_{\mathrm{RESET}}$
Passive	$V_{\mathrm{WL},m} = V_{\mathrm{SET}}$ $V_{\mathrm{BL},n} = \mathrm{GND}$	$V_{\mathrm{WL},[0,M]\setminus m} = \frac{V_{\mathrm{SET}}}{2}$ $V_{\mathrm{BL},[0,N]\setminus n} = \frac{V_{\mathrm{SET}}}{2}$	$V_{\mathrm{WL},m} = V_{\mathrm{RESET}}$ $V_{\mathrm{BL},n} = \mathrm{GND}$	$V_{\mathrm{WL},[0,M]\setminus m} = \frac{V_{\mathrm{RESET}}}{2}$ $V_{\mathrm{BL},[0,N]\setminus n} = \frac{V_{\mathrm{RESET}}}{2}$

2.2 Computing-in-Memory (CIM)

Besides the utilization of memristive crossbar arrays as memories, these structures are capable of facilitating CIM operations. In general, CIM can be realized in two different flavors: analog CIM and Logic-in-Memory (LIM).

2.2.1 Analog Computing-in-Memory (CIM)

Analog CIM harnesses the continuous resistance values of memristors to conduct Multiply–Accumulate (MAC) operations in the analog domain. As depicted in Figure 2.2 (a), the operational principle of analog CIM involves the Digital-to-Analog Converter (DAC) translating the binary input vector into respective voltages and feeding them to the crossbar's rows. According to Ohm's and Kirchhoff's law, the resulting current of column c is defined as:

$$i_{c,\mathrm{res}} = \sum_{r=1}^{R} G_{r,c} V_r \tag{2.1}$$

Here, R signifies the number of rows, $G_{r,c}$ indicates the conductance of the memristor in row r and column c, and V_r stands for the input voltage applied to row r. Subsequently, the Analog-to-Digital Converter (ADC) translates the output current into a digital value. Analog CIM facilitates the computation of MAC operations in a massively parallel manner, providing high precision and low latency, which is beneficial for applications such as machine learning and signal processing. However,

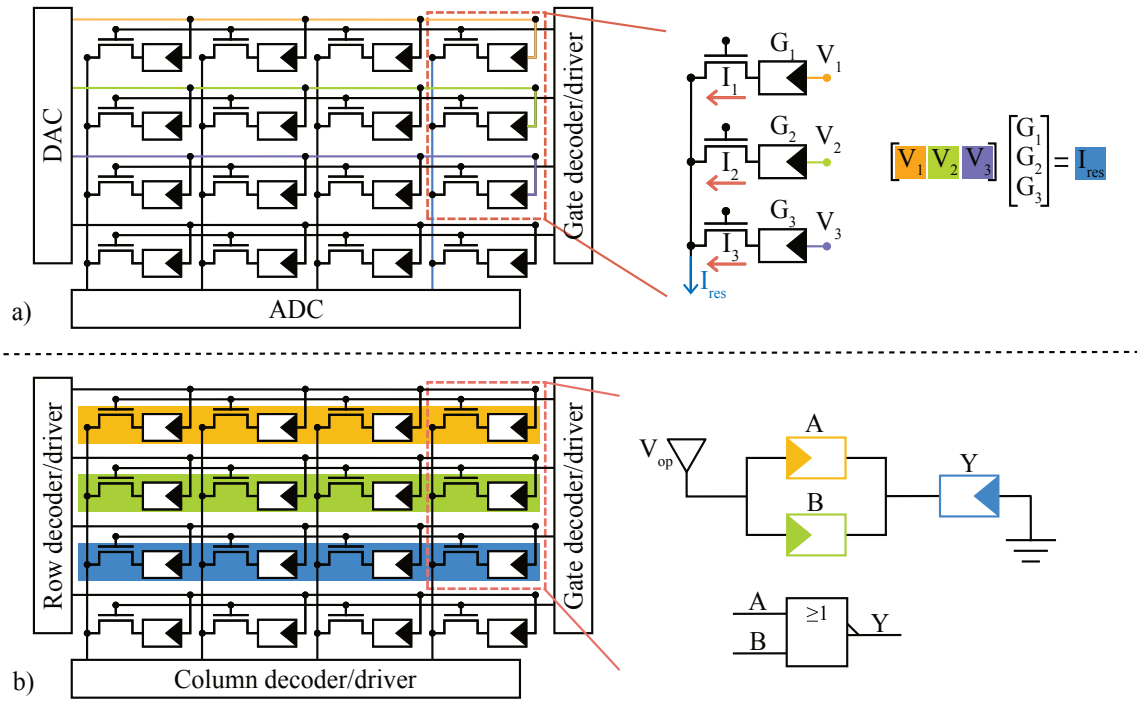


Figure 2.2: Overview of CIM flavors: (a) analog CIM and (b) LIM.

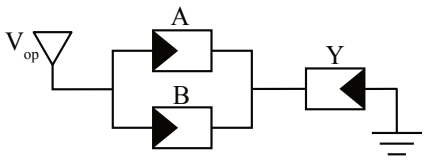
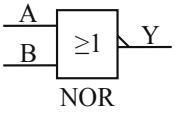
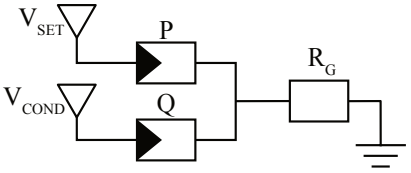
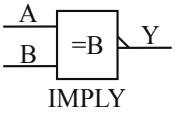
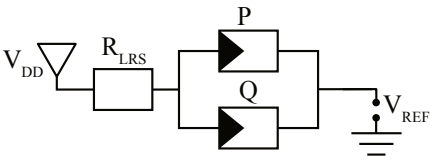
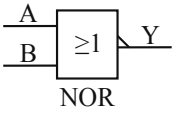
it involves complex peripheral circuitry, notably utilizing ADCs/DACs, which are known for their considerable silicon footprint, increased energy consumption, and latency limitations [185]. Moreover, the variability of the memristors directly impacts $i_{c,res}$, leading to a reduction in the accuracy of the executed application [132].

2.2.2 Logic-in-Memory (LIM)

On the contrary, LIM leverages memristive crossbar arrays in a binary manner to execute logic operations. Numerous logic families have been proposed to integrate logic gates within memristive crossbar arrays [23, 218, 147, 5, 76, 204, 27, 118, 120]. Most of these logic families require additional CMOS circuitry, such as a CMOS inverter [112], to implement logic gates. However, stateful logic represents a group of logic families that aims to conduct logic gate operations entirely within memristive crossbar arrays, eliminating the need for complex external circuitry [84]. Stateful logic gates encode their inputs and outputs in the form of resistance. During computation, the result is directly stored into the output memory cell without data being transferred outside the memory array [162]. Subsequently, the Memristor-Based Material Implication (IMPLY) [111], Memristor-Aided Logic (MAGIC) [110], and Memristor Ratioed Logic (MRL) logic families are elaborated upon, as they represent the most notable logic families.

Memristor-Based Material Implication (IMPLY): The IMPLY logic family features a single logic gate known as the Material Implication (IMP) gate, as shown in Table 2.3 [27]. Together with the FALSE gate, which consistently yields zero, the IM-

Table 2.3: Implementation details for MAGIC, IMPLY, and MRL [111, 110, 60].

	Circuit	Logic gate	Input/output encoding
MAGIC			Resistance/Resistance
IMPLY			Resistance/Resistance
MRL			Resistance/Voltage

PLY gate constitutes a functionally complete set. In its standard configuration, the IMPLY gate consists of two memristors, denoted as P and Q, linked to a resistor R_G with $R_{ON} < R_G < R_{OFF}$. The resulting output is written in the memristor Q based on the initial resistances p and q of the two memristors. To calculate the output of the logic gate, distinct voltages are applied to the input memristors. The voltage V_{SET} is directed to Q, while V_{COND} is connected to P, ensuring that $|V_{COND}| < |V_{SET}|$. The IMPLY logic family allows the execution of an arbitrary Boolean function using only $n + 3$ memristors, employing either FALSE operations with a single memristor or IMPLY operations with two memristors in a sequential manner [118]. Additionally, the memristive IMPLY gate can be incorporated into a memristive crossbar array where P, Q, and the necessary resistor R_G are connected via the same BL. Both voltages, V_{SET} and V_{COND}, are supplied through their respective WLs [111].

Memristor-Aided Logic (MAGIC): The IMPLY logic family's broader utility faces limitations due to the necessary external resistance R_G and the inherent deletion of input values during computation. To address these issues, the MAGIC logic family offers a functionally complete NOR gate. Table 2.3 illustrates the fundamental structure of the MAGIC NOR gate. The MAGIC NOR gate comprises two input memristors, in_1 and in_2, along with a dedicated output memristor out that retains the result value post-computation. Unlike the IMPLY gate, the MAGIC NOR gate preserves the input values. Upon initializing in_1, in_2, and out, an operational voltage V_{Op} is applied across the logic gate. As a result, the output memristor changes its internal state based on

the resistances of the input memristors. The operational voltage must adhere to the constraint:

$$2V_{T,OFF} < V_O < \min\left[\frac{R_{OFF}}{2R_{ON}}V_{T,OFF}, |V_{T,ON}|\right] \tag{2.2}$$

Here, $V_{T,OFF}/V_{T,ON}$ denotes the voltage threshold of the memristor, and R_{OFF}/R_{ON} signifies the resistances for logical zero and logical one. By implementing IMPLY/-MAGIC gates within a crossbar structure, these logic gates can operate in parallel by applying the operation voltage V_{Op} to the appropriate input rows. Figure 2.2 (b) illustrates MAGIC NOR gates embedded in the crossbar structure, where each column is equipped with three memristive devices to form a NOR gate. In this example, applying V_{Op} to the first and second rows enables the execution of four parallel NOR operations.

Memristive Ratioed Logic (MRL): Given that both MAGIC and IMPLY logic families encode Boolean states in the form of resistance, they are inherently prone to state-drift and variability effects of memristive devices. MRL adopts a different methodology by encoding the output in a voltage level between V_{REF} and V_{DD}. Consequently, the result of the computation is not directly stored within the crossbar array, positioning this logic family as near-memory computing. In general, this logic family employs a NOR gate configured with a voltage divider across two memristive devices, where the input values are determined by the resistances of these devices. A logical 1 is represented by R_{HRS}, and a logical 0 is denoted by R_{LRS}, with a computing voltage of V_{REF}. For instance, when the input is $\{0,0\}$, both memristive devices are set to LRS, leading to an output voltage of $V_{out} \approx V_{REF}$, signifying a logical 1. Conversely, for any other input combination, V_{out} is reduced to below $\frac{V_{DD}-V_{REF}}{2} + V_{REF}$, which is interpreted as a logical 0 [60, 56, 55].

2.3 Reliability Aspects of ReRAMs

Notwithstanding their potential to enable novel computing paradigms, memristors face susceptibility to faults attributed to an immature manufacturing process and limited endurance [133]. Consequently, this section serves to introduce crucial terminology, describe relevant fault models, and explore fault injection methodologies concerning ReRAMs.

2.3.1 Terminology

In general, a system may deviate from its intended operation due to various *factors of dependability*, which describe the origins and consequences of system malfunctions as follows [222]:

Definition 2.1. A **fault** is a physical defect, imperfection, or flaw within hardware or software.

Definition 2.2. An **error** represents a departure from precision or correctness stemming from a fault.

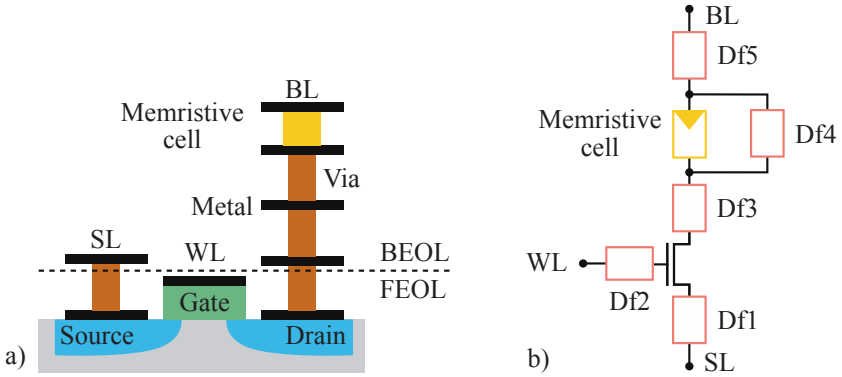


Figure 2.3: Overview of (a) process schematic of an integrated ReRAM cell, and (b) fault model of a 1T1R memory cell, utilizing resistors to emulate defects. [192]

Definition 2.3. A **failure** entails the absence of the expected or anticipated execution of a particular action.

To illustrate this terminology, consider a memory defect in a computer system due to deterioration, which constitutes a fault. This localized fault can cause errors, such as a bit-flip in a memory cell, leading to a malfunction in a calculation during a user application's read operation, resulting in a failure [183, 222].

Memristive devices are regarded as being in an immature state, affected by noticeable defect densities, manufacturing variations, and susceptibility to temperature and voltage fluctuations. Imperfections stemming from various sources, such as spot defects, assembly discrepancies, and fabrication intricacies, significantly impact not only production yield but also the reliability of memristive devices [16]. Such influence results in parametric and logic faults, which can be classified into two distinct categories: *soft faults* and *hard faults*. Soft faults arise from Cycle-to-Cycle (C2C) or Device-to-Device (D2D) variations, in-field read/write operations, and instances of retention faults where the cell content changes over time. Hard faults may arise from variations during the fabrication process, as well as spot defects, extreme parametric deviations, and continuous stress leading to the device being open or shorted [192].

2.3.2 Defects

The manufacturing of ReRAMs can be divided into two primary phases: the Front End Of Line (FEOL) and the Back End Of Line (BEOL) processes [62]. As depicted in Figure 2.3 (a), the FEOL encompasses the creation of transistors, while the BEOL involves the fabrication of metal layers and the ReRAM device. The ReRAM device is typically positioned between metal layers M4 and M5 [74]. During the FEOL phase, transistors are created on the wafer following the standard CMOS process flow. Although this process is well-established, various defects could disrupt the functioning of the transistors, such as patterning proximity effects, line-edge and line-width roughness, polish variations, and fluctuations in the gate dielectric [109]. Moving on to the BEOL phase, the fabrication starts with the creation of metal layers. Impurity

deposition commonly affects these metal layers, leading to defects at the electrical level [87]. Such imperfections cause resistive open defects in the metal lines connecting the source, drain, and gate of the transistor, as shown in the defect model in Figure 2.3 (b). Finally, the ReRAM device is placed between the metal layers. This process involves the deposition of various materials, each of which holds the potential to introduce defects that hinder the proper functioning of the memory cell. The bottom electrode's deposition can be influenced by chemical and physical conditions, significantly impacting the forming process and, consequently, the quality of the LRS. The resistive switching material is also susceptible to various issues, leading to defects, such as thick or thin local spots. Additionally, the top electrode could induce parameter variations and defects [192, 61].

2.3.3 Fault Models

Fault models are a way to abstract physical phenomena for understanding the impact of defects on a system [183]. Several fault models have been introduced to describe the issues related to flawed ReRAMs. Given the similarities between ReRAMs and traditional memories, many traditional fault models are applicable to ReRAMs. These fault models can be grouped into two main categories: *traditional fault models* and *unique fault models* [63, 38, 61].

Traditional Fault Models: In the following, fault models are outlined typically observed in both traditional memories (like Static Random-Access Memories (SRAMs) and Dynamic Random-Access Memories (DRAMs)) and ReRAMs.

- **Stuck-at-Fault (SAF)**: The cell consistently resides either in the LRS, designated as *Stuck-at-0*, or in the HRS, termed as *Stuck-at-1*. A Stuck-at-0 condition, classified as a hard fault, materializes when these faults stem from factors like the existence of substantial resistive open defects along the WL (Df2), a persistent open switch (characterized by the access transistor being predominantly in the OFF state), or the presence of a sufficiently large resistive defect, denoted as Df4, which diverts the current in the resistive device (see Figure 2.3 (b)). These SAFs can also manifest as soft faults if induced by an erroneous forming process. Over-forming the cell results in a Stuck-at-0, signifying that the LRS value falls below its nominal threshold, potentially leading to an incomplete RESET operation due to limitations in the write driver's strength. In the event of a complete failure during the forming operation, resistive switching remains inert, rendering the cell in a Stuck-at-1 [192, 81, 38].

- **Read Destructive Fault/Deceptive Read Destructive Fault (RDF/DRDF)**: During a read operation, the data held within the cell changes as a consequence of the read process and the cell returns the incorrect output. In comparison, the DRDF returns a correct output, but also changes the data held within the cell. Such anomalies predominantly constitute soft faults and manifest in cells with diminished strength, indicated by LRS (HRS) values surpassing (falling below) the designated norm [35, 50, 192].

- **Incorrect Read Faults (IRF)**: The cell returns an incorrect output, although the data stored within the cell remains correct and unaffected by the read process. These irregularities predominantly constitute hard faults, attributed to resistive defects within the memory cell, exemplified by Df3, Df4, and Df5 in Figure 2.3 (b) [192, 80].

- **Slow Write Fault (SWF)**: Slow write faults are characterized by the inability to successfully complete the write operation within the designated time. These faults can be classified as hard faults if they arise from minor resistive defects at Df2, Df3, and Df4 in Figure 2.3 (b). Alternatively, they may be categorized as soft faults when they stem from factors such as a weak access transistor, inadequate capping layer deposition, incorrect stack etching, or due to aging [148, 192].

- **Coupling Fault (CF)**: Coupling faults occur when a write/read operation on one memory cell inadvertently triggers a write operation in an adjacent cell. These faults are categorized as either soft or hard, depending on their origin. Factors such as electromagnetic interference, crosstalk, or manufacturing imperfections can create unintended electrical pathways or influence between adjacent cells, leading to these coupling faults. [32, 126].

Unique Fault Models: Resistive Random-Access Memories have distinct malfunction patterns that require unique fault models to describe their behavior.

- **Undefined Write Fault (UWF)**: During a writing operation, the cell enters an undefined state, positioned between the states LRS and HRS. This anomaly is triggered by an insufficient bias voltage during the writing process, particularly occurring in cells with weaker characteristics, possibly magnified by C2C variability. Consequently, if a read operation is conducted on this particular cell, an arbitrary logic value will be obtained [81]. This fault's manifestation is twofold: it can be detected in a newly manufactured cell, attributed to pronounced process variations such as inadequate forming, or in a cell that has aged, resulting from the gradual shift in resistance over time [150, 192, 63].

- **Deep State Fault (Deep)**: The cell's resistance exceeds its designated limits, manifesting as a scenario where the resistance in the LRS falls below R_{LRS}, while it exceeds the set limit in the HRS [97]. Such a vulnerability may be attributed to factors like over-forming or variations in C2C characteristics [63].

- **Unknown Read Fault (URF)**: When a read operation is conducted, the output yields an arbitrary logic value, regardless of the conditions of the reading process. This form of vulnerability emerges as a soft fault when the LRS and HRS are situated in proximity to each other and consequently close to the reference resistance [63, 192, 97].

2.3.4 Fault Injection

Fault injection techniques have long been acknowledged as indispensable tools for validating system dependability by analyzing device behavior in the event of a fault occurrence. This subsection outlines fault injection methodologies that have been developed to assess system robustness and behavior under diverse fault scenarios [222]:

- **Hardware-based Fault Injection:** This category involves directly disturbing the hardware at a physical level. This technique encompass varying hardware parameters based on the environment, including heavy ion radiation, electromagnetic interferences, and power supply disturbances [222, 119, 18, 57].
- **Software-based Fault Injection:** With the aim of replicating errors that would arise in hardware due to faults, software-based fault injection operates at a software level. The technique simulates the errors in software that would mimic hardware behavior when affected by faults [222, 119, 18, 57].
- **Simulation-based Fault Injection:** This technique injects faults into high-level simulation models. It enables early assessment of system dependability, especially when only a model of the system exists [222, 107].
- **Emulation-based Fault Injection:** Presenting an alternative to time-intensive simulation-based fault injection, this approach employs Field-Programmable Gate Arrays (FPGAs) for accelerating fault simulation and effective circuit emulation. By utilizing FPGAs, designers can study circuit behavior within the real application environment while considering real-time interactions [222, 137, 57].
- **Hybrid Fault Injection:** This technique combines software-implemented fault injection with hardware monitoring [222].

2.4 Hardware Security

Modern computing hardware is a diverse spectrum of components sourced from various vendors with differing levels of trust. Operating within a mixed-trust environment, these components serve diverse security levels. This complex ecosystem, coupled with the extensive connectivity in modern systems, renders critical hardware resources vulnerable to security threats. To counteract these security threats, hardware security measures are crucially needed [85, 170].

These threats span the entire semiconductor life cycle, from design to recycling, emerging from unintentional design flaws [106, 131], malicious modifications [59, 212, 135], and system side effects [182, 43, 184]. Hardware security threats encompass covert channels [30], side channels [138], hardware Trojans [210], and fault injection attacks [68]. These vulnerabilities target a spectrum of critical components, including cryptographic functions, secure architectures, intellectual property, and machine learning models [85].

To design secure system, hardware security properties encompass formal specifications defining invariant security-related properties. These properties guide security verification tools, limit desirable security attributes and assisting in formulating security countermeasures [165]. Subsequent, the three fundamental hardware security properties are outlined referred to as the CIA triad (confidentiality, integrity, and availability).

1. **Confidentiality** mandates that secret information remains undisclosed when observing public outputs or memory locations. Leaks of sensitive data can occur through system side channels, backdoors, covert channels, or hardware Trojans [85].

2. **Integrity** ensures that trusted data remains unaltered by untrusted entities. Attacks targeting critical memory locations, such as cryptographic keys, program counters, or privilege registers, compromise integrity and serve as stepping stones for subsequent malicious activities [85].

3. **Availability** characterizes a system's capacity to consistently execute its designated operations. This hardware security attribute holds paramount importance, as the absence of availability undermines the assurance of the other two properties. Specifically, Denial-of-Service (DoS) attacks focus on this property, aiming to disable or isolate vital components within the system[156].

The remainder of the chapter introduces three common types of hardware security threats: hardware Trojans, side-channel attacks, and fault injection attacks. Each subsection gives a short introduction of the attack by depicting the working principle and briefly discusses the attack surface.

2.4.1 Hardware Trojans

In contrast to software Trojans, the hardware counterpart poses a significant challenge in their removal, making them a serious and persistent threat to computer systems [91]. Hardware Trojans refer to unauthorized changes made to Integrated Circuits (ICs) by adversaries, introducing undesired functionalities. These alterations exploit the global nature of semiconductor design and manufacturing, causing concerns across multiple sectors such as military, finance, and transportation [186]. The production of ICs involves design, fabrication, and testing. To reduce costs and speed up time-to-market, the fabrication is often outsourced to a foundry, while the design takes advantage of third-party Intellectual Property (IP). As a result, the IC supply chain is vulnerable to various hardware security attacks due to the involvement of potentially malicious third parties [206].

Ensuring the authenticity of chips requires expensive end-to-end trust mechanisms or post-manufacturing validation. Hardware Trojans can impact various ICs, including Application-Specific Integrated Circuits (ASICs), microprocessors, and digital signal processors [186]. Reports from reputable sources like the U.S. Administration,

the U.S. Senate, and IEEE Spectrum emphasize the severity of this issue [187, 168, 2]. Efforts to combat Trojan attacks have focused on three primary solutions: (1) Trojan detection methods, (2) Design For Security (DFS) strategies, and (3) runtime monitoring approaches. Trojan detection methods primarily aim to identify Trojans at the IP level using pre-silicon techniques or nondestructive methods during post-silicon manufacturing tests. DFS methods aim to complicate the insertion of hard-to-detect Trojans or assist in their identification during post-silicon validation. However, Trojan detection and DFS methods often lack complete assurance. In contrast, runtime validation methods involve continuous online monitoring of circuit operation, serving as a final defense against Trojan attacks, aiming to mitigate the impact of activated Trojans [24].

2.4.2 Side-Channel Attacks

Another significant class of hardware attacks involves side-channel attacks, exploiting implementation-specific characteristics to extract secret parameters. These attacks capitalize on unintended physical information leaks, aiming to deduce valuable insights about the operational behavior of the target system [93]. Side-channel attacks work by inferring internal computations through the analysis of external parameters like processing time, power consumption, heat dissipation, and electromagnetic emissions. These attacks pose a particular threat to cryptographic implementations, seeking to expose confidential data such as encryption keys [173]. Meltdown [130] is a well-known example of a hardware security breach that exploits a side-channel attack. This attack enables an attacker to gain unauthorized access to the memory of other processes.

Addressing side-channel attacks does not have a general solution to safeguard a computing system. However, specific design alterations can mitigate distinct information leakage and serve as countermeasures against particular side-channel attacks. For instance, the randomization of operation-dependent values is a typical countermeasure to prevent electromagnetic and power side-channel attacks [173]. Additionally, timing side-channel vulnerabilities can be mitigated by equalizing response times, potentially achieved by delaying operations.

2.4.3 Fault Injection Attacks

Fault injection attacks are considered active physical assaults aiming to maliciously extract cryptographic keys, elevate privileges, or compromise the implementation of neural networks. Essentially, an adversary deliberately injects faults into a computer system and observes the system's response to extract sensitive information. Several fault injection methods have been proven to be effective, including clock/voltage glitching and optical/electromagnetic disturbances. Of particular note, the Rowhammer attack [105] has garnered significant attention due to its impact on DRAMs. This attack allows an adversary to intentionally manipulate bits in nearby memory regions by inducing disruptive errors in modern high-density memories.

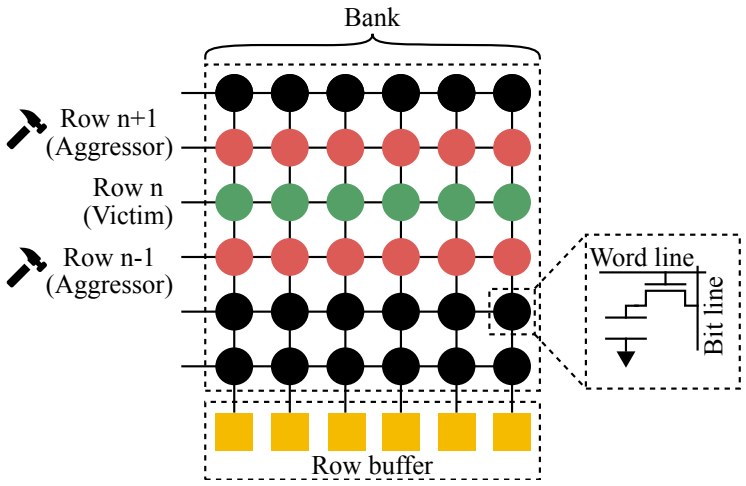


Figure 2.4: Overview of the Rowhammer attack procedure in DRAMs: Hammering the two adjacent rows surrounding the victim row to intentionally trigger bit-flip faults by deliberately diminishing the capacitor's charge.

DRAM cells consist of a capacitor linked to a transistor, with the capacitor's charge encoding two distinct states (refer to Figure 2.4). While all the transistors' gates within a specific row are interconnected via a WL, the corresponding capacitors of a column are linked through a BL. Unfortunately, the charge of these capacitors is typically transient, limiting the retention time. Consequently, the memory controller must continuously refresh the charge of all memory cells to maintain the stored information's integrity. The Rowhammer attack exploits this phenomenon to purposefully diminish the charge of the targeted cell. The disruptive error arises from repetitively targeting a WL, increasing the discharge of neighboring cells, as depicted in Figure 2.4. Rowhammer has been used to gain kernel privileges, enabling Google Project Zero researchers to effectively take control of entire computer systems [167].

2.5 Synopsis

This chapter lays the groundwork for understanding the thesis by discussing eNVMs, CIM paradigms, memristor reliability, and hardware security. It starts by explaining memristive devices, which are central to eNVMs, covering their operation, modeling, and applications in high-density memory structures like crossbar arrays. It then transitions to CIM, detailing how memristive crossbar arrays enable analog and logic operations within memory, presenting a more efficient alternative to traditional computing architectures. The reliability of memristive devices, particularly ReRAMs, is outlined next. The chapter discusses the various faults that can occur in ReRAMs due to manufacturing imperfections or operational stresses, and it introduces fault injection as a method to evaluate system robustness in the face of such faults. Finally, the chapter addresses the critical issue of hardware security. It covers the spectrum of threats across the hardware lifecycle, from design and manufacturing to deployment,

includes hardware Trojans, side-channel attacks, and fault injection attacks. The chapter concludes by emphasizing the importance of confidentiality, integrity, and availability (the CIA triad) in designing secure systems and the ongoing challenges in protecting against sophisticated hardware attacks.

Chapter 3

Related Work

Since Leon Chua's introduction of the memristor in 1971, substantial research efforts have been dedicated to establishing efficient and dependable computing systems rooted in memristor technology [44]. Despite these advancements, memristive devices continue to face persistent reliability challenges. Addressing these issues has become a critical area of research, with efforts aimed at understanding the underlying causes and developing strategies to enhance their stability and longevity. Beyond the need for dependable systems, these reliability flaws also create potential vulnerabilities that could be exploited in hardware security attacks, allowing adversaries to manipulate or extract sensitive information.

This chapter provides a detailed overview of the reliability challenges in memristor-based computing systems and their implications for hardware security. Section 3.1 focuses on the reliability issues specific to memristor-based memory systems. Section 3.2 examines how these reliability weaknesses can be exploited in hardware security attacks and discusses possible defense mechanisms. Section 3.3 covers the development of instrumentation platforms, which are crucial for assessing and improving reliability. Finally, Section 3.4 summarizes the key lessons learned from the survey. This chapter is partially based on the publication presented in [179].

3.1 Reliability of Neuromorphic Computing Systems

Resistive RAMs utilize memristive crossbar arrays to achieve both high memory density and energy efficiency. These crossbar structures not only enable the storage of binary and multi-value data but also support Multiply–Accumulate (MAC) operations. In the quest to explore the reliability characteristics of Resistive Random-Access Memories (ReRAMs), it is imperative to encompass all operational modes. Consequently, this section comprehensively outlines prior research regarding the reliability aspects of ReRAMs, considering their role as conventional memory as well as their involvement in in-memory computing paradigms. Furthermore, this section provides an overview of fault injection platforms and encapsulates the proposed techniques designed to enhance the dependability of neuromorphic systems.

3.1.1 Reliability Aspects of ReRAMs

The landscape of publications exploring the reliability aspects of ReRAM spans from fault detection to the development of memory architectures optimized for reliability. Since the memristor is the fundamental building block, a multitude of publications

have proposed fault models and testing approaches for both binary [38, 78, 161, 134, 94, 33] and multilevel cells [95, 96].

Alongside concerns at the memristor level, the commonly employed crossbar structures introduce challenges such as IR drop issues and read/write disturbances. Zhang et al. [217] introduced a circuit architecture co-optimization framework to address the inherent shortcomings of crossbar structures. They implement a double-sided write driver to diminish IR drops along the bit lines and tackle write disturbances through a disturbance detection scheme. Furthermore, they partition the crossbar structure into multiple regions for storing cold and hot data in slow and fast regions, respectively, enhancing both latency and reliability. The integration of these enhancements yields a 26.1% performance improvement while reducing energy consumption by 21.6%. This approach also enhances system reliability by countering read/write disturbances and the IR drop problem.

While circuit-level optimizations effectively enhance reliability, they tend to increase silicon area and energy consumption. Consequently, Mao et al. [140, 139] investigated the impact of read/write voltages and pulse lengths on ReRAM reliability and endurance. Conventionally, SET and RESET operations use distinct voltages. However, the authors advocate for the use of a single voltage for both operations, reducing write latency and energy consumption. Furthermore, they argue that retention can be extended by adjusting word line, bit line, and select line voltages rather than solely relying on altering the ON/OFF ratio of the memristor.

Conventional memories leverage Error Correcting Codes (ECCs) for error detection and rectification [79]. As a result, various error correction techniques have been proposed to augment ReRAM reliability. Schechter et al. [166] introduced an Error Correction Pointer (ECP) scheme to address hard errors in ReRAMs by substituting faulty cells with new ones and recording the locations of the faulty cells. In contrast, Xu et al. [209] proposed an error-resilient ReRAM architecture utilizing ECC and ECP to mitigate retention failures and stuck-at-faults. Moreover, Zheng et al. [220] introduced a detection and recovery scheme to alleviate pseudo-hard errors in ReRAMs by utilizing a higher programming voltage. The authors define a pseudo-hard error as a type of hard error that is still recoverable. Lastly, Zhang et al. [219] proposed a microarchitectural design termed EnTiered-crossbar, partitioning each crossbar along the bit lines into two halves. These near and far segments are isolated using an access transistor, resolving the IR drop issue.

3.1.2 Reliability Aspects of Computing-in-Memory Applications

Memristive crossbar structures offer not just traditional memory functions but also the ability to conduct analog MAC operations and execute logic gates. However, the reliability challenges related to the underlying memristive cells can significantly impact applications built upon them. Consequently, numerous research initiatives have been conducted to evaluate the impact of these reliability concerns on various applications.

Among the most common operational modes of memristor-based accelerators is the execution of analog MAC operations (see Section 2.2). Feinberg et al. [58] introduce an error correction scheme based on arithmetic codes to enhance reliability. Data is encoded by multiplying it with an integer, allowing error detection and correction through modulus operations and a correction table lookup. Additionally, the authors improve their approach by using data-aware encoding, leveraging the state dependence of errors and prioritizing critical computation segments for overall system accuracy.

In addition to data encoding, matrix transformation has been proposed to address stuck-at faults in ReRAM accelerators. Zhang et al. [216] utilize row flipping, permutation, and value range adjustments to fortify weight matrices against stuck-at faults. The row flipping transformation converts stuck-off (stuck-on) faults into stuck-on (stuck-off) faults, while the permutation transformation maps smaller (larger) weights to memristors stuck-off (stuck-on). The value range transformation diminishes extreme element magnitudes in the matrix, thereby reducing errors introduced by each stuck-at fault. Experimental results indicate that this framework can recover 99% of accuracy loss caused by stuck-at faults, eliminating the need for neural network retraining.

Furthermore, mapping algorithms have been annotated to mitigate stuck-at faults by thoroughly exploring the mapping space. Xia et al. [202] introduce an inner fault-tolerant mapping algorithm capable of addressing multiple faulty columns without hardware overhead. The authors use two cells to encode a single value, enabling the algorithm to adjust, in case of a faulty cell, the other cell to represent the correct value. Even before the mapping, the neural network itself can be optimized to mitigate the non-ideal effects of memristive crossbars. Al-Shaarawt et al. [4] introduce the PRUNIX framework for training and pruning convolutional neural networks tailored for deployment on memristor crossbar-based accelerators. PRUNIX addresses various non-ideal characteristics of memristor crossbars, including weight quantization, state-drift, aging, and stuck-at faults. It incorporates a unique Group Sawtooth Regularization to enhance tolerance to non-idealities and promote sparsity. Additionally, it employs the adaptive pruning algorithm to minimize accuracy loss by considering the sensitivity of different CNN layers to pruning.

Finally, Emara et al. [52] discuss production testing of a XOR gate, using memristors in conjunction with Complementary Metal-Oxide Semiconductor (CMOS) inverters. The research specifically investigates the two-input XOR gate using a fault model that includes stuck-at faults for memristors and a fault model for transistors. The analysis reveals that faults within the XOR gate generate analog output voltage values due to the circuit's architecture. Consequently, a specialized 2-bit Flash Analog-to-Digital Converter (ADC) is employed to achieve comprehensive fault coverage. Notably, the study highlights that four resistive short faults in the XOR gate can only be detected by monitoring the input current, emphasizing the need for exhaustive testing to attain 100% fault coverage.

Table 3.1: Overview of simulation platforms used for evaluating the reliability of non-volatile memories [114, 31].

Simulation platform	Prog. language	Inference	Training	Open-source	Supported devices	Non-idealities
GENIEx [31]	Python	✓	✗	✗	Non-volatile memories.	Selector/device parasitics, source/sink/wire resistances
CrossSim [207]	Python	✓	✓	✓	Non-volatile memories.	C2C/D2D variations, programming errors, conductance drift, read noise variability, and ADC precision loss.
NeuroSim [37]/NeuroSim+ [36]/ NeuroSim+DNN [154]	C++, Python	✓	✓	✓	Non-volatile memories, legacy NAND flash.	C2C/D2D variations, device non-linearities.
SySCIM [169]	C++, SystemC	n/a	n/a	✗	Non-volatile memories.	C2C/D2D variations, device, interconnect parasitics, SAF.
IBM Analog Hardware Acceleration Kit [159]	C++, Python	✓	✓	✓	Non-volatile memories.	C2C/D2D variations, ADC/DAC discretization, noise, and device fluctuations.
MNSIM [203]/MNSIM2.0 [221]	Python	✓	✓	✗	Non-volatile memories.	Non-ideal device factors, interconnect parasitics.
DL-RSIM [128]	Python	✓	✗	✗	Non-volatile memories.	Non-ideal circuit and device properties.
MemTorch [113]	C++, Python	✓	✗	✗	Non-volatile memories, legacy NAND flash.	C2C/D2D variations, device failure.
TxSim [164]	Python	✓	✓	✗	Non-volatile memories, legacy NAND flash.	Interconnect parasitic, sneak-paths, and process variations.
RxNN [90]	C++	✓	✓	✓	Non-volatile memories.	Interconnects and sense parasitics, driver resistances, sneak paths, synaptic conductance variation.
NIXSim [157]	Python	✓	✗	✗	Non-volatile memories.	IR drop, SAF, programming/local/global variability, read noise.
PytorX [83]	Python	✓	✗	✓	Non-volatile memories.	IR drop, SAF, thermal noise, shot and random telegraph noise.

3.1.3 Simulation Platforms

As the landscape of memristor-based memories advances, a multitude of simulation platforms has surfaced to estimate critical system parameters including computing performance, energy consumption, and area. These platforms additionally simulate device variability, process variations, and faults to bolster the reliability of their forecasts. The comprehensive comparative analysis in Table 3.1 serves as the cornerstone for the following discussion in this section.

The majority of publications explore the influence of device variation and variability on the accuracy of Deep Neural Networks (DNNs). Notably, RxNN [90] introduces a swift and precise simulation framework leveraging a Fast Crossbar Model (FCM) that significantly accelerates the evaluation process. This platform adeptly identifies errors stemming from interconnect and sense parasitics, sneak path currents, and synaptic conductance variations during inference and training. The FCM abstracts non-idealities by generating a non-ideal conductance matrix, employing three consecutive matrix transformations to replicate synaptic device characteristics, interconnect/circuit parameters, and the chip variation profile.

Although most simulators adopt a similar approach [157, 113, 128, 164, 203, 221], PytorX [83] distinguishes itself with its comprehensive methodology for conducting end-to-end training, mapping, and evaluation, while considering a wide spectrum of reliability factors. Implemented in Python and built upon the PyTorch library, PytorX replicates MAC operations on memristive crossbars, encompassing considerations of DACs/ADCs, weight mapping, and weight partitioning on multiple arrays. To emulate the impact of the IR-drop in crossbar arrays, the authors included a dynamic crossbar solver based on the simplified modified nodal analysis. However, this approach is computationally intensive for simulating extensive neural networks, leading the authors to propose a noise injection adaption method capable of statistically approximating the effect of the IR-drop.

NeuroSim [37, 36, 154] comprises a group of instruction-accurate simulators building upon a circuit-level macro model that estimates various metrics of neuromorphic architectures, such as area, latency, dynamic energy, and leakage power. This platform facilitates a direct comparison of conventional Static Random-Access Memory (SRAM) cores, digital Emerging Non-Volatile Memory (eNVM) cores, and analog eNVM cores. In essence, NeuroSim furnishes the fundamental components to construct a comprehensive hierarchical architecture, considering not only reliability factors of the underlying memristor but also those of the implemented transistors within the crossbar array and the requisite peripherals.

Conversely, SySCIM [169] introduces a system-level simulator implemented in C++ and leveraging the SystemC and SystemC-AMS library. This approach empowers SySCIM to simulate an entire system encompassing a processor and the memristor-based computing core, setting it apart from the aforementioned frameworks. The usage of the SystemC-AMS timed data flow model enables continuous-time simulation of the crossbar, linking it to the event-driven simulation of the overall system. Furthermore, Device-to-Device (D2D) and Cycle-to-Cycle (C2C) variations are applicable

through dedicated random or systematic variation functions for the fitting parameters of the memristive device. The memristor model offers 11 fitting parameters to customize the behavior within the computing core. Alongside this detailed model, SySCIM also offers a behavioral simulation model, providing an abstract representation of the memristor. To simulate the impact of SAFs, a dedicated fault-map can be incorporated to define the probability of each memristor being faulty.

3.2 Hardware Security in the Era of Neuromorphic Computing

Machine learning applications are pivotal in addressing complex challenges, including autonomous driving and image recognition. As a result, machine learning accelerators become targets for malicious entities that seek to manipulate inference results or steal intellectual property by acquiring trained model parameters. Thus, the domain of neuromorphic computing increasingly emphasizes the importance of hardware security. This section presents an overview of existing literature on three prevalent forms of attacks: Hardware Trojans, side-channel attacks, and fault injection attacks.

3.2.1 Hardware Trojans

Nagarajan et al. [149] introduce a concept known as the Emerging NVM-based Trojan Trigger (ENTT), featuring two forms of Trojan triggers built on ReRAM technology: a delay-based ENTT and a voltage-based ENTT. These triggers are activated by repeatedly accessing a particular memory location (Ntr times).

For the delay-based variant, resistance changes resulting from persistent access to a certain address lead to increased path delays. The trigger mechanism is built around an AND gate with two inputs: *Branch T* and *Branch B*. *Branch T* is responsible for outputting an inverted signal with a prolonged ON phase, while *Branch B* delivers a standard signal that extends its ON phase after each pulse. Upon the Ntr threshold being met, *Branch B*'s signal is sufficiently delayed producing a glitch in the AND gate together with the signal of *Branch T*.

On the other hand, the voltage-based trigger utilizes a comparator to determine the gradual resistance change from targeted memory access. A trigger signal is emitted when the comparator's sensed resistance shift crosses a predefined voltage reference. The authors also note that due to the non-volatility and consequent resistance retention in ReRAM cells, sporadic access to different cells can help circumvent detection mechanisms designed to flag repeated access patterns.

Khan et al. [102] propose an NVM hardware Trojan which can be intentionally activated or deactivated. The design consists of two primary components: a trigger and a payload. The trigger exploits the high write currents typical of NVM cells by writing to a chosen address with a specific pattern, causing a ground bounce. This effect is used to incrementally charge a capacitor, activating the Trojan when a certain voltage threshold is surpassed. Khan et al. describe three distinct payloads:

- **Information Leakage:** This payload copies data from a designated memory cell to another cell controlled by the attacker. It operates by connecting the two cells via the same bit and source lines, with a transistor injected to the victim cell's word line. Activation of the transistor by the Trojan allows for the copying of data when the victim cell is accessed.

- **Read Failure:** To disturb the read process of an NVM cell, the process variation-dependent V_{clamp} voltage can be modified. Since the read circuit of an NVM cell must account for process variations, a V_{clamp} voltage generator is used to adjust the read voltage according to variations detected after manufacturing. These generators typically consist of a resistor ladder with equal resistors. Connected to a multiplexer, the circuit can calibrate the read voltage post-manufacturing. An injected circuit can intentionally alter the generator's output, causing disruption in the read process.

- **Read/Write Failure:** By introducing an additional N-Type Metal–Oxide Semiconductor (NMOS) switch, this payload can short either the bit line or the source line to ground, or connect them to a V_{disturb} to introduce noise during read/write operations.

Both studies cast light on the inadequacy of traditional detection methods, such as failure analysis tools, automatic test pattern generation, and side-channel analysis, against these NVM-specific Trojans. The negligible power consumption when inactive and their capacity to merge with standard memory operations when active make these Trojans particularly challenging to detect. To combat this threat, the authors propose a set of countermeasures including address scrambling, the use of ECC, machine learning analysis of memory images, and modulation of temperature and voltage to help identify and neutralize potential Trojans.

3.2.2 Side-Channel Attacks

Spin-Transfer Torque Random-Access Memories (STT-RAMs) are increasingly recognized as a viable alternative to SRAM-based caches, primarily due to their high density and low power consumption. Nevertheless, STT-RAMs are plagued by high and asymmetric read/write currents, which potentially enable malicious attacks aimed at information leakage. Khan et al. [99] delve into this issue by investigating a differential power analysis-based side-channel attack, specifically targeting the recovery of the secret key during an AES-128 execution. The system under attack comprises a microcontroller executing the Advanced Encryption Standard (AES) algorithm, interfaced with an STT-RAM-based Last Level Cache (LLC). The authors assume that due to the limited number of general-purpose registers, intermediate data from the cryptographic algorithm is likely stored in the LLC. Their attack methodology successfully retrieves 8 bytes of the key using a minimum of 800 traces, a notably higher number than required for SRAM, attributable to STT-RAM's lower signal-to-noise ratio.

Likewise, Wang et al. [198] explore the exploitation of power side-channels to extract the complete network architecture of DNN models. They meticulously analyze power traces across various layer types, sequences, output channels/feature sizes of convolutional and fully connected layers, and kernel sizes of convolutional layers. The introduction of a mixed-signal power simulator, with configurable hardware-level properties and a PyTorch interface for mapping pre-trained Neural Network (NN) models, is a highlight of their approach. By recording power data from both digital and analog peripherals, they create lookup tables for use during simulation. This attack model assumes the adversary's familiarity with the hardware implementation of the accelerator and control over the input and output pins of the chip, albeit without access to individual memory cells. Their investigation demonstrates the feasibility of systematically extracting all layers and reconstructing the full NN model using the proposed attack methodology.

Turning to attacks on digital Logic-in-Memory (LIM) architectures, SCARE [53] is focused on reverse engineering LIM gates using power and timing side-channels. The authors base their assumption on the execution of LIM operations over two cycles — first executing the AND gates, followed by the OR gate. Prior to extracting information, it is necessary to acquire template current profiles either through foundry-calibrated simulations or by fabricating test chips. SCARE presents two distinct attack models: the first is most effective when the inputs to the Boolean function are direct, while the second, more generic model requires extensive reverse engineering efforts but do not necessitate access to the direct inputs. This research demonstrates the ability of SCARE to reveal critical details about the implemented Boolean functions and highlights its potential in compromising real-world implementations.

3.2.3 Fault Injection Attacks

The Rowhammer attack on Dynamic Random-Access Memories (DRAMs) has revealed a significant vulnerability by enabling the intentional injection of faults into computing and memory systems. Khan et al. [100] further explore this issue, examining the susceptibility of STT-RAMs to row hammering attacks. Their research focuses on how ground bounce, induced by high write currents, can compromise the integrity of STT-RAMs. This effect reduces the thermal energy barrier of memory bit cells, making them more prone to retention failures, magnetic field interferences, and thermal noise. A key finding of their study is that persistent writing (hammering) to a specific memory location can substantially weaken the thermal barrier of nearby unselected bits, resulting in bit flips. Moreover, the study indicates that ground bounce can impact bit-line and source-line drivers, negatively affecting the performance of selected cells by reducing the headroom voltage. Simulations in their research demonstrate that row hammering attacks can modify bits in STT-RAM within approximately 30.84 s, with the risk increasing at higher temperatures.

The same authors also investigate fault injection attacks on ReRAM-based caches, specifically for initiating Denial-of-Service (DoS) attacks [101]. They consider a 1-Transistor 1-Resistor (1T1R) ReRAM-based LLC used by both the victim and the at-

tacker. An attacker can write specific data patterns to produce deterministic supply noise in their memory space, facilitating DoS attacks or targeted polarity fault injection attacks on the shared memory space. The simulations show that attackers can launch DoS attacks by injecting over 120 mV of supply noise at the victim's write location, and a polarity fault injection attack with noise levels above 50 mV but below 120 mV. To counter these attacks, the authors suggest design-level countermeasures such as sequential read/write access, high-quality power/ground grids, and separate power rails for each bank.

Li et al. [121] present two innovative hardware attacks, Variation-oriented Adversarial Attack (VADER) and Enhanced Fault Injection Attack (EFI), exploiting the variability of ReRAMs. They aim to target a neuromorphic DNN accelerator based on memristive crossbar arrays. VADER employs a variation amplification algorithm to manipulate variation-sensitive pixels in input images, bypassing conventional defense mechanisms. This algorithm selects pixels in input samples that can magnify the effects of ReRAM crossbar variations, while limiting the number of affected pixels to conceal the manipulations. Conversely, EFI leverages ReRAM variations for covert and efficient low-cost fault injection attacks. It exploits variation-induced deviations in weight parameters to deliberately cause misclassification in specific sample categories by the NN model, minimizing the number of targeted weights to evade detection. Both attacks have demonstrated high success rates, underscoring the vulnerability of RRAM-based computing systems to hardware-aware adversarial attacks.

3.3 Instrumention Platforms

Due to the immature state of memristive devices, various testing and instrumentation platforms have been proposed to evaluate single devices and crossbar structures.

Berdan et al. [20] developed the memristor Characterization And Testing (mCAT) platform, designed to redistribute sneak-path currents within the crossbar array, significantly improving measurement accuracy. This platform provides all required potentials to the passive memristive crossbar through a multiplexer array. The bias generator applies the appropriate $V/2$ potentials to all unselected lines, while the sense bank aims to minimize measurement errors. Notably, the platform employs five different sense resistances, conducting 50 consecutive read operations with each resistance to diminish noise. Tested on a 32×32 discrete resistive crossbar array and solid-state TiO2-x ReRAM arrays, the mCAT platform demonstrated measurement accuracy with less than 1% error for standalone memristive devices and less than 10% error for 90% of devices in a custom resistive crossbar. Its versatility, enhanced by an NXP mBED microcontroller and a MATLAB Graphical User Interface (GUI), allows for seamless integration with ReRAM crossbar arrays. The mCAT platform, distributed commercially by ArC Instruments Ltd. under the name ArC ONE measurement system [17], features open-source software for the microcontroller and Python software for the host PC, including predefined test routines and data visualizations.

The mCAT system, however, is constrained by its interface to facilitate only passive (selectorless) memristive crossbar arrays. Addressing this, Foster et al. [65, 64] proposed an improved version centered around an Field-Programmable Gate Array (FPGA) EFM-03 development board. This system includes a 64-channel Source-Meter Unit (SMU) and two 32-pin banks for digital I/O, achieving a current noise floor of 170 pA, pulse delivery of ±13.5 V, and a maximum current drive of 12 mA per channel. The SMU channel, a key subsystem, incorporates a programmable gain Transimpedance Amplifier (TIA), a high-speed independent pulse generator, and a switch for current source access. The TIA functions as both a source and a meter, with differential ADCs for voltage reading. Digital terminals include a selector bank with 32 digital outputs and an arbitrary level logic bank supporting a wide voltage range. This system, capable of controlling up to a 32×32 selectorless crossbar or a 21×21 array with transistor selectors, is available from ArC Instruments Ltd. as the ArC TWO measurement system [17].

Kaya et al. [98] introduced another FPGA-based measurement system, focusing on 1T1R ReRAM structures. This system stands out for its straightforward and independent design, avoiding complex carrier modules. It addresses key ReRAM parameters like switching voltages, resistance states, data retention, and endurance. Based on a XILINX Artix 7 series FPGA board (AC701), it features a 5-channel arbitrary waveform generator and voltage buffers. The system supports both current and voltage-based resistance measurements, with a maximum output of 10 mA. The proposed platform utilizes a 32-Bit Microblaze soft processor with FreeRTOS, enabling flexible application development. The implemented software facilitates various memory operations and is complemented by a MATLAB GUI for cell operations and resistance variability analysis, useful for developing security applications like Random Number Generations (RNGs) and Physical Unclonable Functions (PUFs).

Additionally, Tektronix, Inc. offers the 4200A-SCS parameter analyzer, a premier system for materials, semiconductor devices, and process development. It boasts an impressive measurement resolution of up to 100 fA and 0.2 µV, specializing in I-V, C-V, and ultra-fast pulsed I-V measurements.

However, for crossbar measurements, the 4200A-SCS is not suitable. To address this, aixACCT Systems GmbH developed the aixMATRIX, a comprehensive matrix test system for simultaneous stimulation of test structures on 64 analog channels, each with a 16-bit Digital-to-Analog Converter (DAC) and a sample rate of 100 MS/ms. Capable of generating bipolar signals up to ±10 V and featuring ultrafast, bipolar adjustable current limiting, this system excels in analyzing memristive memories with response times below 50 ns. It can investigate both active and passive memory arrays up to 32×32 cells and is integrated into a 200 mm wafer prober, facilitating advanced research in neuromorphic memory systems [3].

3.4 Lessons Learned

The postulation of the memristor launched an avalanche of research to investigate these devices from the material stack to the system level performance and energy efficiency. The merit of works has focused on the analog Computing-in-Memory (CIM) paradigm which promises to drastically outperform conventional von Neumann architectures. However, specifically this operational mode significantly suffers from the high variability of the memristive devices limiting the scope of real-world applications. Consequently, a thorough investigation of the impact of reliability issues on LIM operations is missing. While neuromorphic systems are gaining momentum, an increasing number of publications prove the existence of hardware security vulnerabilities stemming from the unique characteristics of the underlying memristive device. This novel attack surface represents a significant threat to future neuromorphic computing systems and to the application executed on them. To investigate concerns like the reliability and hardware security threats, instrumentation platforms have been proposed offering different measurement accuracies, in/output channels, and characterization routines. Nevertheless, the vast majority of publications bases their investigation on simulated devices failing to reproduce the impact of variability and reliability concerns of memristive devices. To the best of our knowledge, there is currently no instrumentation platform capable of performing analog nor digital operations on memristive crossbar arrays with the aim to determine the impact of these concerns on their computational result.

Therefore, in this thesis, we aim to provide a comprehensive analysis of reliability and hardware security concerns of neuromorphic systems. Our work aims to add crucial findings to usher the way towards trustworthy neuromorphic computing. Alongside other contributions, in the rest of this thesis, we introduce the following:

1. **Fault Injection Platform:** A fault injection platform for LIM operations allowing a comprehensive examination of reliability, ranging from the individual memristor level up to complete, real-world workloads. This platform enables a comparative analysis of various logic families regarding their resilience to faults and supports detailed investigations at the application level to identify the parameters that most critically affect potential workloads.

2. **NeuroHammer**: This represents a novel hardware security attack, enabling adversaries to deliberately manipulate bits in ReRAMs. NeuroHammer poses significant challenges to the integrity of neuromorphic systems.

3. **NeuroBreakoutBoard**: An innovative instrumentation platform designed to facilitate CIM operations on actual memristive crossbar arrays. It provides valuable insights into how reliability issues can affect CIM operations.

3.5 Synopsis

This chapter presents a detailed survey of existing research in neuromorphic computing, focusing on the reliability and hardware security of memristor-based systems. It explores the challenges in ensuring the reliability of ReRAMs, including strategies for fault detection and error correction. The chapter also discusses various simulation platforms that model system performance and reliability, accounting for device variability and faults. A significant portion is dedicated to hardware security, examining vulnerabilities to hardware Trojans, side-channel, and fault injection attacks in neuromorphic systems. Additionally, it reviews different instrumentation platforms for testing and evaluating memristive devices and crossbar structures.

Chapter 4

Fault Injection in Logic-in-Memory Architectures

As discussed in Chapter 2, there are two fundamental ways to perform computation with memristive crossbar arrays. Analog Computing-in-Memory (CIM) leverages the continuous resistance values of memristive devices to perform Multiply–Accumulate (MAC) operations in the analog domain. While offering exceptional performance through extensive parallel computation, analog CIM copes with inherent limitations, such as the need for Digital-to-Analog Converters (DACs)/Analog-to-Digital Converters (ADCs) for value conversion between digital and analog domains. On the contrary, Logic-in-Memory (LIM) is another flavor, executing logic functions within memristive crossbar arrays by strictly utilizing the memristive devices in a binary manner. This approach may reduce computing performance compared to analog CIM but promises enhanced reliability without requiring conversion between the analog and digital domains. However, as concluded in Chapter 3, the reliability of LIM architectures remains largely unexplored, with the majority of research focusing on analog CIM. LIM's reliance on logic families to implement LIM gates introduces an additional layer of complexity and uncertainty by potentially utilizing a faulty memristor multiple times within a single logic gate. Consequently, modeling the reliability of such operations requires a fundamentally different approach that accounts for the memristive devices, the logic family, and the executed application. Therefore, this chapter introduces a comprehensive fault injection framework for LIM operations, capable of investigating reliability from the memristor level to actual full-fledged workloads. The framework includes a memristor-level fault injection simulator named **X-Fault** and an operational-level simulator called **Faulty Logic-in-Memory (FLIM)**. The former excels at emulating the impact of faults on individual logic gates with high precision, while the latter offers exceptional simulation speed for executing realistic workloads using abstracted fault models. Together, these simulators allow for a comparison of logic families in terms of fault resilience and enable an investigation at the application level to understand which parameters most significantly influence potential workloads.

This chapter is organized as follows: Section 4.1 provides an overview of the framework. The process of application mapping is discussed in Section 4.2, while Sections 4.3 outlines the crossbar simulator and its fault models. Delving into the internals of FLIM, Section 4.4 elaborates on the fault generation mechanism, and Section 4.5 discusses the implementation of the fault injection method. Comprehensive case studies presented in Section 4.7 underscore the significance of our fault injection framework. Finally, in Section 4.8, we discuss the limitations and future directions of this work. This chapter summarizes the contributions presented in [180, 175, 176].

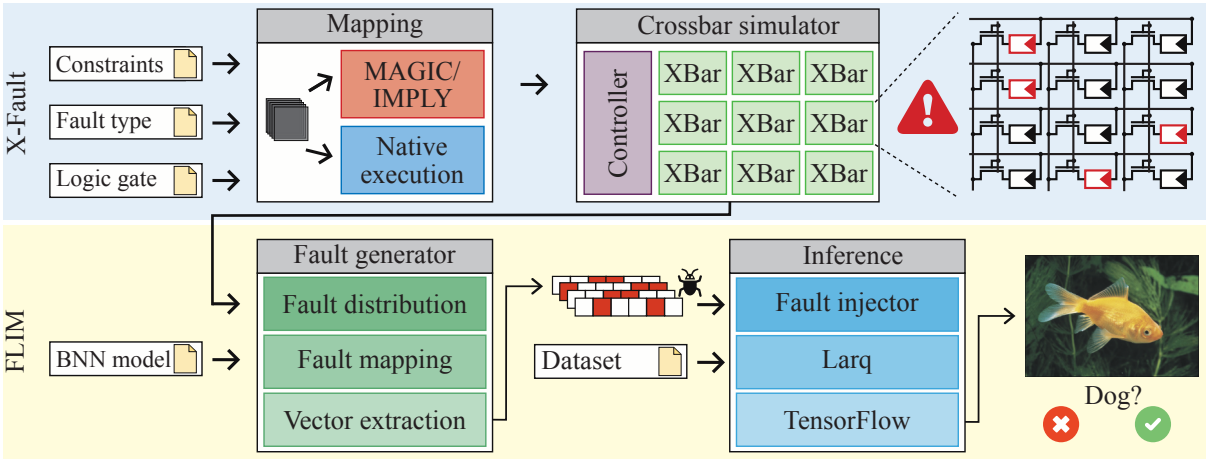


Figure 4.1: Overview of the fault injection framework featuring X-Fault and FLIM: X-Fault provides a mapping tool and crossbar simulator for highly accurate simulations that assess the resilience of logic families. In contrast, FLIM employs a more abstracted approach with its fault generator and fault injector, offering a platform for high-speed simulation.

4.1 Framework Overview

An overview of the fault injection framework is depicted in Figure 4.1. This framework consists of two separate modules, X-Fault and FLIM, specifically designed for the execution of Binary Neural Networks (BNNs), which predominantly utilize binary XNOR operations. X-Fault requires inputs such as hardware constraints, fault type, and the type of logic gate. The hardware constraints specify the dimensions of the crossbar array and the mapping algorithm, while the fault type identifies the fault model and its parameters, namely, injection rate and fault pattern. The logic gate parameter not only determines the gate itself but also the corresponding logic family. In general, X-Fault comprises a mapping tool and a crossbar model. The mapping tool can interpret Larq [69] models and relay the relevant XNOR operations to the crossbar model while all other operations are executed within TensorFlow [1]. The crossbar simulator incorporates a comprehensive memory controller which orchestrates the instantiated crossbar arrays. Closely linked with the crossbar model is the binary memristor model, which implements various fault models that can be parameterized to accommodate a wide array of experiments.

The second component of the simulation framework, FLIM, employs an abstracted fault injection methodology aimed at achieving high simulation speed. This simulator requires a BNN model based on the Larq framework. The necessary fault distribution can be sourced from X-Fault or input manually. To significantly boost simulation speed, FLIM transfers the labor-intensive tasks to a preprocessing phase. The fault generator embodies these preprocessing steps, comprising three distinct phases: fault distribution, mapping, and vector extraction. The resultant noise vector serves as an input for the fault injector, which is deeply integrated within the TensorFlow/Larq environment.

Algorithm 4.1: Weight stationary XNOR mapping on memristive crossbar arrays

Data: crossbar instance (*crossbar*), kernel instance (*kernel*), input data instance (*input*), FlatBuffer handler instance (*flatbuffer_handler*)
Result: Mapped XNOR operations on crossbar array

```
1  nr_kernel_values ← kernel.nr_values_per_kernel × kernel.nr_kernels;
2  for i ← 0 to kernel.kernel_iterations() do
3      kernel_to_write ← kernel.get_kernel_index(i,nr_xnor_gates);
4      crossbar.write_kernel(kernel_to_write, flatbuffer_handler);
5      kernel_start ← i × nr_xnor_gates;
6      kernel_end ← (i + 1) × nr_xnor_gates;
7      if kernel_end > nr_kernel_values then
8          kernel_end ← nr_kernel_values;
9      end
10     (i_values, i_indices) ← get_input(kernel_start, kernel_end);
11     for y ← 0 to input.get_size()[0] − kernel.get_size()[0] +1 do
12         for x ← 0 to input.get_size()[1] − kernel.get_size()[1] +1 do
13             crossbar_input ← empty list;
14             for each pair (input, input_loc) in zip(i_values, i_indices) do
15                 input_list ← list containing int(input);
16                 y_p ← input_loc[1] + y;
17                 x_p ← input_loc[0] + x;
18                 crossbar_input_element ← list containing input_list, y_p, x_p;
19                 append crossbar_input_element to crossbar_input;
20             end
21             crossbar.write_input_to_crossbar(crossbar_input, flatbuffer_handler);
22             crossbar.calculate_xnor(flatbuffer_handler);
23         end
24     end
25 end
```

4.2 Application Mapping

X-Fault's mapping tool adeptly maps convolutional and dense layers of a BNN onto a crossbar of a predefined size. The mapping tool extracts the dimensions, as well as the respective values of the kernel and inputs, to generate a set of write, read, and logic instructions. Central to the mapping algorithm is its focus on minimizing the number of instructions. This is achieved by efficiently tracking partial fits of the kernel within the remaining cells of a crossbar array. The mapper facilitates communication with the crossbar simulator through a binary file interface, leveraging the FlatBuffers library [71] which offers an efficient and cross-platform serialization methodology. The binary file encapsulates the memory addresses of the kernels, their corresponding values, and the requisite logic operations.

At its core, the algorithm adopts a weight stationary mapping, wherein the weights (kernels) of the BNN are sequentially mapped onto the crossbar array and remain stationary throughout the computation process. While the simulator does not account for weight drift effects, it is noteworthy that depending on the logic family used, some input cells may be overwritten during the execution of logic gates, necessitating a rewrite of weights. Algorithm 4.1 shows the implemented mapping methodology, which is detailed in the following.

Initially, the algorithm calculates the total number of values in the kernel matrix and processes these values in segments iteratively. In each iteration, a particular kernel segment, determined by the number of XNOR gates in the crossbar, is selected and stored in the binary file. These segments are computed based on the total number of kernels multiplied by the number of values per kernel.

Next, the algorithm identifies the starting and ending points of the kernel within the input matrix. This step is critical for accurately aligning the kernel with the input data for effective XNOR operations. To achieve this, nested loops are utilized to traverse the dimensions of the input matrix, tailored to the size of the kernel. Within these loops, the algorithm linearly maps the input values to a one-dimensional list object, corresponding to the kernel values. This structured input list is then written to the binary file, aligned with the kernel parameters. Upon successful mapping of inputs and kernels, the mapping tool issues a compute command, activating the crossbar simulator to execute the XNOR operations as per the instructions in the binary file. This algorithmic design not only maximizes the parallel computational capacity of the crossbar array but also aligns with the constraints of the implemented LIM families.

4.3 Crossbar Simulator

The crossbar simulator follows the hierarchy of Dynamic Random-Access Memory (DRAM) consisting of channels, ranks, banks, rows, and columns. Internally, the simulator implements this hierarchy in three distinct entities: memory controller, crossbar, and memristor. In the following, the functionality of each entity is described in detail.

4.3.1 Memory Controller

The memory controller parses the provided binary file from the mapping tool to extract the required kernels and input values. Furthermore, the controller is responsible for orchestrating the underlying crossbars by issuing the instructions to the respective banks. The overall dimensions of the memory can be customized via a configuration file which determines the number of ranks, banks, and the dimensions of the crossbar arrays. The memory address translation and allocation employs two different interleaving methods (page or rank interleave) to manage how physical addresses are mapped to memory channels, ranks, banks, rows, and columns. This flexibility al-

Table 4.1: Overview of logic gate implementations based on IMPLY and MAGIC logic families including the number of memristors (#mem) and cycles (#cycles) required for each operation.

Logic family	Logic gate	#mem	#cycles	Internal structure
IMPLY	IMP	2	1	IMP(1,2)
	NOT	2	2	FILL(2,HRS)→IMP(1,2)
	NAND	3	3	FILL(3,HRS)→IMP(2,3)→IMP(1,3)
	AND	3	4	NAND(1,2,3)→NOT(3,1)
	OR	3	3	FILL(3,HRS)→IMP(1,3)→IMP(3,1)
	NOR	3	5	OR(1,2,3)→NOT(2,1)
	XOR	4	5	IMP(1,3)→IMP(4,2)→NOT(2,4)→IMP(3,4)
	XNOR	4	6	XOR(1,2,3,4)→NOT(4,1)
MAGIC	NIMP	3	1	NIMP(1,2,3)
	NOR	3	1	NOR(1,2,3)
	OR	3	1	OR(1,2,3)
	XOR	3	3	FILL(3,HRS)→NIMP(1,2,3)→NIMP(2,1,3)
	XNOR	4	6	FILL(4,LRS)→FILL(3,HRS)→NIMP(1,2,3) →NIMP(2,1,3)→FILL(1,HRS)→NIMP(4,3,1)
	AND	5	9	OR(1,2,3)→FILL(4,HRS)→NIMP(1,2,4) →FILL(5,HRS)→NIMP(3,4,5)→FILL(4,HRS) →NIMP(2,1,4)→FILL(1,HRS)→NIMP(5,4,1)
	NAND	5	12	AND(1,2,3,4,5)→FILL(2,HRS)→FILL(3,LRS) →NIMP(3,1,2)

lows for a more comprehensive exploration of memory management strategies in the presence of faults. Additionally, the implementation takes into account the bit-level manipulation of data, foster the unique requirements of LIM.

4.3.2 Crossbar Model

The crossbar model is designed to simulate a binary memristive crossbar array, with a particular focus on the implications of in-field faults within the system. It abstracts the interactions between memristive cells, aiming to balance the accuracy of simulation with the need for high simulation speed. Each cell within the array is instantiated through the memristor model.

Initialization of the crossbar is executed by parsing a configuration file that determines the crossbar's dimensions, the proportion of memristors in the Low Resistive State (LRS) versus the High Resistive State (HRS), the prevalence of faulty cells, and the characteristics of faults, including the type and pattern. To assign initial states

and fault models to cells, the crossbar model employs a uniformly distributed random number generator. The distribution of these random values is given by the probability density function:

$$P(x|a,b) = \frac{1}{b-a} \tag{4.1}$$

Here, x represents the random value uniformly distributed over the interval $[a,b)$. In addition to cell initialization, the crossbar constructs inter-cell relationships to account for coupling faults. Read and write operations are conducted on a per-row basis. While the model incorporates standard read and write methods, it also features a specialized interface to support access based on the $V/2$ scheme, as discussed in Section 2.1.2.

Table 4.1 presents a comprehensive overview of the logic gates implemented within the simulation framework. Both the Memristor-Based Material Implication (IMPLY) and Memristor-Aided Logic (MAGIC) logic families offer only a fundamental set of logic gates. To construct more complex Boolean functions, the simulator combines these basic gates, which, as a result, significantly increases the number of required cycles and memristors, as detailed in Table 4.1. To streamline the implementation process of these logic gates, we introduce a utility function as follows:

Definition 4.1. The `FILL(row, value)` function assigns a specified *value* (either HRS or LRS) to a given *row*.

The implemented memory controller mandates that read and write operations are executed row-wise. Therefore, logic gates are mapped vertically within the crossbar array. As discussed in Section 2.2.2, the execution of a single logic operation triggers computation across all available columns, resulting in substantial parallelization. The internal structure shown in Table 4.1 refers to the rows using unique integer numbers ranging from $[1,n]$ with $n \in \mathbb{N}$ where n denotes the total number of rows of the crossbar. For example, the NAND gate constructed using the IMPLY logic family requires three memristors in three adjacent rows. Initially, the third row is set to HRS. Subsequently, Material Implication (IMP) operations are executed between the second and third rows, and then between the first and third rows, resulting in the output of the NAND gate, which utilizes three computational cycles/steps.

4.3.3 Memristor Model

The lowest level of X-Fault's memory hierarchy is represented by the memristor model which encapsulates the essential characteristics and functionalities of a memristive device. The abstracted model tracks the utilization by accumulating the read, write, and $V/2$ accesses. The simulator implements the MAGIC and IMPLY logic family because of their sole use of the memristive crossbar without any required peripherals, except the required resistor for the IMPLY family which we omit in our investigations. Furthermore, the memristor model utilizes an action log to keep a history of executed instructions for each cell. This history is required to facilitate dynamic faults which occur every n-th operation to be sensitized [77]. The memristor model only

Algorithm 4.2: Coupling fault write/read disturb algorithm

Data: pattern length (*pattern_len*), coupling pattern (*pattern*), memristor pointers (*east*, *west*, *north*, *south*)
Result: state of the memristor cell after evaluating the coupling fault condition

```
1  for i ← 0 to pattern_len − 1 do
2      it_ptr ← this;
3      for j ← 0 to |pattern[i].x| - 1 do
4          if pattern[i].x ≥ 0 then
5              it_ptr->east ≠ NULL? (it_ptr ← it_ptr->east) : exit;
6              it_ptr->west ≠ NULL? (it_ptr ← it_ptr->west) : exit;
7          end
8      end
9      for j ← 0 to |pattern[i].y| - 1 do
10         if pattern[i].y ≥ 0 then
11             it_ptr->north ≠ NULL? (it_ptr ← it_ptr->north) : exit;
12             it_ptr->south ≠ NULL? (it_ptr ← it_ptr->south) : exit;
13         end
14     end
15     if it_ptr->get_state() ≠ pattern[i].aggressor_state then
16         exit;
17     end
18 end
19 this->state ← (this->state = LRS ? HRS : LRS);
```

implements the traditional fault models outlined in Section 2.3.3, as the unique fault models require more precise simulations due to their analog characteristics. The fault models are randomly assigned to a certain percentage of the instantiated memristors and are enumerated as follows:

1. **Stuck-at-Fault (SAF)** is characterized by the memristor adopting a constant resistive value, manifesting either as a HRS or an LRS.

2. **Read-Destructive-Fault (RDF)** inverts the current state of the memristive cell while still returning a correct value.

3. **Deceptive Read Destructive Fault (DRDF)** modifies the current state of the cell and yields an incorrect value, misleading the read operation.

4. **Incorrect Read Fault (IRF)** causes the cell to remain unchanged, but the value retrieved is incorrect.

5. **Slow Write Fault (SWF)** occurs when a write operation to the cell fails, resulting in the preservation of the cell's previous state.

6. **Coupling Fault (CF)** occur when a write operation to a cell results in an unintended write operation to an adjacent cell.

Table 4.2: Comparison of fault models in X-Fault and FLIM: the fault models handled by X-Fault with their corresponding abstracted representations in FLIM illustrates the differences in fault handling between the two simulators and emphasizes the trade-off between simulation accuracy and speed.

X-Fault	**FLIM**
Stuck-at fault	Stuck-at mask
Read destructive fault	Bit-flip mask
Deceptive read destructive fault	
Incorrect read fault	
Slow write fault	
Coupling fault	
Dynamic faults	Repeated bit-flip mask

Algorithm 4.2 illustrates the function used to evaluate the state of a cell in the presence of potential coupling faults induced by surrounding aggressors. The algorithm iteratively examines each cell based on a defined coupling pattern, utilizing directional pointers (east, west, north, south) to navigate the crossbar array. If the pattern is absent or the target cell's neighbors do not match the aggressor state, the algorithm terminates prematurely, indicating that no coupling fault will occur. However, if the aggressor pattern is detected, the state of the target memristor is toggled between the LRS and HRS, simulating the read/write disturb fault.

4.4 Fault Generator

As discussed in Section 4.1, the FLIM simulator is designed to significantly enhance the simulation speed by abstracting the fault injection methodology. Specifically, this simulator separates the computationally intensive aspects of the fault injection, performing them offline, while the core fault injection process is integrated with the inference phase. This preprocessing stage, referred to as the fault generator, encompasses three primary steps: fault distribution, fault mapping, and noise vector extraction.

4.4.1 Fault Distribution

In contrast to X-Fault, FLIM introduces faults at the level of XNOR operations to enhance simulation performance, as depicted in Table 4.2. This abstraction level prohibits the ability to simulate single-cell accesses, thereby limiting the emulation of complex fault models. FLIM, therefore, restricts the supported fault models to primarily stuck-at and bit-flip faults.

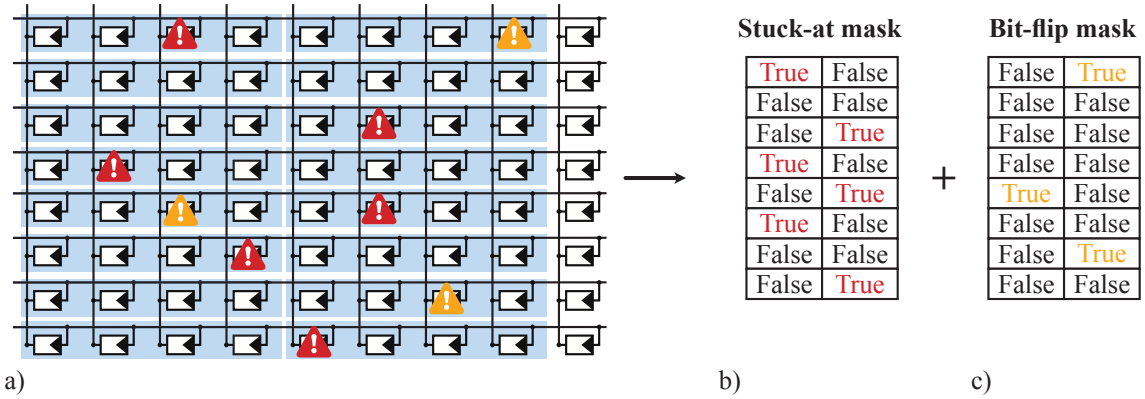


Figure 4.2: Overview of the implemented fault mapping: detailed correlation between the presumed faulty memristive devices and the resulting stuck-at and bit-flip masks.

In FLIM, the emulation of a crossbar array is bypassed in favor of annotating the inference process with fault masks. Figure 4.2 illustrates the correlation between randomly assigned values in fault masks and the potential fault distribution within a crossbar array. This is illustrated in Figure 4.2 (a), where in-field faults lead to malfunctioning XNOR operations. The FLIM fault masks, as shown in Figure 4.2 (b-c), denote the locations of these malfunctions and are mapped onto the workload during the fault mapping stage.

The bit-flip mask is a two-dimensional Boolean array initially filled with zeros. The array's fault distribution, dictated by the injection rate, is represented by assigning a corresponding number of elements to one. FLIM also allows for the analysis of faults impacting entire rows or columns, stemming from potential address decoder issues. In these cases, the affected row or column in the bit-flip mask is entirely marked as one. FLIM is capable of handling dynamic faults, which necessitate the duplication of the fault mask across multiple layers. This replication involves constructing a sequence of bit-flip masks, each sequentially applied to different layers during the inference stage. Similarly, the stuck-at mask is represented as a two-dimensional Boolean array, initialized with zeros and stuck-at faults marked with ones.

This pre-processing method of generating masks significantly enhances performance by shifting the computationally intensive tasks of mapping and distributing faults away from the actual process of inference.

4.4.2 Fault Mapping

In the next phase, the masks previously generated are allocated to designated layers within the BNN model. For this purpose, the framework requires information about the dimensions and the total number of crossbars assumed to be utilized in the hardware accelerator. Initially, the mapping tool computes the number of concurrent XNOR operations based on the number of crossbars. As shown in Table 4.1, the IM-

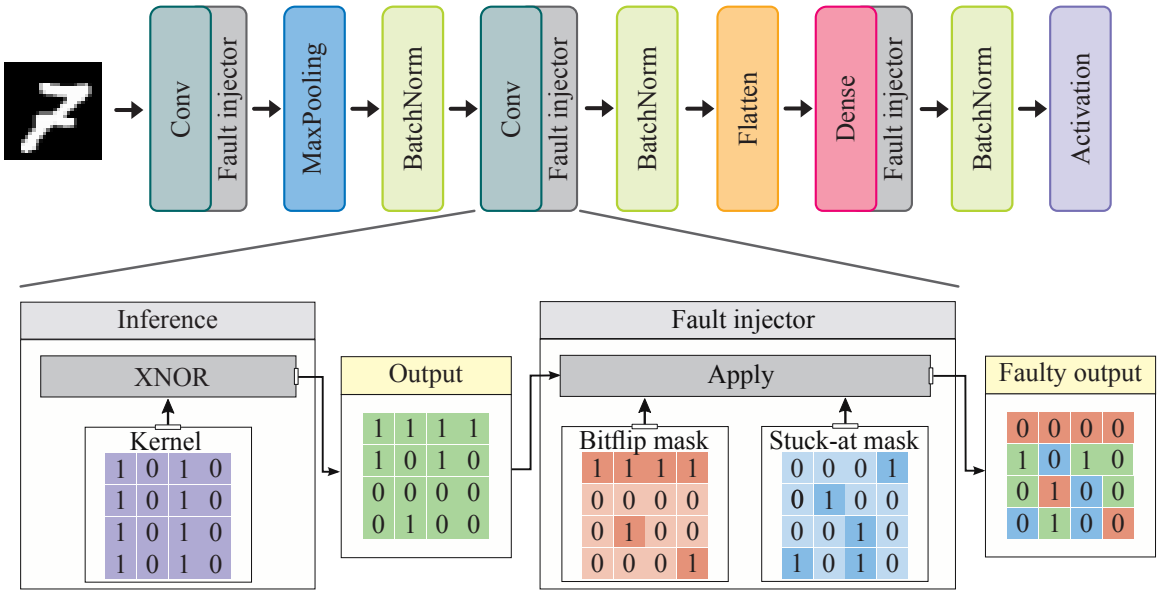


Figure 4.3: Overview of FLIM's fault injection methodology composed of the inference stage and the fault injector.

PLY and MAGIC XNOR operation requires a total of four memristors representing the dominant Boolean function in BNNs.

Next, the tool extracts the number of required XNOR operations based on the provided model. Due to the structure of the BNN layers, XNOR operations are dominantly utilized in the 2-dimensional convolution layers and fully binarized dense layers. Therefore, these specific layers are targeted for mapping onto memristive crossbar arrays to leverage acceleration, while other layers continue to be processed using conventional Complementary Metal-Oxide Semiconductor (CMOS) technology. Hence, the mapping tool extracts the dimensions of these layers and assigns the previously generated fault masks.

4.4.3 Noise Vector Extraction

Finally, the generated fault masks are transformed in a 1-dimensional representation and stored in a binary file together with meta information about the target layer, the in-layer location, and the potential dynamic fault behavior. The binary file is independent of the dataset and allows for a parallel simulation of different system parameters to facilitate, for example, a reliability assessment of an architecture exploration.

4.5 Fault Injector

The Fault Injector stands as the core component of the FLIM platform, seamlessly integrated within the Larq and TensorFlow frameworks to achieve optimal performance through precise fault injection. Larq, essentially an extension of the Keras frame-

work [40], is specifically designed to facilitate BNNs by introducing custom quantized layers that enhance the capabilities of standard Keras layers. In this context, we have augmented the base class of these layers by incorporating an instance of the Fault Injector. This integration allows the activation of the fault injection mechanism during the inference process, necessitating a modification of the original convolution method. Figure 4.3 visually depicts FLIM's fault injection approach. As elaborated in Section 4.4, FLIM is capable of injecting faults in both convolutional and dense layers. The process begins with the computation of the convolution between the feature map and the kernel, which initially remains fault-free, thereby yielding a correct computation outcome. Following this, the bit-flip and stuck-at masks are applied to the output through an additional XNOR operation, effectively integrating the predetermined faults based on the injection rate. The subsequent subsections provide a comprehensive exploration of the layer specific implementation of the fault injection mechanism.

4.5.1 Fault Injection in Conv2D Layers

The Larq library implements the `Quant2D` layer by taking advantage of the TensorFlow `tf.nn.conv2d()` function which performs a convolution of the input feature map and the given kernel. This function is invoked from the `convolution_op()` function within the Conv2D layer which acts as the entry point for FLIM's fault injector. The Fault Injector provides a custom convolution method extending the `tf.nn.conv2d()` function by injecting bit-flip and stuck-at faults.

Initially, the custom convolution function preprocesses the input feature map, tailoring its shape to align with the fault masks. This preprocessing is crucial because when a kernel is applied to the input, it typically results in the reduction of the spatial dimensions (width and height) of the output feature map. Such a reduction, particularly in deep networks, can lead to a significant decrease in input size and the potential loss of edge information. To mitigate this issue, padding techniques are employed to add extra pixels around the input image's border. This step ensures that the kernel can effectively process the border pixels, thereby preserving the input's spatial dimensions. Consequently, two padding methods are implemented, each of which is detailed in the following:

VALID padding reduces the size of the output feature map to omit the necessity to add additional values around the actual input which proves beneficial to reduce the spatial dimensions of the feature maps. Consequently, the kernel only iterates over a subset of pixels resulting in a reduced dimension of the output. The resultant output width O_w and height O_h is defined as

$$O_w = \frac{I_w - K_w}{s_w} + 1 \tag{4.2}$$

$$O_h = \frac{I_h - K_h}{s_h} + 1 \tag{4.3}$$

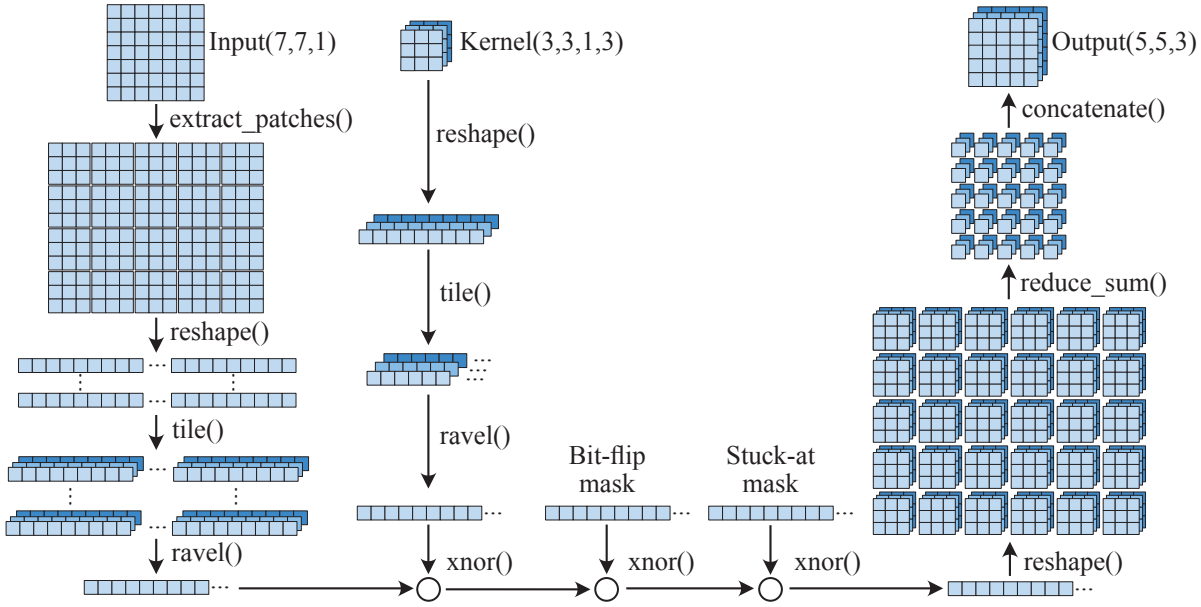


Figure 4.4: Overview of the convolutional layer preprocessing including binary XNOR operation with fault injection and tensor aggregation.

where I_w, I_h denotes the input data dimensions, K_w, K_h denotes the kernel dimensions, and s_w, s_h the width/height of the stride, respectively [51].

SAME padding adds additional pixels around the edges of the input image so that the output feature map matches the dimensions of the input data. The resultant output width O_w and height O_h is defined as

$$O_w = \frac{I_w + 2p - K_w}{s_w} + 1 \tag{4.4}$$

$$O_h = \frac{I_h + 2p - K_h}{s_h} + 1 \tag{4.5}$$

where I_w, I_h denotes the input data dimensions, K_w, K_h denotes the kernel dimensions, s_w, s_h the width/height of the stride, and p represents the number of zeros added along both axis [51].

Figure 4.4 illustrates the preprocessing required for the input image and the associated kernel in a convolutional layer. Patches corresponding to the kernel's size are extracted from the padded input and prepared for the XNOR operation with the kernel. Both the patches and the kernel are reshaped—flattened and duplicated—to enable the binary convolution. Following the reshaping, bit-flip and stuck-at masks are applied through additional XNOR operations, producing a feature map that incorporates the intended faults. This output is subsequently reshaped back to the form it would take after a standard, non-binary convolution. To finalize the output, TensorFlow's `reduce_sum` function aggregates values across a designated dimension, streamlining the tensor's structure. Each aggregated value thus forms a part of the concatenated output, with each contributing to the overall feature representation.

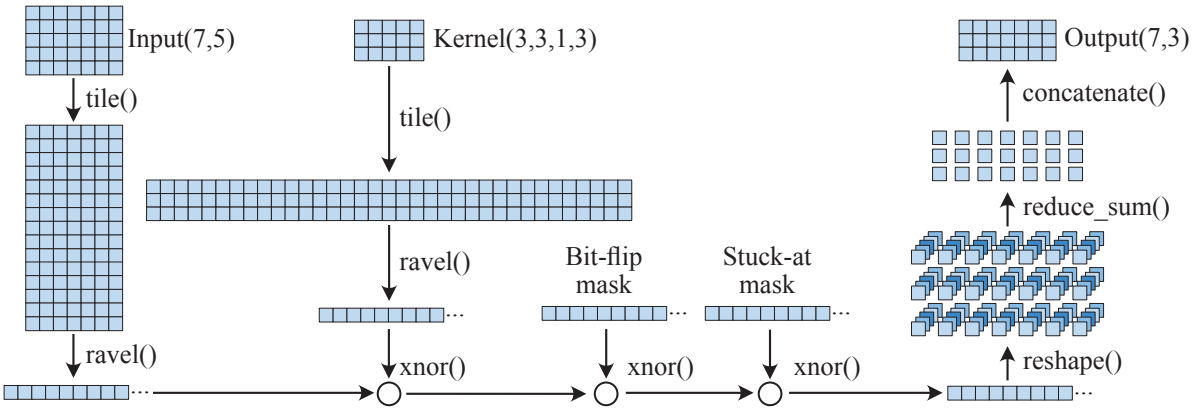


Figure 4.5: Overview of the dense layer preprocessing including binary XNOR operation with fault injection and tensor aggregation.

4.5.2 Fault Injection in Dense Layers

In comparison to convolution layers, dense layers, also known as fully connected layers, operate by performing a matrix multiplication between the input features and the weights of the neurons, followed by the addition of a bias term. Mathematically, the output $\mathbf{o}$ of a dense layer can be expressed as:

$$\mathbf{o} = f(\mathbf{W} \cdot \mathbf{x} + \mathbf{b}) \tag{4.6}$$

where $\mathbf{o}$ is the output vector, $\mathbf{W}$ is the weight matrix, $\mathbf{x}$ is the input vector, $\mathbf{b}$ is the bias vector, and f denotes the activation function. Since dense layers operate on a flattened input where the spatial structure is not preserved, padding is not required. Therefore, the input preprocessing of dense layers is simplified as shown in Figure 4.5. First, the input, initially in a 2-dimensional form, is expanded by the `tile()` function, which replicates the tensor to match the kernel's dimensions. This is followed by `ravel()`, which flattens the expanded tensor into a 1-dimensional array. The kernel undergoes a similar `tile()` and `ravel()` process. Subsequently, both the input and kernel maps are subject to a XNOR operation resulting in the output feature map. Subsequently, the bit-flip and stuck-at masks are applied introducing faults in the layer. After the fault injection, the resultant feature map is reshaped, preparing it for the subsequent layer or output.

4.6 Resilience Metric for Logic-in-Memory Families

In the context of LIM architectures, a diverse set of logic families has been proposed, each with its unique characteristics and operational efficiencies. However, a critical aspect that often remains ambiguous is their resilience to faults, which is a key characteristic in terms of the real-world feasibility. This is particularly crucial as memristive devices, the backbone of these architectures, are inherently prone to various in-field

faults. To address this gap, a standardized metric is essential for quantitatively assessing the resilience of these logic families and their individual gates. Our simulation framework, as detailed in Chapter 4, lays the groundwork for an in-depth analysis of the reliability of LIM gates. However, it lacks a universal metric that can effectively measure and compare the fault tolerance across different logic families. Such a metric would not only facilitate a more comprehensive understanding of each family's robustness but also aid in identifying optimal logic families for specific applications. Furthermore, it opens up the possibility of strategically mixing gates from different logic families to optimize for power, area, or latency, which can significantly enhance the overall performance and efficiency of neuromorphic systems. Consequently, we propose two distinct metrics: one that evaluates the overall resilience of a logic family and another that focuses on the resilience of individual gates within a family.

Definition 4.2. The *Quality of Logic (QoL)* is defined for a single fault model as

$$\mathrm{QoL} = \sum_{i=0}^{G-1} \frac{\Lambda}{\Omega} \cdot 100\%, \tag{4.7}$$

where G represents the number of gate types, Λ denotes the total number of faulty outputs, and Ω signifies the total number of outputs. The QoL metric provides an indication of how well the entire set of supported logic gates performs under the influence of a specific fault type.

Definition 4.3. The *Impact of Fault (IoF)* is defined for a single gate type as

$$\mathrm{IoF} = \sum_{i=0}^{F-1} \frac{\Lambda}{\Omega} \cdot 100\%, \tag{4.8}$$

where F is the number of fault types, Λ denotes the total number of faulty outputs, and Ω signifies the total number of outputs. This metric is particularly insightful as it reveals the impact of all fault types on a single logic gate's functionality.

By applying these metrics, we can systematically evaluate and compare the fault resilience of different logic families and individual gates, providing a comprehensive understanding of their robustness and reliability in real-world applications. This approach not only enhances our understanding of the vulnerability of these systems to in-field faults but also guides the design and optimization of neuromorphic computing systems for enhanced performance and reliability.

4.7 Evaluation

In this section, we present a detailed evaluation of in-field faults on LIM architectures, while also discussing the trade-off between simulation accuracy and speed through a comparative analysis of X-Fault and FLIM. All experiments were carried out on a dedicated workstation equipped with an AMD Ryzen 7 5800X processor and a DDR4

	SAF	RDF	DRDF	IRF	IoF
AND	42%	42%	39%	42%	**38%**
IMP	38%	25%	50%	50%	**34%**
NAND	33%	25%	58%	42%	**34%**
NOR	42%	42%	33%	42%	**37%**
NOT	50%	50%	75%	50%	**48%**
OR	33%	25%	42%	42%	**31%**
XNOR	44%	38%	44%	50%	**41%**
QoL	**40%**	**35%**	**49%**	**45%**	

(a)

	SAF	RDF	DRDF	IRF	IoF
AND	35%	25%	35%	43%	**31%**
NIMP	33%	29%	67%	38%	**34%**
NAND	45%	28%	60%	60%	**45%**
NOR	44%	38%	33%	42%	**36%**
XOR	50%	75%	50%	38%	**46%**
OR	34%	27%	40%	40%	**30%**
XNOR	44%	38%	44%	50%	**40%**
QoL	**41%**	**37%**	**47%**	**44%**	

(b)

Figure 4.6: Fault resilience of logic families: (a) assessment of the IMPLY and (b) MAGIC.

2666 MHz 64 GB main memory. To accelerate the simulation process, FLIM leverages the computational power of an NVIDIA GeForce RTX 3080 Ti with 12 GB of memory. Both simulators, X-Fault and FLIM, are based on a modified version of Larq 0.12.0 which utilizes TensorFlow 2.8.0 for their operations.

In the following, we delve into two case studies aimed at assessing the resilience of the LIM paradigm at varying levels of abstraction. Initially, the robustness of different logic families is exhibited through a series of comprehensive simulations using X-Fault. This is followed by an investigation of the reliability of neuromorphic applications accelerated by LIM, taking into account a variety of parameters including diverse fault models, types of layers, and BNN models.

4.7.1 Case Study: Resilience of Logic Families

To assess the resilience of various logic families, X-Fault's crossbar model is utilized to conduct an exhaustive analysis of both MAGIC and IMPLY logic families, along with their respective logic gates. The output Z of a defective logic gate depends on the input values $(x,y) = \{(x,y) \mid x,y \in \{1,0\}\}$, the chosen fault model $f \in \{\text{SA}, \text{RDF}, \text{DRDF}, \text{IRF}\}$, the number of memristors $M \in \mathbb{N}$, and the fault location $l = \{m_n \mid n \in \mathbb{N}, n \leq M\}$. To evaluate resilience, we simulated each logic gate under various fault models, considering all possible combinations of input values, initial states of memristive devices, and fault locations. We then compared the output to that of a fault-free gate to determine the percentage of incorrect outputs. The proportion of faulty outputs for all logic gates in both the MAGIC and IMPLY families is depicted in Figure 4.6. It's important to note that coupling faults were not included in this

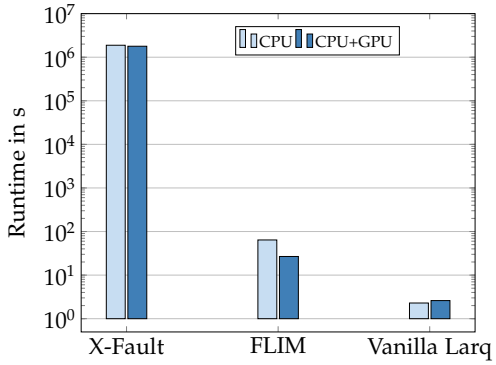


Figure 4.7: Performance evaluation of the fault injection platform: running a pre-trained binarized LeNet model on the MNIST dataset with FLIM and vanilla Larq, and X-Fault.

series of experiments, as they necessitate multiple consecutive accesses. Additionally, the corresponding QoL and IoF metrics are calculated and presented in Figure 4.6.

Noteworthy, the OR gate shows the highest resilience in both families, with an IoF of 31% for IMPLY and 30% for MAGIC. The QoL metric suggests that RDF has the least influence on logic gates, whereas DRDF significantly impacts them. Moreover, the NOT gate in the IMPLY family and the XOR gate in the MAGIC family rank as the least resilient in terms of fault tolerance. Overall, our experiments indicate comparable performance between the two families, as reflected in the similar QoL and IoF values. Nevertheless, architectural design decisions can be optimized by considering resilience towards specific fault models. For example, a designer might prefer the IMPLY implementation of the NAND gate, with an IoF of 34%, over the MAGIC version, which has an IoF of 45%.

4.7.2 Case Study: Resilience of Binary Neural Networks (BNNs)

Considering the previous findings in Section 4.7.1, it is crucial to gain an understanding about the impact of faults on the application level to be able to assess the reliability of the LIM paradigm. Given LIM's binary nature, BNNs are an ideal application to be accelerated with LIM operations, particularly due to their reliance on XNOR operations for inference computation. To simulate comprehensive BNN models, this case study employs the FLIM simulator. As FLIM prioritizes simulation speed at the expense of simulation accuracy, our experiments are restricted to the injection of stuck-at and bit-flip faults. To demonstrate the trade-off between X-Fault and FLIM, we benchmark both simulators, presenting a comparison of their respective performance levels. Furthermore, this study explores the impact of faults on two distinct types of layers (dense and convolutional) and their positions within the network. Finally, a range of BNN models are simulated to assess the influence of different model designs on fault impact.

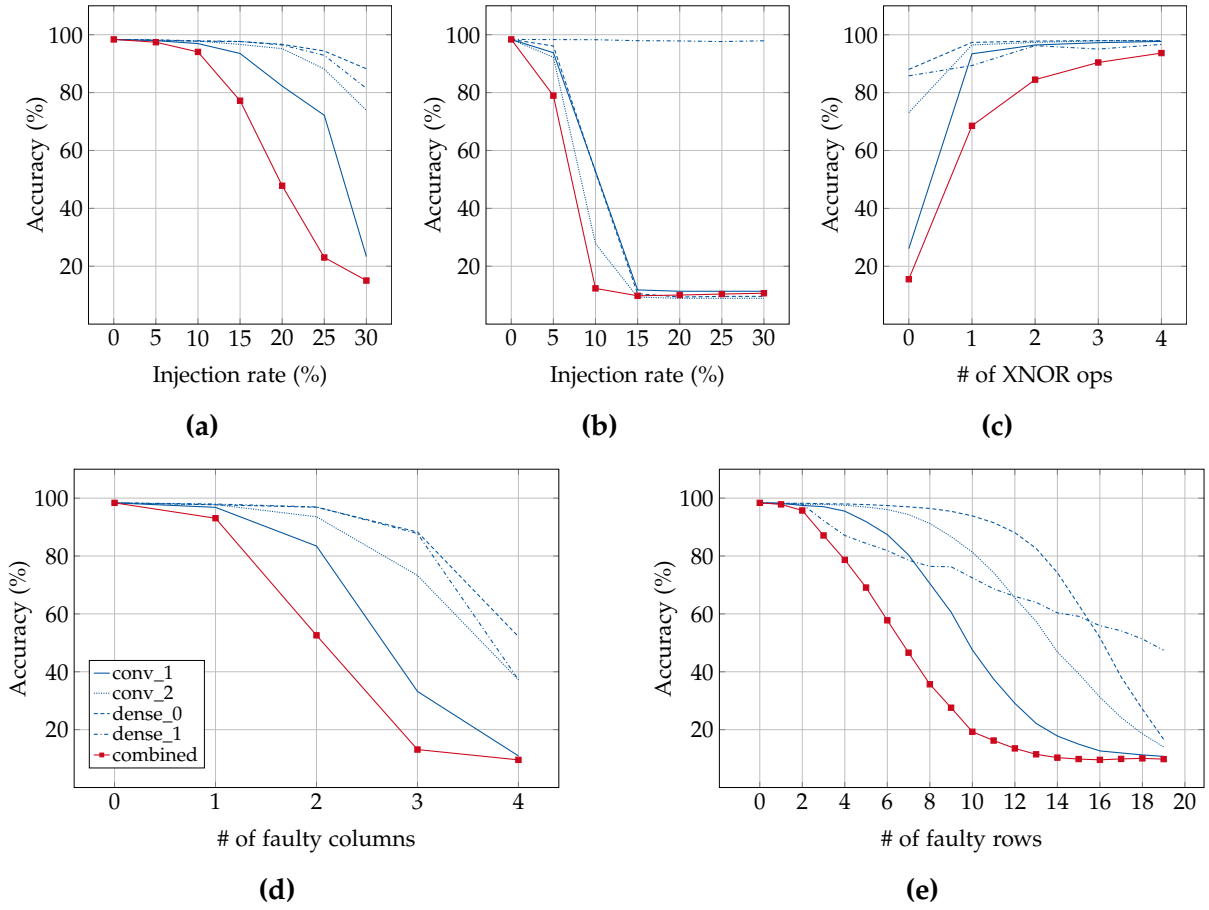


Figure 4.8: Simulation results: impact of (a) bit-flips, (b) stuck-at, (c) dynamic faults, (d) faulty columns, and (e) faulty rows on different layers.

4.7.2.1 Performance Evaluation

To evaluate the simulation performance of our fault injection platform, we run the inference of a pretrained binarized LeNet model on the entire MNIST test datatest composed of 10,000 images. While LeNet [116] is a convolutional neural network with three convolutional layers and two dense layers, MNIST [49] resembles a large database of handwritten digits in 28×28 pixel image. FLIM and the vanilla Larq implementation were each run fifty times on the full dataset, while the total runtime for X-Fault was extrapolated based on the analysis of just five images due to the extensive simulation runtime. During these inferences, the fault injection mechanism mapped the respective operations on a 40×10 crossbar array but did not actively inject faults, positioning the vanilla Larq implementation as a baseline for comparison in terms of simulation time.

As depicted in Figure 4.7, FLIM significantly outperforms X-Fault. *Notably, FLIM processes the 10,000 images approximately 29,375 times faster than X-Fault.* Furthermore, leveraging the benefits of GPU acceleration through deep integration with Larq and TensorFlow, FLIM achieves a staggering *speed-up of about 66,754 times relative to X-*

Table 4.3: Summary of BNN models and their associated parameters [69].

Model	Top-1 Acc.	Size	Parameters	MACs	Binarized
RealToBinaryNet [142]	65.0%	5.13MB	12M	1.81B	92.39%
BinaryDenseNet45 [22]	65.0%	7.54MB	13.9M	6.67B	96.34%
BinaryDenseNet37 [22]	62.9%	5.25MB	8.7M	4.71B	96.76%
BinaryDenseNet28 [22]	60.9%	4.12MB	5.13M	3.79B	94.66%
BinaryResNetE18 [82]	58.3%	4.03MB	11.7M	1.81B	92.4%
BinaryAlexNet [108]	36.3%	7.49MB	61.8M	841M	91.34%
MeliusNet22Z[21]	62.9%	3.88MB	6.94M	4.76B	97.14%
Bi-Real Net [136]	57.5%	4.03MB	11.7M	1.81B	92.4%
XNORNet [160]	45.0%	22.81MB	62.4M	1.14B	90.05%

Fault. In summary, by abstracting the fault model to the XNOR operation level, FLIM strikingly balances simulation accuracy with a remarkable boost in performance.

4.7.2.2 Layer Resilience

This experiment is designed to explore the effects of various faults on different layers and their locations within a BNN model. Utilizing the same pretrained binarized LeNet model as in the previous Section 4.7.2.1, which achieves an accuracy of 97.62%, we employ the MNIST test dataset comprising $10,000$ images of 28×28 pixels. Each layer is allocated to a 40×10 crossbar array for this purpose. Subsequently, we inject bit-flip, stuck-at, and dynamic faults into different crossbars corresponding to their respective layers, while adjusting the fault injection rate. To address the randomness of fault placement on the crossbars, we conducted each test run a hundred times.

The simulation outcomes are depicted in Figure 4.8, where individual layer traces are illustrated in blue, and the red trace represents the overall accuracy impact when faults are uniformly injected across all layers. *Figures 4.8 (a-b) demonstrate that stuck-at faults significantly degrade accuracy more than bit-flip faults, irrespective of layer type. Stuck-at faults consistently affect all layers, whereas bit-flip faults' impact on accuracy depends on the layer's depth, with convolutional layers being more susceptible than dense layers.* Figure 4.8 (c) examines dynamic bit-flip faults, indicating the number of XNOR operations required for fault activation. *This analysis shows that the BNN model's accuracy typically stabilizes back to its original value after around four consecutive XNOR operations.*

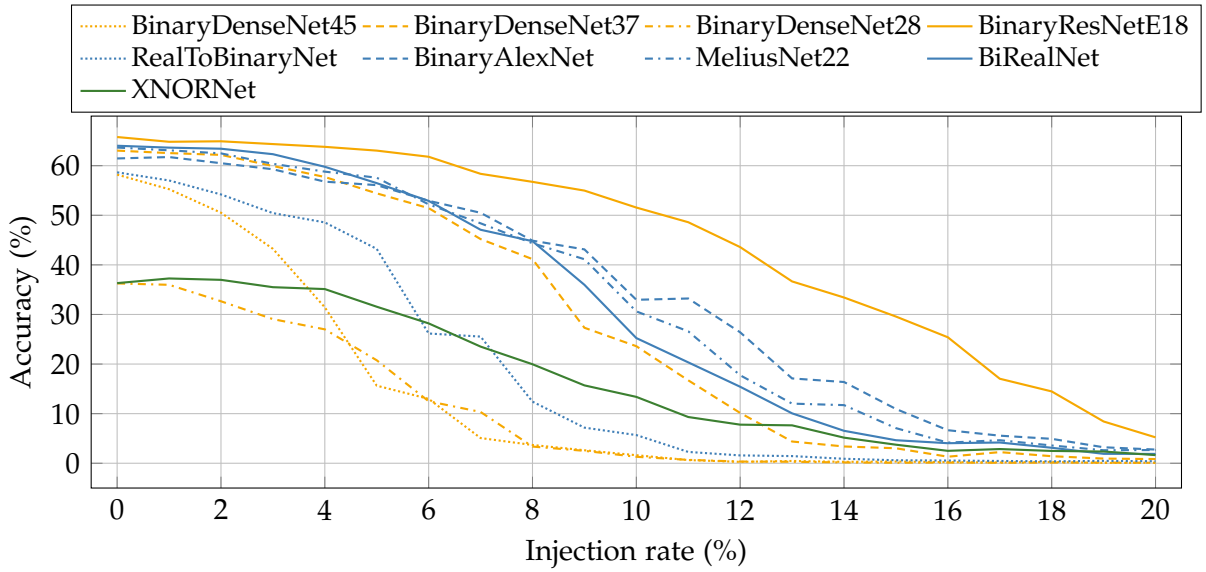
Moreover, the data suggest that the layer's depth has a direct correlation with its impact on accuracy, especially highlighting a nearly linear reduction in the performance of the final dense layer, as illustrated in Figures 4.8 (d-e). *Overall, faulty columns tend to have a more significant impact than faulty rows, which seems plausible due to the column-wise parallel execution of XNOR operations.*

4.7.2.3 Model Resilience

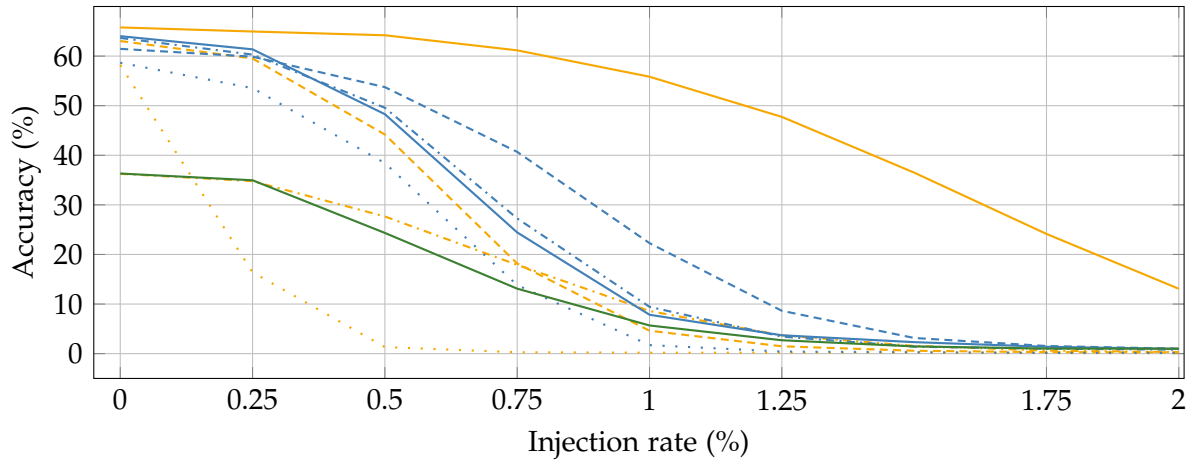
Table 4.3 lists the BNN models used in this study to examine the influence of the BNN model architecture on fault resilience. These models have been pretrained using the ImageNet dataset [48], into which both (dynamic) bit-flips and stuck-at faults were subsequently injected. To mitigate the effects of randomness from the random number generator, each inference run was repeated a hundred times.

The previous experiment established that stuck-at faults significantly reduce the accuracy more than bit-flip faults. *Figure 4.9 (a-b) proves this finding across different BNN models, demonstrating that the impact is consistent independent of the model. Furthermore, the majority of models return to their initial accuracy levels after about three consecutive XNOR operations when dynamic faults are introduced as illustrated in Figure 4.9 (c).*

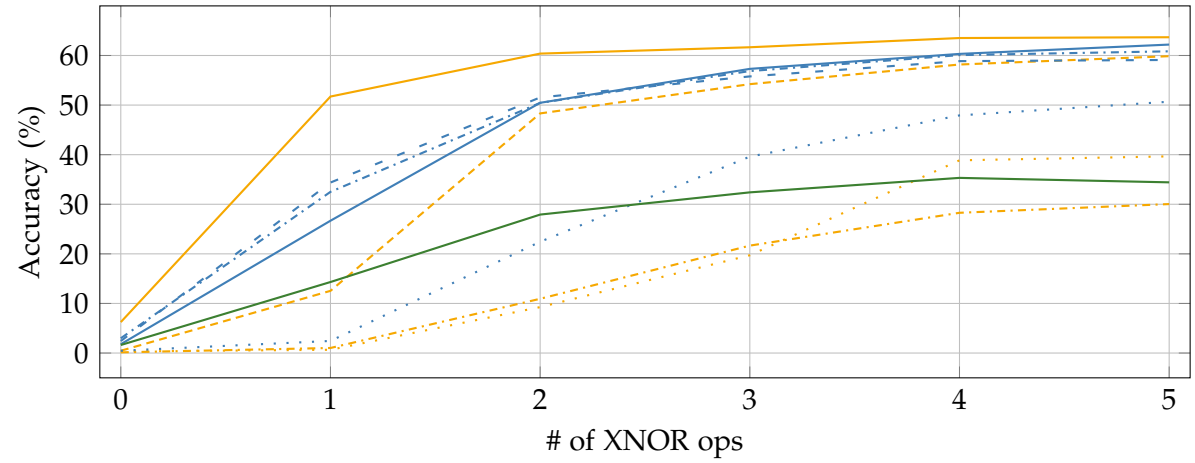
Overall, it becomes evident that time-dependent fault variations influence the reliability of neuromorphic applications in diverse ways. Depending on the fault injection rate, transient faults impact the applications' reliability to different degrees. Likewise, the findings suggest that the durability of these emerging applications is predominantly jeopardized by permanent faults, like stuck-at faults. BiRealNet and XNOR-Net present unique cases due to their non-strict binarization approach in convolutions. BiRealNet employs real-valued activation functions through identity shortcuts [136], while XNOR-Net applies a channel-wise gain to the weights, reflecting each channel's magnitude. Despite these differences, FLIM successfully simulates both models with minor modifications to the bit-flip mask.



(a)



(b)



(c)

Figure 4.9: Simulation results of (a) bit-flips, (b) stuck-at, and (c) dynamic faults on different models.

4.8 Limitations and Outlook

The fault injection framework introduced in this chapter has proven to be an instrumental tool for assessing the resilience of the LIM paradigm, spanning from the logic level to the application-level. The utilization of both X-Fault and FLIM offers a comprehensive strategy, assisting designers in selecting suitable logic families and validating their choices through the execution of full-fledged applications.

Nevertheless, X-Fault's performance bottleneck restricts its capacity to explore the effects of faults across extensive sets of operations. As a result, simulating only a single layer may be practical with X-Fault to keep simulation times within acceptable limits. Likewise, FLIM delivers the necessary performance for fault injection in realistic BNN models but falls short in terms of the supported fault models. Additionally, X-Fault's design limits it to logic families that rely exclusively on memristive crossbar arrays without additional peripheral circuitry. Given that most logic families require some form of external peripherals, this limitation limits X-Fault's compatibility with a wider array of logic families. In terms of FLIM, addressing the effects of in-field faults during inference is essential for understanding the practical viability of the LIM paradigm. Considering the challenges posed by the non-ideal behaviors of memristive devices, on-device training emerges as a promising solution. Therefore, expanding FLIM to include fault injection capabilities during the training phase would significantly broaden the range of applications for this fault injection framework, offering deeper insights and more robust solutions for the implementation of the LIM paradigm.

4.9 Synopsis

This chapter introduced a fault injection framework for assessing the resilience of LIM architectures. The framework consists of two simulation tools, X-Fault for memristor-level emulation and FLIM for high-speed operational simulation, to explore fault resilience across logic families and neuromorphic applications. Figure 4.1 illustrates the interaction between the two simulation platforms. X-Fault is used to assess the fault distribution, while FLIM evaluates its impact at the application level. Through comprehensive case studies, the framework evaluates the impact of faults on various logic gates and BNN models, highlighting the trade-offs between simulation accuracy and speed. A novel metric is proposed for quantifying the fault resilience of logic families and gates, aiming to enhance the understanding of their robustness against in-field faults. The chapter identifies limitations, such as X-Fault's performance bottleneck and FLIM's restricted fault model support, and suggests future enhancements, including extending FLIM to support fault injection during training. This chapter significantly advances the understanding of LIM's viability and guides the optimization of neuromorphic systems for improved reliability and efficiency.

Chapter 5

Deliberately Flipping Bits in Memristive Crossbar Arrays

Hardware security represents a crucial aspect of modern computing systems, often overshadowed by the emphasis on software vulnerabilities. Unlike their software counterparts that can be mitigated through updates, hardware security flaws are embedded within the physical circuitry, making them almost impossible to correct without replacing the chip entirely. This intrinsic challenge poses a profound threat to the integrity and reliability of computing systems, as seen by the Meltdown [130] and Spectre [106] attacks.

In Chapter 4, we have highlighted the susceptibility of neuromorphic computing, especially the Logic-in-Memory (LIM) paradigm, to faults that drastically undermine the reliability of applications running on neuromorphic hardware. This chapter builds upon these results by introducing a novel hardware security attack termed NeuroHammer which threatens the integrity of the entire system. This attack specifically targets the memristive crossbar arrays utilized in neuromorphic computing systems, intentionally inducing bit-flip faults to undermine the foundational principle of modern computing systems—memory separation. By exploiting the distinct properties of memristive devices to alter the switching kinetics through thermal crosstalk, NeuroHammer unveils an attack surface, similar to the Rowhammer attack in Dynamic Random-Access Memories (DRAMs) (see Section 2.4.3). This chapter begins by outlining the fundamental attack scenario and the working principles underlying NeuroHammer in Section 5.1. To assess the feasibility and scope of the NeuroHammer attack, Section 5.2 and Section 5.3 present a simulation methodology comprising both a thermal and a circuit simulation. This dual-simulation approach enables a realistic emulation of thermal crosstalk within memristive crossbar arrays, providing insights into the conditions under which the NeuroHammer attack can be most effectively executed. Section 5.4 discusses a comprehensive set of experiments aim to evaluate the impact of various parameters on the feasibility of the NeuroHammer attack. Through these experiments, we aim to determine the critical factors that influence the vulnerability of neuromorphic computing systems to this novel hardware security threat.

To underline the real-world implications of NeuroHammer, Section 5.5 provides a case study which showcases the leakage of an Rivest–Shamir–Adleman (RSA) key from a computing system utilizing memristive memory. This case study underscores the seriousness of this security threat and its capacity to compromise integrity of sensitive information processed by neuromorphic computing systems. The chapter concludes with a discussion on NeuroHammer's limitations and future directions in Section 5.6. This chapter summarizes the contributions presented in [178, 174].

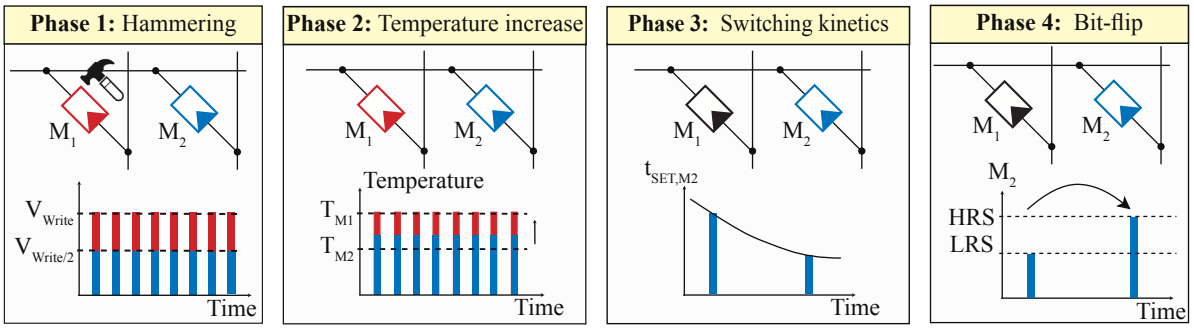


Figure 5.1: Working principles of NeuroHammer consisting of four stages: hammering, temperature increase, switching kinetics, and, finally, the intended bit-flip.

5.1 NeuroHammer

The existence of the NeuroHammer attack in passive crossbar arrays stems from two fundamental observations. Firstly, Von Witzleben et al. [195] investigate the impact of elevated temperatures on the switching kinetics of Resistive Random-Access Memories (ReRAMs), with a particular emphasis on Valence Change Material (VCM) that utilize a transition metal oxide. By employing a nanometer-scale heating structure, the authors were able to achieve a rapid temperature increase within the cell and performed kinetic measurements at different temperatures to observe the effect on the switching times and pre-SET slope. The study revealed that higher temperatures significantly decrease the SET times which also aligns with simulations results based on an analytical model.

Secondly, as discussed in Section 2.1.2, passive crossbar arrays necessitate some form of isolation mechanism to safeguard half-selected cells from unintended switching. The $V/2$ scheme, the most commonly adopted technique, applies a voltage drop of V_{write} across the designated device, while the non-selected devices along the word and bit lines experience an absolute voltage of $|V_{\text{write}}/2|$. Within the context of NeuroHammer, this scheme ensures that in passive crossbars, adjacent cells are exposed to a $|V_{\text{write}}/2|$ pulse with each write operation targeting the selected cell.

Merging these observations, Figure 5.1 showcases the intrinsic workings of the NeuroHammer attack, represented through four discrete phases. The attack and targeted cells are represented by red and blue cells, respectively. In the following is a comprehensive and detailed account of each phase:

1. **Hammering:** Ideally, the red cell is initially in a Low Resistive State (LRS) to maximize current flow. The attacker repetitively writes to this cell, inducing a V_{write} pulse across its terminals. Owing to the $V/2$ scheme, the blue cell endures repeated stress from the generated $V/2$ scheme.

2. **Temperature increase:** The continuous V_{write} pulses lead to a temporary rise in the red cell's temperature, which in turn increases the temperature of the adja-

cent blue cell due to thermal crosstalk. Concurrently, the blue cell experiences regular $|V_{\text{write}}/2|$ voltage pulses, further increasing its temperature.

3. **Switching kinetics:** As demonstrated by Von Witzleben et al. [195], increased temperatures alter the switching kinetics, reducing the SET time. Consequently, the device is more susceptible to gradually change its resistance.

4. **Bit-flip:** Eventually, the blue cell alters its internal state after a gradual change over time. The combination of thermal crosstalk and the $V/2$ scheme enables the attacker to flip a bit without direct access to the targeted cell.

This attack procedure requires a $|V_{\text{write}}/2|$ across the target cell's terminals to eventually provoke a bit-flip. However, the $V/2$ scheme introduces sneak-path currents, which constrain the crossbar size, lower the read accuracy, and increase power dissipation (see Section 2.1.2). Hence, 1-Transistor 1-Resistor (1T1R) structures emerge as a promising solution, leveraging transistors to insulate half-selected cells from the read/write pulses.

In the following, we validate the existence of the NeuroHammer attack on both passive and active crossbar arrays by utilizing our simulation methodology to assess the impact of thermal crosstalk. Initially, a thermal simulation confirms that thermal crosstalk sufficiently increases the temperature of adjacent cells. Additionally, the simulator yields thermal coupling coefficients, termed alpha values, for emulating crosstalk at the circuit level. A subsequent circuit-level simulation employs these alpha values to simulate thermal crosstalk, considering the electrical properties of the memristive devices and the crossbar structure. This simulation methodology not only provides key insights about the influence of thermal crosstalk in dense passive/active crossbar structures but also manifests the impact of the NeuroHammer attack in neuromorphic computing.

5.2 Thermal Simulation

In this section, we describe the implementation of a memristive crossbar model within the COMSOL Multiphysics® [45] simulation platform, aimed at quantifying thermal crosstalk between adjacent memristive cells. The simulation leverages the finite element method to determine heat transfer coefficients, referred to as *alpha values*, which are subsequently used as inputs for the circuit-level simulation (refer to Section 5.3). Section 5.2.1 details the procedural steps undertaken to evaluate thermal crosstalk originating from an individual device. Moreover, Section 5.2.3 elaborates on extending this methodology to analyze more advanced patterns involving multiple devices.

5.2.1 Memristive Crossbar Model

Figure 5.2 (a) depicts the crossbar model employed for thermal simulations. This model consists of Bottom Electrode (BE) and Top Electrode (TE), which intersect perpendicularly on a Si/SiO_2 substrate. Unlike the `JART VCM v1b` compact model [19]

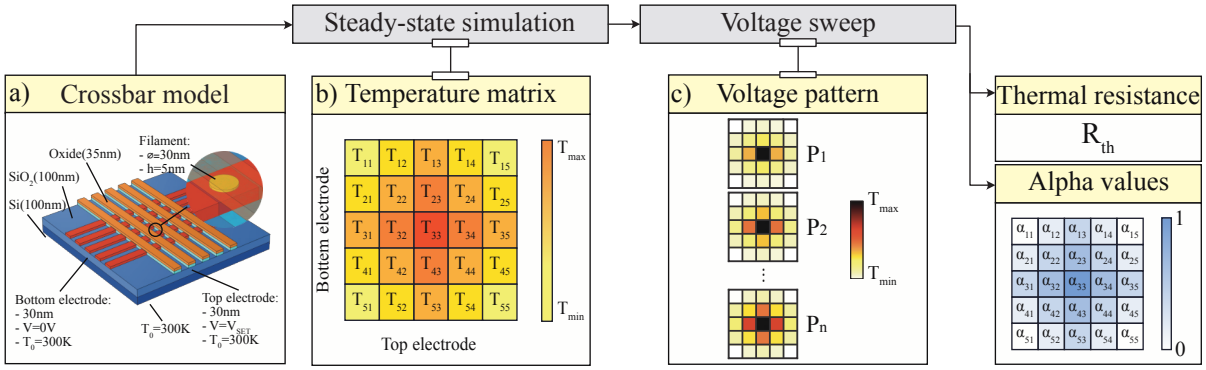


Figure 5.2: Overview of the thermal simulation methodology: (a) depiction of the crossbar model alongside its boundary conditions, (b) extraction of device temperatures within the crossbar array, and (c) assessment of thermal resistance for a centered cell and the corresponding alpha values for the adjacent devices.

used in circuit simulations (refer to Section 5.3), the memristive devices placed between the TE and BE in this model are represented solely by their filament. This approach omits the division into a resistive switching disc area and a highly conductive plug region, which significantly enhances simulation performance.

The resistance of each memristive device is thus characterized by the filament's resistance, which can be adjusted as needed. The temperature of the ReRAM cells is calculated by solving the static heat transfer equation

$$-\nabla \cdot (\kappa \nabla T) = \boldsymbol{j} \cdot \boldsymbol{E}, \tag{5.1}$$

and the current continuity equation

$$\nabla \cdot \boldsymbol{j} = -\nabla \cdot (\sigma \nabla \phi) = 0, \tag{5.2}$$

where T represents the temperature of the memory cell, ϕ the electrical potential, κ the thermal conductivity, σ the electrical conductivity, $\boldsymbol{j}$ the local current density, and $\boldsymbol{E}$ the electric field.

In this setup, the memristive device does not simulate any switching characteristics, meaning the heat generated by a device is determined purely by its dissipated power. Consequently, the electrical conductivity of the filament is assumed to remain constant throughout the simulation. To establish a specific current I through the cell, we manually adjust the filamentary resistor's electrical conductivity R_{fil} in accordance with Ohm's law $V_{\text{SET}} = R_{\text{fil}} I$. Figure 5.3 (a-b) illustrates the electrical and thermal model of the memristive cell, respectively.

The thermal boundary conditions are defined by the top surface of the substrate and the crossbar structure, both acting as thermal insulators. Additionally, the contacts of the electrodes and the simulation model's bottom surface are set to the ambient temperature T_0, serving as an ideal heat sink. Electrical currents are directed into or out of the memristive array exclusively via the top and bottom electrodes, with the electrode potentials set to V, $V/2$, or 0 V.

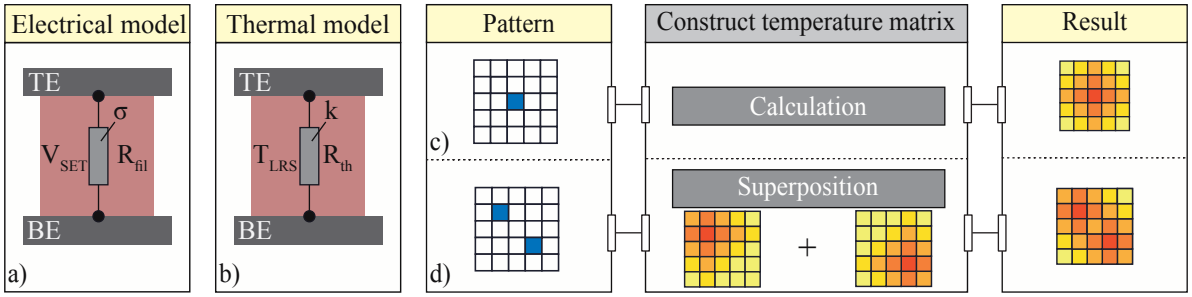


Figure 5.3: (a) Electrical circuit diagram representing the modeled memristive device within the crossbar model and (b) its corresponding equivalent thermal diagram. (c) The temperature matrix featuring a single selected cell can be directly calculated. (d) For configurations with two selected cells, the resultant temperature matrix emerges from the superposition of two shifted temperature matrices.

5.2.2 Thermal Crosstalk—Single Device

Initially, we assess the thermal crosstalk generated by a single memristive cell. To isolate the effect, we assume that only the targeted cell is in an LRS, while all other cells remain in a High Resistive State (HRS), thus minimizing the influence of self-heating from adjacent half-selected cells. Our investigation centers on the thermal crosstalk that occurs during the write process of a cell in the LRS, which maximizes the current through the targeted device and consequently, the resultant thermal crosstalk. In the context of a passive memristive crossbar array, we employ the $V/2$ scheme to selectively address the targeted cell.

Figure 5.2 (b) presents the temperature matrix derived from a steady-state simulation of the specified crossbar model. Each element T_{ij} within the temperature matrix indicates the peak temperature of the filament at its respective position in the array. To obtain the necessary alpha values and the associated thermal resistance R_{th} of the targeted cell, we execute a voltage sweep of V_{SET}, yielding various temperature matrices. Based on Fourier's law, the temperature rise across a thermal resistor directly correlates with the heat generated from the dissipated power. Hence, we can calculate the thermal resistance R_{th} through linear regression between the dissipated power $P_{LRS} = V_{SET} I$ and the temperature $T_{LRS}(P_{LRS})$ of the targeted cell in LRS as follows:

$$T_{LRS}(P_{LRS}) = T_0 + R_{th} \cdot P_{LRS}. \tag{5.3}$$

To derive the alpha values, we apply the same approach but note that the temperature increase dT_{ij} in an adjacent cell, due to thermal crosstalk, constitutes only a portion of the total temperature rise $dT_{LRS} = R_{th} \cdot P_{LRS}$ in the heated device. The alpha values quantify this proportion and are defined as:

$$T_{ij}(P_{LRS}) = T_0 + R_{th} \cdot P_{LRS} \cdot \alpha_{ij}, \tag{5.4}$$

where α_{ij} denotes the alpha value for a particular memristive cell located between the BE i and the TE j. The alpha value for the central cell, origin of the thermal

crosstalk, is set to 1 to ensure Equation (5.4) equals with Equation (5.3). Therefore, alpha values for all neighboring cells will be less than 1. These values illustrate the extent of the thermal crosstalk and are influenced by the crossbar array's configuration, including electrode spacing and material properties. After determining the alpha values for a specific crossbar array, Equation (5.4) facilitates the computation of the resultant temperature matrix for varying power dissipations of a given targeted cell.

5.2.3 Thermal Crosstalk—Multiple Devices

In this section, we develop a methodology to quantify thermal crosstalk within memristive crossbar arrays when multiple memory cells are accessed simultaneously. This approach is particularly beneficial for examining the effects of various attack patterns on the efficiency of NeuroHammer attacks. To estimate the temperature rise due to thermal crosstalk from multiple cells, we adopt the premise that heat contributions from different sources can be superimposed. As established in Section 5.2.2, the temperature increase of a specific memory cell is expressed as

$$dT_{\mathrm{ij}} = R_{\mathrm{th}} \cdot P_{\mathrm{ij}}, \tag{5.5}$$

relying solely on the thermal resistance R_{th} and power dissipation P_{ij}. Assuming a constant thermal resistance for all cells, an adjacent device at position kl undergoes a temperature increase from the heat source at ij as described by:

$$dT_{\mathrm{kl}}^{\mathrm{ij}} = \alpha_{\mathrm{k-i+r,l-j+s}} \cdot dT_{\mathrm{ij}} \tag{5.6}$$

where r and s represent the distance from the selected cell to the top left corner of the alpha matrix. For example, considering the dimensions of a 5×5 crossbar as shown in Figure 5.2 (a), the offsets would be $r = 3$ and $s = 3$. Equation (5.6) illustrates that thermal crosstalk on an adjacent cell depends only on the relative distance and direction to the heat source, not on the absolute position of the selected cell within the array. Hence, for devices ij not positioned centrally, shifting the alpha matrix enables easy computation of thermal crosstalk. Consequently, the size of the alpha matrix does not need to correspond to the size of the crossbar array, considering that all memory cells outside the alpha matrix can be marked negligible in terms of their impact on thermal crosstalk. The cumulative temperature increase in a cell from the thermal crosstalk of all adjacent cells within the alpha matrix is given as

$$dT_{\mathrm{kl}} = \sum_{\substack{0<\mathrm{i}<=\mathrm{m} \\ 0<\mathrm{j}<=\mathrm{n}}} dT_{\mathrm{kl}}^{\mathrm{ij}} \tag{5.7}$$

where m and n represent the number of rows and columns in the memristive crossbar array, respectively. The dT_{kl} considers the thermal crosstalk from all adjacent

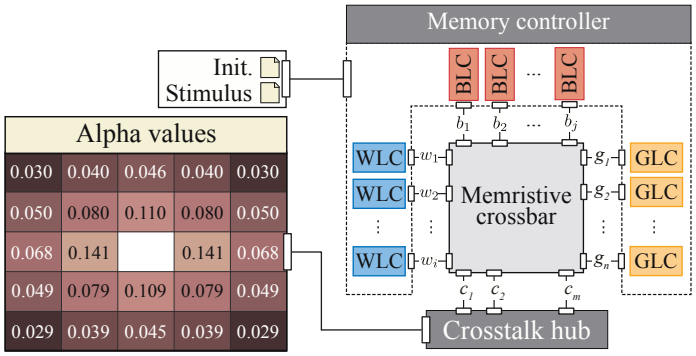


Figure 5.4: Overview of the circuit simulation methodology, comprising the memory controller, the crosstalk hub, and the crossbar array.

cells, including the cell's own self-heating ($k = i \wedge l = j$). The overall temperature for each device within the superimposed temperature matrix is defined as

$$T_{\mathrm{kl}} = T_0 + dT_{\mathrm{kl}} \tag{5.8}$$

$$= T_0 + \sum_{\substack{0<\mathrm{i}<\mathrm{m} \\ 0<\mathrm{j}<\mathrm{n}}} \alpha_{\mathrm{k-i+r,l-j+s}} \cdot dT_{\mathrm{ij}} \tag{5.9}$$

$$= T_0 + \sum_{\substack{0<\mathrm{i}<\mathrm{m} \\ 0<\mathrm{j}<\mathrm{n}}} \alpha_{\mathrm{k-i+r,l-j+s}} \cdot R_{\mathrm{th}} \cdot P_{\mathrm{ij}}. \tag{5.10}$$

Figure 5.3 (c-d) illustrates the methods for determining thermal crosstalk in a crossbar array. Equation (5.4) calculates the temperature of adjacent cells for a single centrally selected cell, while Equation (5.8) is utilized for computing resultant temperatures when two or more cells are selected. Here, the thermal crosstalk contribution for each selected cell is calculated individually, followed by the superposition of these distinct temperature matrices to result in the temperature profile of the crossbar array.

5.3 Circuit Simulation

In this section, we introduce a circuit-level simulation framework designed to investigate the effectiveness and feasibility of NeuroHammer. Cadence Virtuoso® [29], serving as the base circuit simulator, incorporates our modules implemented in VerilogAMS to enable scalable simulations of crossbar structures considering thermal crosstalk. The platform is designed to be customizable via the standard graphical user interface of Virtuoso, enabling the exploration of diverse crossbar structures and experimental configurations. Figure 5.4 presents an overview of the simulation framework, which includes the memory controller, the crosstalk hub, and the memristive crossbar array, with a detailed discussion in the following.

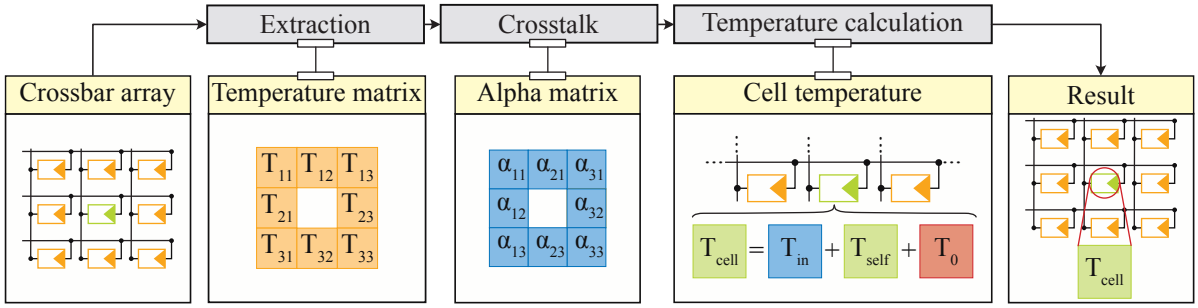


Figure 5.5: Detailed breakdown of the crosstalk hub: (a) extraction of temperatures from adjacent cells, (b) integration with alpha matrix, and (c) calculation of the final cell temperature.

5.3.1 Memory Controller

The memory controller represents the centerpiece of the circuit simulator responsible for orchestrating the essential signals for interfacing the crossbar model as shown in Figure 5.4. Specifically, the memory controller comprises three sub-controllers, which can be instantiated based on the crossbar array's type and dimensions. The World Line Controller (WLC) and the Bit Line Controller (BLC) manage the application of appropriate read/write/compute voltages to the targeted memory cells, with i and j denoting the number of rows and columns in the crossbar array, respectively. Similarly, the Gate Line Controller (GLC) activates the access transistors to select the memristive devices. Depending on the crossbar's architecture, n may correspond to the number of columns/rows as detailed in Section 2.1.2, or it may adjust to more complex configurations such as 2T1R crossbar arrays [214]. To execute various attack patterns on the crossbar, the sub-controllers must produce synchronized signals with exact timing specifications. These attack patterns are specified in a stimulus file that describes the pulse details (including length and amplitude), mapping (which allocates the pulses to specific lines), and timing. This setup is adjustable within the Cadence Virtuoso tool, allowing for parametric sweep analysis to investigate corner cases. Furthermore, an initialization (init) file specifies the initial state of each ReRAM cell.

5.3.2 Crosstalk Hub

The crosstalk hub acts as a centralized module for calculating thermal crosstalk within the crossbar array. Thus, as illustrated in Figure 5.4, the crosstalk hub relies on the alpha matrix to indicate the influence of surrounding cells on the temperature increase of a given cell (see Section 5.2.1). Figure 5.5 depicts the overall procedure for determining the absolute temperature of a cell, considering thermal crosstalk and self-heating. The temperature increase due to thermal crosstalk of a given cell T_{in} can be expressed as

$$T_{\text{in}}(\boldsymbol{\alpha}, \boldsymbol{T}) = \sum_{\substack{0<i<m \\ 0<j<n}} \alpha_{ij} T_{ij}, \tag{5.11}$$

where $\boldsymbol{\alpha}$ represents the alpha matrix, $\boldsymbol{T}$ denotes the temperatures of adjacent cells, and m, n are the number of rows and columns, respectively. The crosstalk hub determines T_{in} for each cell in every simulation cycle to simulate the continuous impact of thermal crosstalk.

5.3.3 Memristive Crossbar

The memristive crossbar module represents a versatile component for creating crossbar structures. For the memristive device model, we utilize the deterministic version of the JART VCM v1b model, designed for filamentary switching in VCM cells. This model has been calibrated to a nano-crossbar $Pt/HfO_2/TiO_x/Ti$ device [47, 19, 145].

The model internally employs the concentration of oxygen defects N_{disc} in the HfO_2 within a "disc" region at the Pt/HfO_2 interface as the state variable. These positively charged defects affect electron transport across the Pt/HfO_2 interface and the local conductivity. A high defect concentration signifies the device is in the LRS, while a low concentration indicates a HRS.

In general, ion migration under an applied electric field alters the defect concentration. Therefore, applying a negative voltage to the Pt electrode draws the positively charged oxygen defects, increasing N_{disc} and facilitating the SET transition. On the other hand, applying a positive voltage repels these defects, reducing N_{disc} and triggering the RESET state. The local temperature T is influenced by the dissipated power P_{d} according to

$$T = R_{\text{th,eff}} \cdot P_{\text{d}} + T_0, \tag{5.12}$$

where T_0 is the ambient temperature, and $R_{\text{th,eff}}$ represents the effective thermal resistance (in K/W), reflecting the heat dissipated to the surrounding cells and the thermal characteristics of the materials used. Further details on the JART VCM model are documented in [19], with the parameters for our experiments detailed in Appendix A.1.

The JART VCM model has been modified to enable the interaction with the crosstalk hub during simulation. Two interface variables have been introduced to relay the current device temperature to the crosstalk hub and to receive the temperature increase from adjacent cells. As illustrated in Figure 5.5, the temperature of a specific cell T_{cell} can be computed using Equation (5.8) and reformulated as

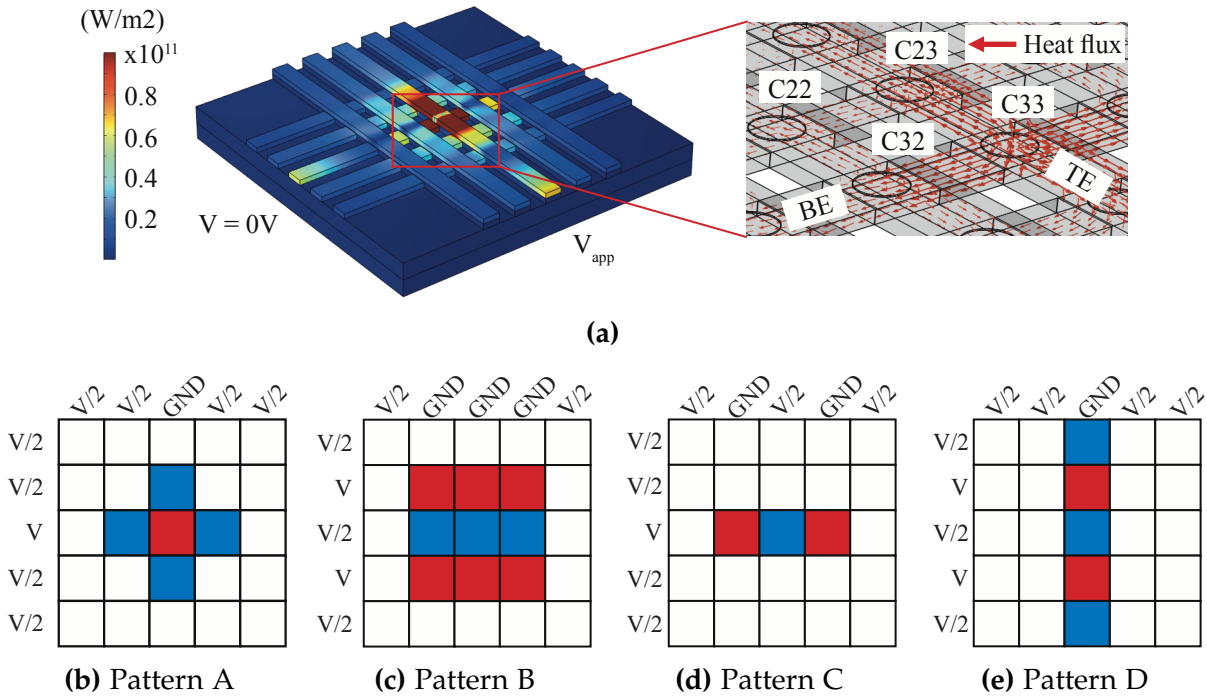


Figure 5.6: (a-d) Overview of the attack patterns employed in our experiments, with blue indicating the targeted cells and red highlighting the attacked cells. (e) Simulation results visualizing the heat flux along the electrodes, showing that heat from cell C33 is distributed via cells C23 and C32 to the adjacent cell C22.

$$T_{\text{cell}} = T_0 + dT_{\text{kl}} \tag{5.13}$$
$$= T_0 + T_{\text{self}} + T_{\text{in}} \tag{5.14}$$
$$= T_0 + \underbrace{V_{\text{m}} \cdot I_{\text{m}} \cdot R_{\text{th,eff}}}_{T_{\text{self}}} + T_{\text{in}}, \tag{5.15}$$
$$\tag{5.16}$$

where T_{cell} signifies the cell temperature, T_{in} the temperature rise due to thermal crosstalk as determined by the crosstalk hub (see Section 5.3.2), V_{m} the voltage difference across the memristor m, I_{m} the current through memristor m, T_0 the ambient temperature, and $R_{\text{th,eff}}$ the effective thermal resistance (in K/W).

5.4 Results

In this section, we utilize the developed simulator to explore the feasibility and effectiveness of the proposed NeuroHammer attack. Prior to delving into the specifics of the attack, we conduct a detailed analysis of the assumptions related to thermal crosstalk in memristive crossbar structures. Specifically, we validate the method of

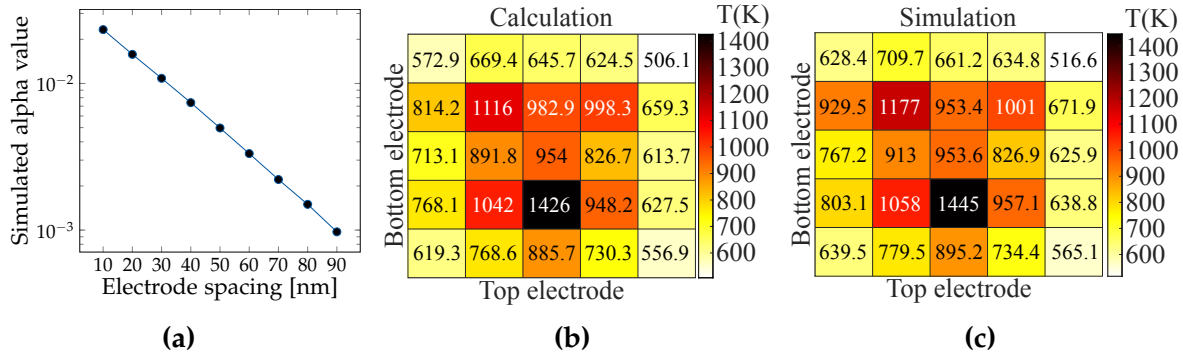


Figure 5.7: (a) The alpha value of a ReRAM cell as a function of electrode spacing in a crossbar array. A comparison between (b) temperatures obtained from simulation and (c) temperatures calculated by employing the method of superimposing temperature matrices.

superimposing temperature matrices. Based on these findings, we validate the applicability of our simulation approach for investigating the NeuroHammer attack on 1T1R structures.

Following the thorough justification of the simulation methodology, we present a comprehensive series of experiments examining the effects of pulse length, electrode spacing, ambient temperature, and various attack patterns on the NeuroHammer attack. Additionally, we investigate different technology nodes and device variations in 1T1R structures.

5.4.1 Thermal Simulation

To understand the implications of thermal crosstalk in memristive crossbar arrays, the heat flux is studied to provide insights about the parameters influencing the strength and the propagation. Furthermore, the assumptions made in Section 5.2.3 have to be verified. For this experiment, we utilize the implemented crossbar array in Section 5.2 along the boundary conditions shown in Figure 5.2.

Heat flux: Figure 5.6 (a) visualizes the heat flux of within a passive crossbar array based on arrows. In this setup, the center cell (C33) is set to the LRS and being selected by the respective BE and TE. *The heat flux, indicated by the red arrows, spreads primarily into symmetrically into the top and bottom electrodes.* Considering that the thermal conductivity of the oxide layer under the top electrode and the substrate is significantly lower compared to that of the electrodes, the heat naturally remains in the more thermally-conductive lines. Therefore, the adjacent cells C23 and C32 experience a temperature increase due to thermal crosstalk. *Likewise, Figure 5.6 (a) shows that the thermal coupling decreases the farther the cells are apart from the heat source. As a consequence, the dimensions and the spacing of the crossbar array plays a crucial role for the magnitude of the thermal crosstalk.* Therefore, encasing this setup in a thermally

insulating material like SiO_2 would not significantly alter the temperature profiles, since heat dispersal primarily occurs through the electrodes.

Figure 5.7 (a) shows the alpha value of the cell (1,1) depending on the spacing between the top/bottom electrodes from one row/column to the next one. *The results illustrate a general tendency that the farther the electrodes are apart, the thermal crosstalk becomes less prevalent. However, since the selected lines also increase their temperature due to Joule heating, the alpha values may increase depending on the geometric and material properties of the electrodes.* Since the heat distributes primarily along the electrodes, the thermal crosstalk may be reduced by intentionally increasing the gap between the memristive devices and the electrodes. Such a thermal isolator may be represented by a transistor as used in 1T1R structures. *Therefore, these more complex crossbar structures can be simulated by reducing the alpha matrix to a line vector because the heat flux only propagates along the electrodes directly connected to the memristive device while the transistor effectively serves as isolator.*

Superimposing temperature matrices: To validate the proposed method of superimposing temperature matrices, we conducted simulations on a pattern involving three selected cells in the LRS, with the remaining devices in the HRS. Figure 5.7 (b) displays the simulated temperature matrix for this pattern, while Figure 5.7 (c) presents the resulting temperature matrix derived from the alpha values. *This comparison reveals that the simulated values closely match the calculated temperatures, especially for cells located in the central portion of the array.* However, discrepancies between the simulated and calculated temperatures become more noticeable for cells at the array's periphery. *In the context of investigating NeuroHammer, this level of approximation is sufficient, as the attack primarily aims to induce bit-flips in adjacent cells.* In general, the precision of our method could be enhanced by applying a larger alpha matrix to a comparatively smaller actual array size. For instance, a 9×9 alpha matrix could be used to approximate temperatures within a 5×5 crossbar array, ensuring comprehensive coverage of the 5×5 array, even for memory cells at the array's edges. Nevertheless, it is important to note that comparing a crossbar array to an alpha matrix of differing dimensions may introduce complexities due to variations in boundary conditions.

Von Witzleben et al. [195] conducted a study exploring the impact of temperature on the switching kinetics of memristive devices. Their findings reveal highly localized temperatures within the device, reaching approximately 1000 K, which are crucial for facilitating rapid switching behavior in VCM cells within the nanosecond timeframe. *The absolute cell temperatures produced by our simulations align closely with the measurements results on real memristive devices presented by Von Witzleben et al., thereby affirming the validity and precision of our simulation approach.*

5.4.2 1R Crossbar Arrays

In this section, we examine the NeuroHammer attack at the circuit level for passive crossbar arrays (1R), with the objective of assessing the impact of pulse length, ambient temperature, electrode spacing, and different attack patterns on the number of

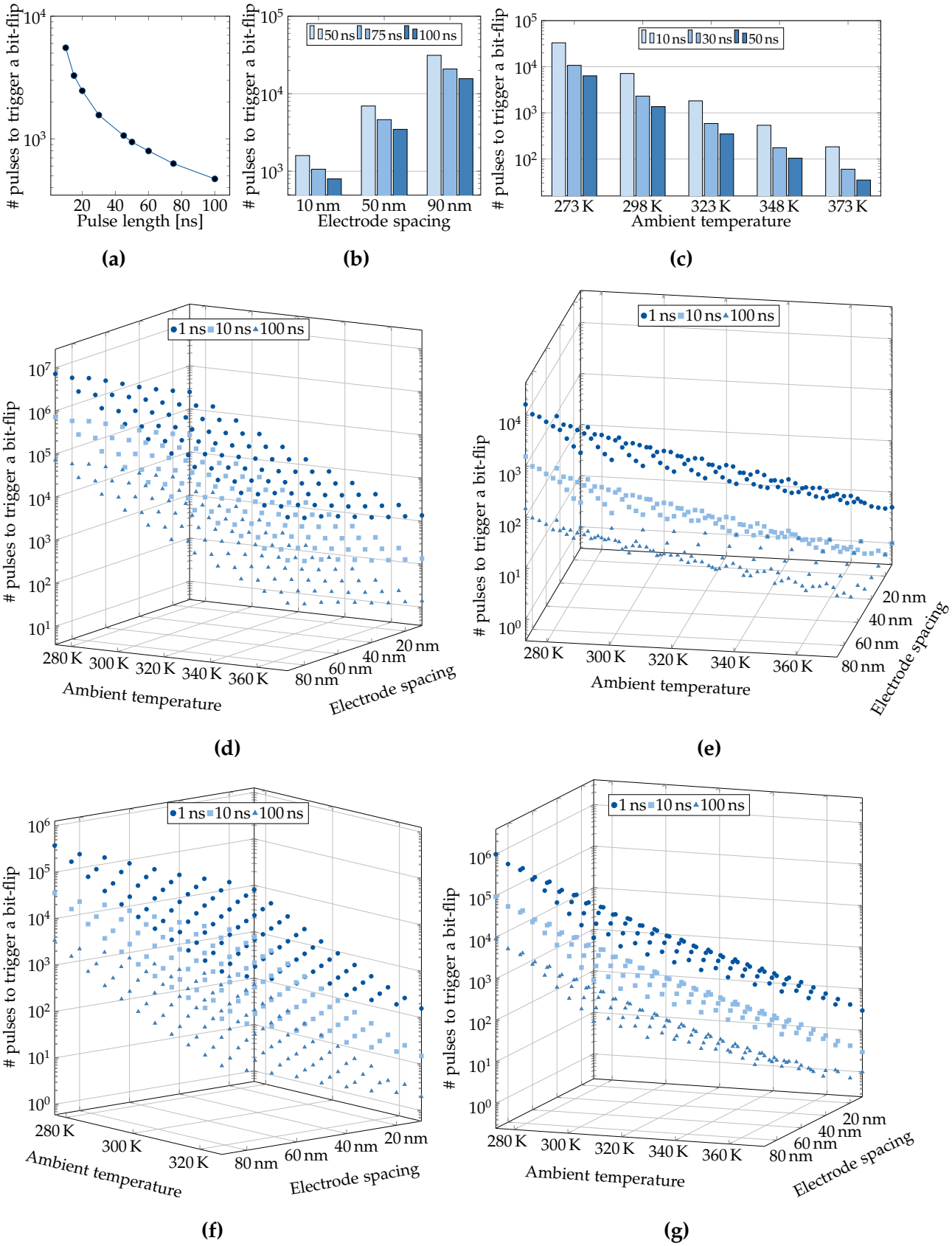


Figure 5.8: Circuit-level simulation results for passive crossbar structures are presented as follows: (a) the effect of pulse length (pattern A), (b) the influence of electrode spacing (pattern A), (c) the effect of ambient temperature (pattern A), and (d-g) the impact of attack patterns (A-D), respectively.

pulses required to induce a bit-flip. The attack patterns used in our experiments are depicted in Figure 5.6 (b-e), where red tiles mark the attacked cells and blue tiles indicate the cells most susceptible to a bit-flip due to the NeuroHammer attack. The likelihood of a bit-flip in blue cells is higher because they share a common electrode with the attacked cell (as discussed in Section 5.4.1) and are in proximity to it. The alpha matrices utilized in these experiments are detailed in Appendix A.2.

Pulse length: This experiment explores how the length of the write pulse affects the NeuroHammer attack's efficacy. We employ a 5×5 passive crossbar with an electrode spacing of 50 nm and an ambient temperature of 300 K, using pattern A (as shown in Figure 5.6 (b)) to target a single cell located in the center. The memory controller's sub-controllers apply the necessary voltages to the rows and columns, with the attacked cell receiving V_{Write} and Ground (GND), while the $V/2$ scheme is used for the rest to mitigate sneak-path currents. *Figure 5.8 (a) plots pulse lengths against the number of pulses needed for a bit-flip, showing that longer pulses reduce the required number of pulses to flip a bit in adjacent cells.*

Electrode spacing: As shown in Section 5.4.1, the heat flux decreases with increasing distance to the attacked cell. To quantify this impact on the NeuroHammer attack, we investigate the influence of the electrode spacing on the NeuroHammer attack. Again, Pattern A is utilized with an ambient temperature of 300 K. Figure 5.8 (b) illustrates the number of pulses required to trigger a bit-flip in dependence of the electrode spacing and the pulse length. *In general, the results indicate that memristive crossbars are more susceptible to NeuroHammer as the electrode spacing decreases. Consequently, we argue that security attacks like NeuroHammer become a severe problem for dense crossbar memories as the technology node advances.*

Ambient temperature: As indicated by Equation (5.8), ambient temperature T_0 significantly influences the temperature rise in a memristive device within crossbar structures. This experiment examines the effect of ambient temperature on the number of pulses required to induce a bit-flip. We utilize attack pattern A on a passive crossbar array with an electrode spacing of 50 nm. As demonstrated in Figure 5.8, the impact of the ambient temperature is profound. At 273 K, 33,030 pulses are needed to cause a bit-flip, whereas at 373 K, merely 184 pulses suffice. *This finding highlights that higher ambient temperatures significantly lower the pulse count needed for a NeuroHammer-induced bit-flip, posing a potential reliability risk for memristive crossbar arrays.*

Attack patterns: Considering that a potential adversary may not have direct access to the targeted system, it might be challenging to externally influence the parameters previously discussed. However, utilizing specific attack patterns, which involve repeatedly writing a certain bit pattern within a memory region, offers a viable strategy to enhance the NeuroHammer attack's effectiveness. Initial experiments employed the attack pattern depicted in Figure 5.6 (b), targeting a single cell at the center of the crossbar array. Nonetheless, our results presented in Section 5.4.1 suggest that attacking multiple cells simultaneously can increase the overall heat flux within the crossbar, thereby altering the switching kinetics of the targeted cell in a way that benefits the injection of bit-flips.

Table 5.1: Simulation parameters for access transistors at 180 nm and 45 nm nodes, and slow, medium, and fast memristors.

Transistor	**Parameter**		
180nm	Name: nmos	w_{gate}: 2 µm	S/D metal width: 60 µm
	Version: v3.3	l_{gate}: 180 nm	Folding threshold: 10 µm
	Fingers: 1	V_{Gate}: 1.8 V	
45nm	Name: nmos1v	w_{gate}: 120 nm	S/D metal width: 60 µm
	Version: v6.0	l_{gate}: 45 nm	Folding threshold: 10 µm
	Fingers: 1	V_{Gate}: 1.8 V	
Memristor	**Parameter**		
Slow	N_{max}: 0.39×10^{26} m−3	l_{var}: 0.44 nm	V_{SET}: 1.54 V
	N_{min}: 4×10^{23} m−3	r_{var}: 40.5 nm	
Medium	N_{max}: 0.4×10^{26} m−3	l_{var}: 0.40 nm	V_{SET}: 1.54 V
	N_{min}: 8×10^{23} m−3	r_{var}: 45.0 nm	
Fast	N_{max}: 0.41×10^{26} m−3	l_{var}: 0.36 nm	V_{SET}: 1.54 V
	N_{min}: 25×10^{23} m−3	r_{var}: 49.5 nm	

Hence, this experiment explores the effects of varying attack patterns. Pattern B, see Figure 5.6 (c), seeks to maximize heat flux by simultaneously attacking six cells surrounding the target cell, with the intermediary three cells showing the highest likelihood of undergoing a bit-flip due to their shared bottom electrode with the attacked cells. Pattern C, as shown in Figure 5.6 (d), employs the word line as a thermal conductor connecting the attacked and target cells. The effectiveness of this pattern may depend on the memory controller's addressing scheme, potentially limiting the strategy's efficiency. Figure 5.6 (e) presents a variation of Pattern C.

The outcome of the simulations, corresponding to Patterns A-D, are illustrated in Figure 5.8 (d-g), conducted across different electrode spacings, ambient temperatures, and pulse lengths. *As anticipated, the data confirms that attacking more cells at once reduces the number of pulses needed for a bit-flip. Moreover, the proximity of the attacked cells to the target cell is critical, as heat transfer predominantly occurs through the electrodes.*

5.4.3 1T1R Crossbar Arrays

As discussed in Section 2.1.2, passive crossbar arrays are prone to sneak-path currents, which substantially impact their reliability. To address this, 1T1R structures have been proposed, incorporating an access transistor to effectively isolate unselected memory cells within the array.

Nevertheless, the NeuroHammer attack requires a trigger pulse to induce a bit-flip in the target cell. In contrast to passive crossbar arrays that deploy the $V/2$ scheme which is used as trigger pulse, such a scheme is unnecessary in 1T1R arrays. Table 2.2 provides an overview of the writing schemes utilized for the three possible 1T1R configurations—each designed to mitigate sneak-path currents, thereby countering the NeuroHammer threat. *After simulating all three 1T1R structures—typical, vertical, and pseudo—using an ideal access transistor, this assumption holds true, and no bit-flips occur.*

Consequently, the following experiments adopt the Cadence Generic Process Design Kit (GPDK) to incorporate a realistic transistor model, which facilitates the simulation of leakage currents across diverse technology nodes. Furthermore, we incorporate the model parameters described by Bengel et al. [19], which are based on measurements of memristors, each characterized by fast, medium, and slow switching behaviors. Table 5.1 shows the simulation parameters for the memristive devices and transistors. All simulations were executed at a constant temperature of 293.15 K and with a 10 nm inter-electrode gap.

Crossbars: This experiment seeks to evaluate the potential for NeuroHammer attacks within 1T1R structures, utilizing a realistic transistor model for analysis. Illustrated in Figure 5.9 (a-c), are the vertical, typical, and pseudo crossbar configurations, where cells under attack are marked in red and those susceptible to leakage currents are highlighted in blue. These susceptible cells are identified by the occurrence of a voltage drop across the memory cell which may lead to leakage currents through the access transistor. For this evaluation, we apply the fast memristor parameters together with the 45 nm access transistor. As depicted in Figure 5.9 (d), we examine the resistance of the susceptible memristor throughout the simulation for each type of crossbar. *The results reveal that the leakage current passing through the access transistors could incrementally alter the resistance, potentially culminating in a bit-flip.* Findings in Section 5.4.1 suggest that the thermal flux predominantly traverses via the electrodes across the crossbar array. Consequently, the typical and vertical arrays demonstrate comparable susceptibility for bit-flips. *Nonetheless, the pseudo crossbar demonstrates an immunity to NeuroHammer attacks due to its unique design wherein the attacked cell and the susceptible cells do not share an electrode.* Despite experiencing similar leakage currents, the lack of thermal crosstalk within the pseudo crossbar architecture prevents substantial changes in resistance.

Memristor variability: Given that the leakage current passing through the access transistor is minimal, the intrinsic switching behavior of the memristive device becomes pivotal for the feasibility of the NeuroHammer attack. In this context, our experiment examines the influence of three distinct parameter sets for the JART VCM model, which are derived from empirical measurements of real devices. These parameters, as reported by Bengel et al. [19], correspond to fast, medium, and slow switching behaviors and are detailed in Table 5.1. The simulation results, illustrated in Figure 5.9 (e), demonstrate the resistance dynamics of a vulnerable cell within a typical 1T1R crossbar structure under repeated hammering on an adjacent cell, marked in red. For this experiment, we employ attack pattern A alongside a 45 nm access

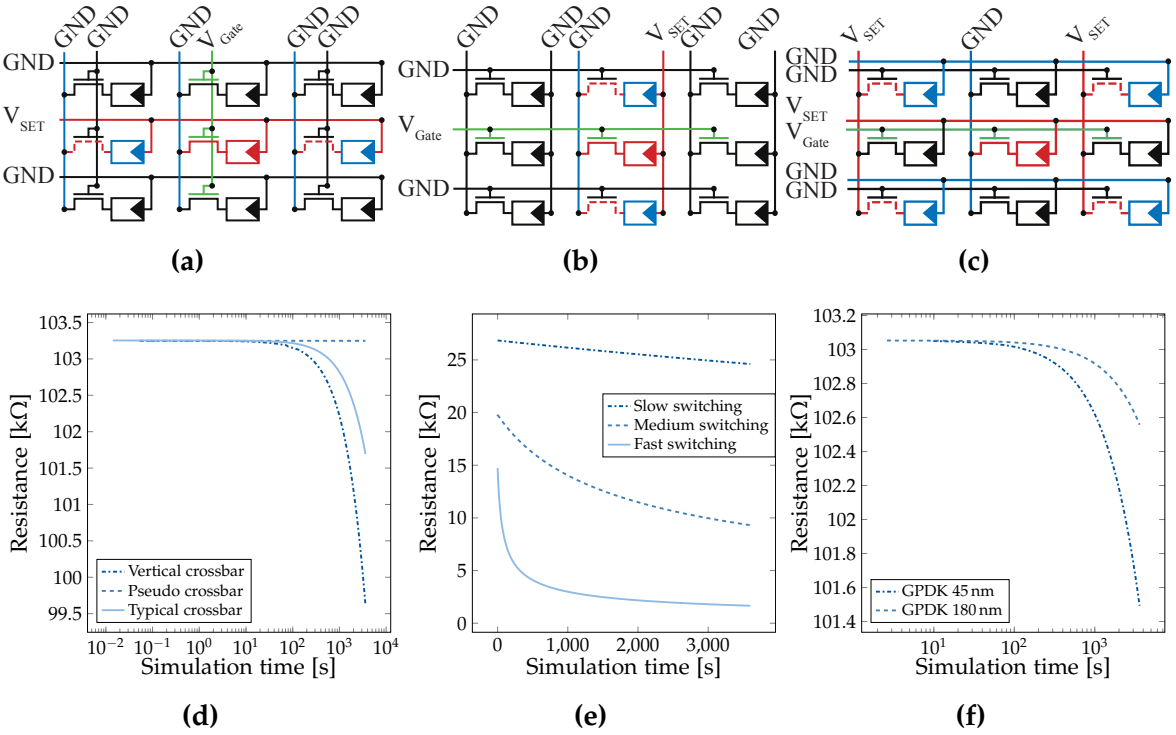


Figure 5.9: Circuit-level simulations of (a) vertical, (b) typical, and (c) pseudo 1T1R crossbar structures, with impact of the crossbar structures (d), memristor variability (e), and (f) technology node.

transistor. It becomes evident that device variability significantly influences the success rate of the NeuroHammer attack. *The results indicate that while the memristor with slow switching attributes exhibits only marginal resistance fluctuations, the device with fast switching properties experiences a significant resistance shift, which eventually result in a bit-flip.*

Access transistor: The impact of the leakage current through the access transistor is critical in our investigation, prompting an examination of various transistor models across two distinct Generic Process Design Kit (GPDK) technology nodes. Anticipating future miniaturization of memristive crossbar arrays to enhance memory density and cost efficiency, it is projected that both memristive devices and their accompanying access transistors will experience technological scaling.

In this experiment, we utilize a typical crossbar array together with the fast memristor parameter set. *The findings, as depicted in Figure 5.9 (f), confirm the hypothesis that smaller technology nodes result in higher leakage currents, which in turn, enhance the potential for a successful NeuroHammer attack.*

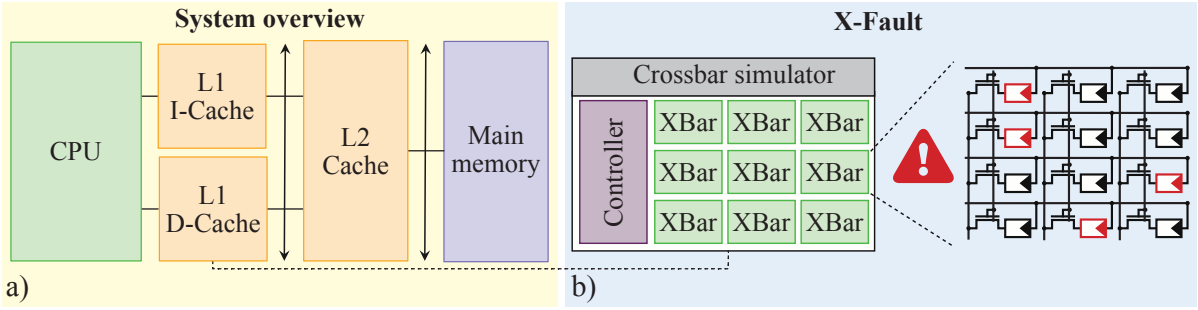


Figure 5.10: (a) Overview of the simulated computing system including processor model, cache memories and main memory. (b) X-Fault's crossbar simulator integrated in the Gem5 architecture simulator.

5.5 Case Study: Leaking RSA Keys with NeuroHammer

In Section 5.4, we have comprehensively investigated the existence of the NeuroHammer attack and the influence of physical parameters on its effectiveness. The ability to deliberately flip bits in memories employed in modern computing systems pose a fundamental threat as shown by Rowhammer for DRAMs (refer to Section 2.4.3). However, DRAM is dominantly used as main memory because of its limited access time in comparison to Static Random-Access Memory (SRAM). Therefore, the attack surface of the Rowhammer attack is bound to data stored in the main memory.

On the other hand, ReRAM, while still being heavily under research, is a promising candidate to replace not only DRAMs [124] but also SRAMs [117] for cache memories. Considering the existence of NeuroHammer, intentional bit-flip attacks are no longer restricted to main memories but may also affect cache memories. Therefore, in this case study, we explore the potential impact of NeuroHammer attacks in cache memories. In particular, we utilize the crossbar simulator presented in Section 4.3 and connect it to the architecture simulator Gem5 [25]. This simulation setup enables us to simulate a complete computing system including a processor, caches, and main memory.

In the following, we define a comprehensive attack scenario in Section 5.5.1. The implemented simulation methodology is discussed in Section 5.5.2 together with a thorough evaluation in Section 5.5.3.

5.5.1 Attack Scenario

In this case study, we assume a computing system with two users, victim and adversary, whereas the victim executes an RSA signature generation and the adversary aims to leak the secret key. Our attack scenario assumes that the attacker has the ability to execute arbitrary code within a distinct user process environment. The attacker's influence is restricted to the virtual memory space of his own process, thus prohibiting any direct modifications of data outside this context. Furthermore, the adversary also possesses thorough knowledge of the processor's architecture and its

```
#ifndef H_FAULTYMEMIF_HPP
#define H_FAULTYMEMIF_HPP
#include <cstdint>
#define GEM5_PACKET_MAXSIZE 64 /* in bytes */
typedef enum {PAGE_INTERLEAVE, RANK_INTERLEAVE} addr_trans_t;

class FaultyMemIF {
    public:
        virtual int size_gigabytes(void) = 0;
        virtual uint64_t size_kilobytes(void) = 0;
        virtual void write(uint64_t phys_addr,
                           uint8_t *src,
                           uint_fast16_t len) = 0;
        virtual void read(uint64_t phys_addr,
                          uint8_t *dst,
                          uint_fast16_t len) = 0;
        FaultyMemIF(addr_trans_t transltr) {
            buf_len = GEM5_PACKET_MAXSIZE/sizeof(uint64_t);
            translator = transltr;
        }
        virtual ~FaultyMemIF() {};
    protected:
        uint64_t      buffer[GEM5_PACKET_MAXSIZE/sizeof(uint64_t)];
        uint_fast16_t buf_len;
        addr_trans_t  translator;
};
#endif /* H_FAULTYMEMIF_HPP */
```

Figure 5.11: Abstract memory interface of X-Fault's crossbar simulator. The interface defines the required functions to enable the interaction between gem5 and the crossbar simulator.

memory hierarchy. The victim uses an RSA implementation called Chinese Remainder Theorem (CRT) optimization to enhance signature generation.

Generally, RSA relies on the computational difficulty of factoring a large number N, which is the product of two large prime numbers, p and q. While an RSA signature is generated using

$$s = m^d \mod N, \tag{5.17}$$

a given message is verified by

$$s^e = m \mod N, \tag{5.18}$$

where m represents the plaintext message, $\{N, e\}$ is the public key, $\{N, d\}$ is the private key, and s is the resulting signature [163, 70]. RSA-CRT optimizes RSA by performing calculations separately for p and q, reducing the computational load by working with smaller numbers. In RSA-CRT, the signature is calculated as

$$\left.\begin{aligned} s_1 &= m^{d_p} \mod p \\ s_2 &= m^{d_q} \mod q \end{aligned}\right\} \implies s \mod pq, \tag{5.19}$$

where $d_p = d \mod (p-1)$ and $d_q = d \mod (q-1)$. However, RSA-CRT is vulnerable to fault attacks. If an attacker induces a fault during these calculations, it can result in an incorrect signature $\widetilde{s}$, computed as

$$\left.\begin{aligned} s_1 &= m^{d_p} \mod p \\ \tilde{s}_2 &= m^{d_q} \mod q \end{aligned}\right\} \implies \tilde{s} \mod pq \tag{5.20}$$

By analyzing the difference between $\widetilde{s}$ and the correct signature s, the attacker can extract the private key. Specifically, the adversary exploits the fact that $gcd(\widetilde{s}^e - m, N)$ reveals p, thereby compromising the entire RSA key pair since $N = pq$ [26].

The adversary exploits this vulnerability in RSA-CRT by injecting bit-flips using NeuroHammer in the ReRAM-based L1 data cache. By disrupting the victim's signature generation process, the adversary is able to retrieve the secret key, thereby gaining full control over the signature process. Unlike traditional fault attacks, which require physical access, this scenario involves a purely software-based attack, potentially allowing for remote execution without any need for physical access to the computing system.

To the best of our knowledge, this scenario represents a novel non-intrusive attack that can actively inject faults in ReRAM caches. Furthermore, it emphasizes the profound implications of NeuroHammer on ReRAM technology and highlights the urgent need for continued research in hardware security, especially within the context of Emerging Non-Volatiles Memories (eNVMs).

5.5.2 Simulation Methodology

The Gem5 computer system architecture simulator is employed to emulate the processor, including caches and main memory. The processor model implements an out-of-order x86 processor operating at 1GHz, equipped with L1 instruction and data caches of 16 kB each, a unified 256 kB L2 cache, and 2 GB of main memory. We have enhanced the Gem5 simulator by integrating a generic memory interface to link with the crossbar model outlined in Section 4.3, as depicted in Figure 5.10. The X-Fault crossbar simulator's generic interface, illustrated in Figure 5.11, includes a `write` and `read` function. These functions are set up as callback functions, triggered whenever the processor attempts to access cache memory. Originally, X-Fault's crossbar simulator monitored read, write, and compute accesses for each cell. However, to enhance our simulation's efficiency, we have deactivated this tracking feature and simplified the crossbar model to solely accommodate the essential fault model. The fault model is parameterized by a specific pattern and a threshold value, which dictate the number of write accesses required to trigger a bit-flip in an adjacent cell.

5.5.3 Evaluation

In this section, we detail the steps an adversary must undertake to successfully induce a bit-flip at the precise location within the L1 data cache during the victim's

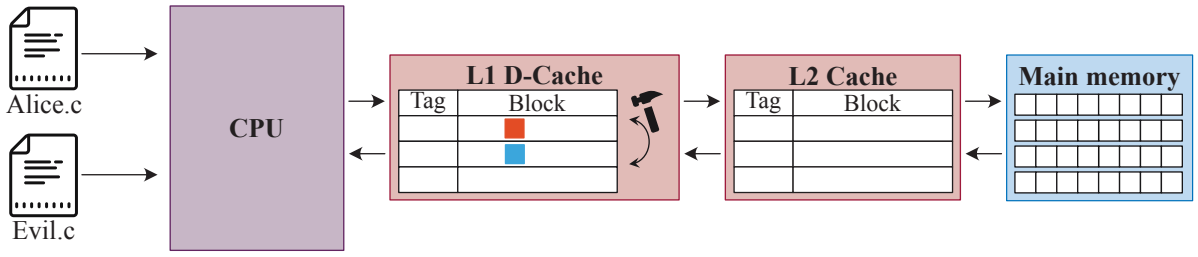


Figure 5.12: Attack Scenario: leveraging NeuroHammer to induce bit-flips in the L1 data cache, ultimately compromising the victim's secret key.

RSA signature generation process as shown in Figure 5.12. For an adversary to introduce bit-flips into a specific memory cell, it is crucial to comprehensively profile the cache memory during the RSA signature generation, aiming to achieve two objectives. Initially, the attacker needs to identify the cache sets utilized during the signature process and, subsequently, select an address of a neighboring cache line that will serve as the target for the attack.

Target cache sets: Identifying a potential target cache set begins with the adversary allocating a substantial amount of data and accessing it. This action ensures that the data populates the L1 data cache, spanning multiple cache sets. Then, by executing an RSA signature generation, the attacker profiles the cache to identify which cache sets are being evicted. By repeating this process, it becomes apparent which addresses are frequently evicted, indicating a high likelihood of containing data used during the victim's signature generation. Figure 5.13 (a) displays a heatmap of the L1 data cache activity during signature generation, where light blue represents minimal write activity and dark blue/black signifies intense activity. To compromise the integrity of the signature generation process, the attacker aims to inject a single bit-flip in a heavily utilized cache set, where adjacent sets are infrequently accessed. As an example, cache set 15 as depicted in Figure 5.13 (a) would fulfil this requirement.

Adjacent address: The method to determine an adjacent address varies based on the provided memory allocation technique. If the CPU architecture permits the use of larger-than-standard pages, known as huge pages, the attacker can easily choose a cache set by altering the lower bits of the address within the huge page. With standard page sizes, the attacker may adjust the virtual address to acquire the address of an adjacent cache set. Considering a cache entry of 64 B, the six Least Significant Bits (LSBs) of the virtual address determine the offset. The following bits of the virtual address specify the corresponding cache set. By inverting the seventh bit of the virtual address, which represents the LSB of the set index, an address adjacent to the original address in the cache is generated.

L1 cache attack: Once the adjacent memory cell's address has been identified, the attacker proceeds with the actual attack by repeatedly hammering the targeted cell. The pseudocode for this method is outlined in Algorithm 5.1. This algorithm requires the target address, which correlates with a cache set active during RSA signature generation, the victim user to initiate the signature process, and an arbitrary message.

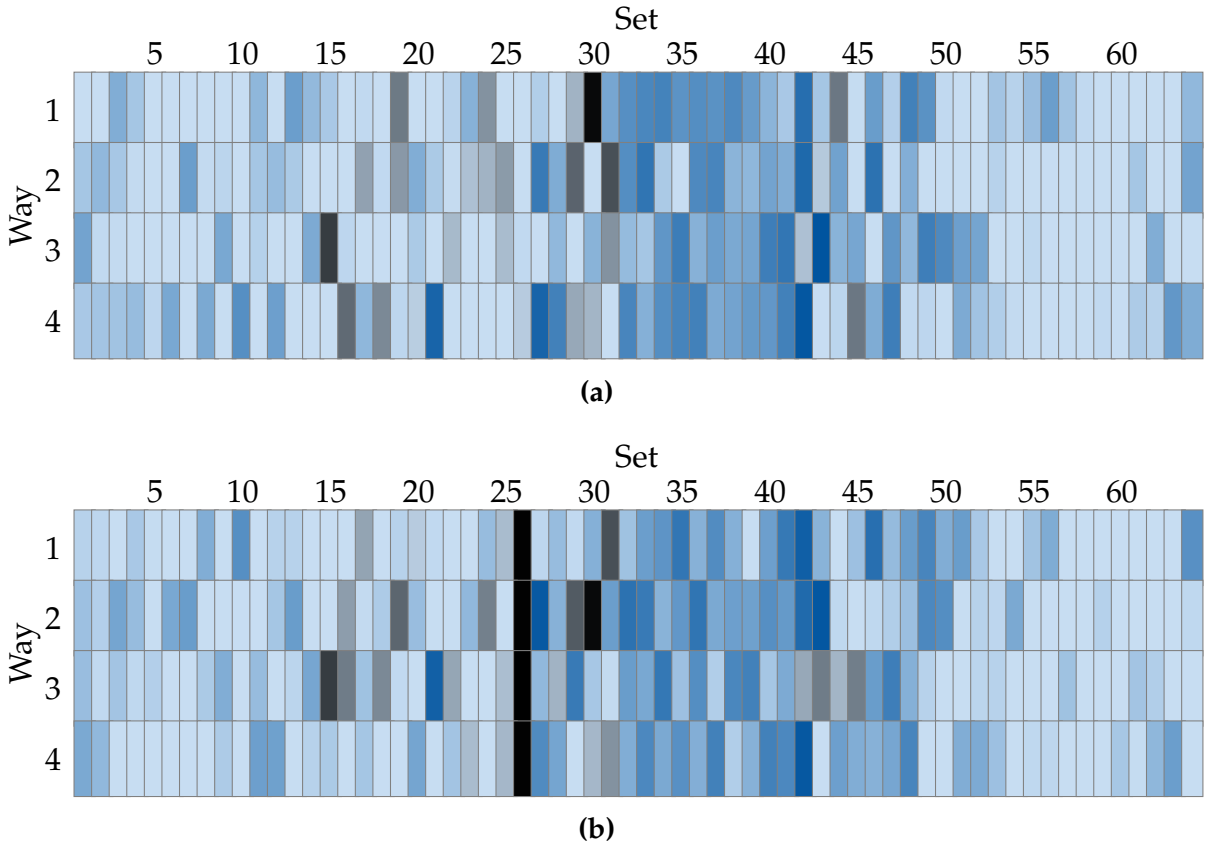


Figure 5.13: Write access patterns to the L1 data cache during RSA signature generation: (a) in the absence of an attacker, and (b) with an active attacker targeting the 26th cache set.

To enhance the attack's efficacy, we leverage eviction sets to target the entire cache set rather than a single entry. Eviction sets consist of a collection of virtual addresses that all map to the same cache set [193]. Constructing eviction sets can be achieved through either a top-down approach [172] or a bottom-up strategy [188], making the attack agnostic to the cache's replacement policy. Attacking a single address risks the RSA-CRT data being located in a different cache way than the intended target. By employing eviction sets, faults are induced across all adjacent memory cells linked to the eviction set's addresses, potentially causing unintended application crashes, although this was not observed in our experiments. Additionally, the flipped bits are unlikely to be propagated to the main memory since the entries are not marked as *dirty*. Next, the attacker requests a signature from the victim while simultaneously hammering the adjacent cache line. Figure 5.13 (b) illustrates the L1 data cache's simulated write activity during the attack, with cache set 26 distinctly as showing intense activity marked in black, aiming to induce bit-flips in the adjacent 27th set. Our simulations consistently managed to inject bit-flips within the RSA-CRT signature generation using the NeuroHammer attack. As elaborated in Section 5.5.1, injecting a fault during the computing of s_1 and s_2 results in a faulty signature $\tilde{s}$. Finally, the adversary can retrieve the private key of the victim by calculating $p = \gcd(\tilde{s}^e - m, N)$.

Algorithm 5.1: Injecting bit-flip during RSA-CRT signature generation with NeuroHammer.

Data: Target address (t), victim user (*victim*), message (*msg*)

```
1  EvSet ev ← getEvSet(t-64);
2  (e, N) ← getPubKey(victim);
3  sendSignReq(victim, msg);
4  while !sig = recvSignResp(victim) do
5  |  for Addr a in ev do
6  |  |  a.data ← 0b0000...1...0;
7  |  |  a.data ← 0b0000...0...0;
8  |  end
9  end
10 if sig.valid then
11 |  return 0;
12 else
13 |  return gcd(sig^e − m, N);
14 end
```

5.5.4 Additional Attack Targets

This section provides a brief overview of potential targets susceptible to the NeuroHammer attack, extending the scenario illustrated in Figure 5.12. It assumes that both L1 and Last Level Cache (LLC) are composed of memristive crossbar arrays, rendering them vulnerable to NeuroHammer.

Last Level Cache (LLC) attack: In our initial analysis, we considered a situation where both the victim and the attacker share a single processor core via hyper-threading, allowing access to the same L1 cache. However, in systems prioritizing security, this feature might be deactivated, restricting attackers from accessing the shared L1 cache memory. Under these circumstances, an adversary might target the LLC for the attack. For an *inclusive* LLC, the attack procedure is relatively straightforward: all write operations to the L1 cache are also immediately reflected in the LLC. Thus, any bit-flips induced in the L1 cache will similarly affect the LLC. However, it's unlikely that the bit-flip impacts the identical data in both caches, a nuance the attacker must account for. On the other hand, *non-inclusive* LLCs inherently block access to other cores' private caches, thereby hindering NeuroHammer's ability to induce bit-flips directly [34]. Consequently, the adversary needs to identify a cache set that contains data critical for the victim's signature generation process yet lies outside the victim's core private cache. While locating such a cache set poses a significant challenge, the attacker benefits from essentially limitless attempts, with a single successful bit-flip revealing the secret key.

Tag/Flag bit attacks: The NeuroHammer attack allows an attacker to arbitrarily flip bits in memristive memories. Thus, a potential adversary might aim at the tag and flag bits within a cache entry, leading to data corruption and possibly data loss. To manipulate the tag section of an adjacent cache set, an adversary must frequently

access an eviction set that exceeds the cache's associativity. This action results in the tag section being continuously updated with various tags, eventually triggering a bit-flip fault in the targeted entry. Such an occurrence could cause incorrect cache hits for certain addresses or even intentionally alter the tag of a dirty entry to disrupt write operations. In addition to targeting the tag section, an attacker might also focus on the flag section of a cache entry. For example, by altering the valid flag, the adversary can falsely mark specific cache entries as valid, leading to data corruption and inconsistent cache states.

5.6 Limitations and Outlook

Given the impact of the NeuroHammer attack identified in our study, it's important to discuss the limitations of our investigation. Although our dual-simulation approach—combining detailed thermal simulation with higher-level circuit analysis—demonstrates considerable accuracy, the actual verification of thermal crosstalk and the existence of the NeuroHammer attack on real hardware remains unexplored. Our simulation, inspired by crossbar arrays fabricated in various research labs [103, 104, 125, 205, 197], may not fully capture the complexities of thermal crosstalk in fully integrated systems. Moreover, more sophisticated crossbar structures, such as the 2T1R configuration, may be potentially immune to the NeuroHammer attack which our study does not address. Additionally, the attack scenario inspired by the attack model of Rowhammer, may not cover the entire range of possible real-world threats, possibly overlooking capabilities and constraints of an attacker.

To bridge these gaps, a more holistic simulation approach is required by considering a wider range of crossbar structures and calibrate our simulation with measurements of physical structures. Future work should aim to identify new potential attack vectors, extend the attack scenario to cover a broader spectrum of threats, and crucially, investigate effective countermeasures and mitigation techniques.

5.7 Synopsis

This chapter presents NeuroHammer, a novel hardware security threat that compromises memory integrity by causing bit-flips in memristive crossbar arrays. We have adopted a dual-simulation approach to examine thermal crosstalk in nanoscale crossbar structures, which represents the fundamental mechanism behind NeuroHammer. Our extensive analysis of system parameters confirms the attack's feasibility, demonstrating its effectiveness and limitations. Through a case study, we highlight the practical consequences of NeuroHammer by detailing an attack scenario where cache memories are targeted. The chapter concludes by outlining further research perspectives to refine our simulation approach and validate it against actual crossbar array hardware.

Chapter 6

Instrumentation Platform for Non-Volatile Memory Technologies

The previous chapters have underscored the significant influence of memristive devices' nonidealities on the reliability and security of neuromorphic computing systems. Although the discussed simulation methodologies aim to replicate these inherent properties accurately, simulation alone cannot offer a comprehensive analysis that incorporates real memristive devices and circuits.

As highlighted in Section 3.4, various instrumentation platforms for emerging non-volatile memory technologies have been developed, primarily focusing on characterizing memristive devices. Yet, existing platforms fall short in performing Computing-in-Memory (CIM) operations on memristive crossbar arrays, leading to a reliance on simulation for most reliability and hardware security research.

Therefore, this chapter introduces the NeuroBreakoutBoard (NBB), a versatile and adaptable instrumentation platform designed to investigate the characteristics of memristive devices at the device, crossbar, and operational levels. Equipped with custom-designed signal generation and sensing circuitry, the NBB enables precise control over memristive cell programming and supports the execution of both analog CIM and Logic-in-Memory (LIM) operations. Additionally, the platform features three distinct application interfaces, facilitating seamless integration of its measurement capabilities across different levels of analysis.

Section 6.1 and Section 6.2 explore the hardware and software components of the NeuroBreakoutBoard, followed by a comprehensive case study to demonstrate the platform's unique functionalities in Section 6.3. The chapter concludes with a discussion on the platform's limitations and potential future enhancements. This chapter summarizes the contributions detailed in [177].

6.1 Hardware

In this section, we detail the hardware components and the circuitry implemented for the NeuroBreakoutBoard as shown in Figure 6.1 (a). The signal path of the NeuroBreakoutBoard is divided into three main parts: signal generation, routing, and sensing. Additionally, the board provides two hardware interfaces. The first interface, termed the NVM interface, facilitates easy integration of a wide variety of memristive crossbar arrays. The second interface allows for the connection of a controller to the board.

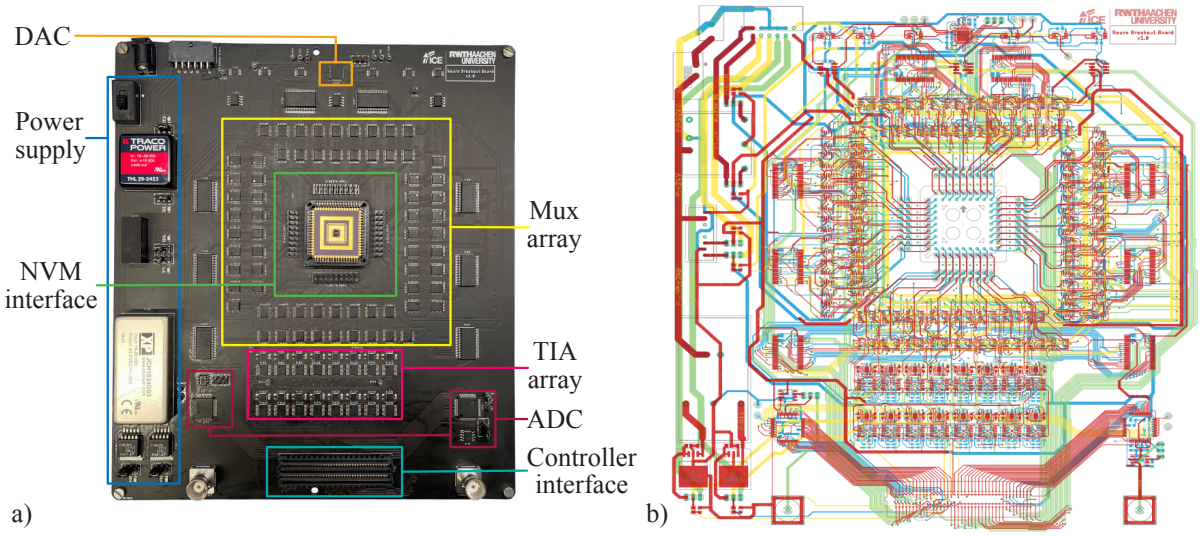


Figure 6.1: Overview of the NeuroBreakoutBoard: (a) image of the manufactured Printed Circuit Board (PCB) and (b) multi-layer layout.

6.1.1 Signal Generation

The signal generation module of the NeuroBreakoutBoard, as shown in Figure 6.2 (a), comprises a Digital-to-Analog Converter (DAC), an analog switch, and an amplifier circuit. The DAC is responsible for providing the voltages required for writing, reading, and computing in the crossbar array. The component is managed via a Serial Peripheral Interface (SPI) bus and features eight channels, each offering a 12-bit resolution across multiple voltage ranges: 0 V to 5 V, 0 V to 10 V, ±5 V, ±10 V, and ±2.5 V, all powered by an internal high-precision reference [12]. Out of these, five channels are dedicated to providing the interconnection matrix (refer to Section 6.1.2) with the required V_{SET}, V_{RESET}, V_{Read}, V_X, and V_{Gate} voltages, while the remaining three channels are connected to a pin header for additional uses.

Given that the writing algorithms of memristive devices often rely on pulsed signals, a simple but efficient pulse generator, using an analog switch with a switching time of $t_{ON} = 60\,\text{ns}$ at ±5 V, has been implemented [14]. This setup, comprising five analog switches, is linked to the controller interface, enabling independent, rapid, and precise pulse generation.

While read and write operations within a crossbar array typically require minimal current, the current demand significantly increases during analog vector/matrix multiplication operations, especially as the dimensions of the crossbar array expand. To accommodate currents up to several milliamperes, which exceed the DAC's output capability, the NeuroBreakoutBoard incorporates load drivers placed between the pulse generator and the interconnection matrix. These load drivers, built from operational amplifiers, are capable of handling a maximum current of 50 mA at 15 V [7].

In summary, the signal generation module of the NeuroBreakoutBoard is designed for versatility, allowing for the adjustment of pulse amplitudes and durations. This

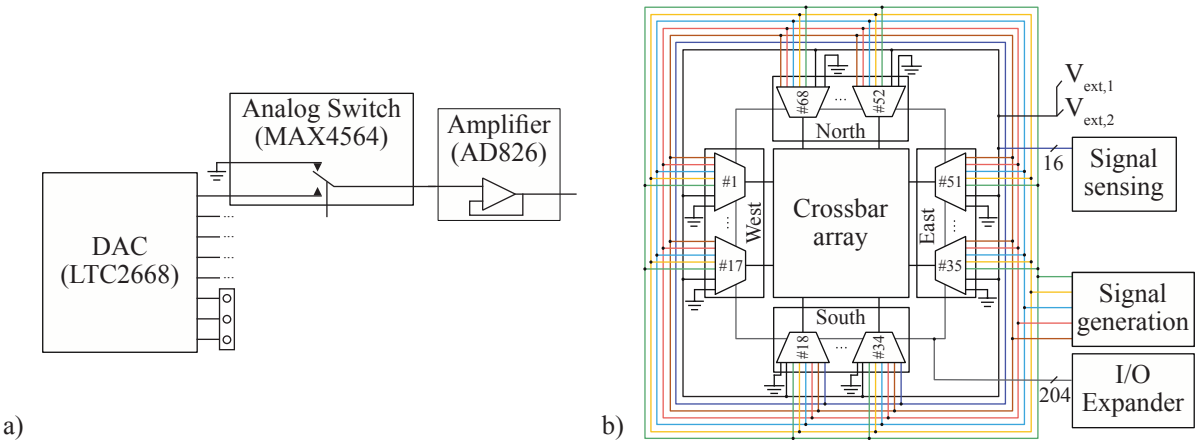


Figure 6.2: Overview of (a) the signal generation module, comprising the DAC and pulse generator, and (b) the interconnection matrix that links the signal generation/sensing modules to the NVM interface.

feature not only supports standard read/write operations but also facilitates CIM tasks within memristive crossbar arrays by driving the necessary currents effectively.

6.1.2 Flexible Interconnection Matrix

The signal generation module connects to the NVM interface via a flexible interconnection matrix, enabling arbitrary mapping of input pulses to the crossbar array through the use of 68 precision analog multiplexers [8]. Figure 6.2 (b) illustrates the architecture of the interconnection matrix, including its links to the signal generation and sensing modules. Each multiplexer can route one physical line of the crossbar to eight distinct signals. The V_{SET}, V_{RESET}, and V_{Read} signals facilitate programming and reading a memristive cell. V_X is versatile, serving either to apply $V/2$ to non-selected cells in a passive array or to set any chosen potential for computing voltage in LIM operations. Additionally, V_{Gate} is connected to the gates of access transistors in 1-Transistor 1-Resistor (1T1R) arrays. However, these signals can be arbitrarily assigned to any potential, with the only limitation being the DAC's capabilities within the signal generation module. The $V_{\text{ext,1}}$ and $V_{\text{ext,2}}$ signals are connected to a pin header to serve as probe lines, allowing external measurement devices to connect to the crossbar array and enhance the NBB's measurement capabilities. To operate the multiplexers, each requiring three control bits for signal selection, results in a total of 204 control wires. By utilizing an I/O expander, the typically limited number of digital output signals of a microcontroller is accommodated. Multiplexers at the interconnection's south end are unique in that one line connects to the signal sensing module, allowing for 16 sensing lines to be exclusively mapped to these specific multiplexers on the crossbar interface's south side.

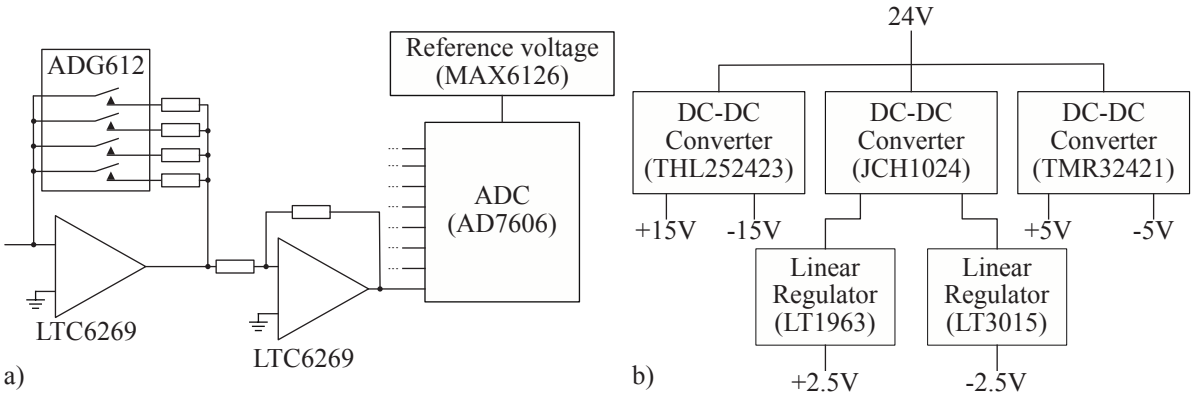


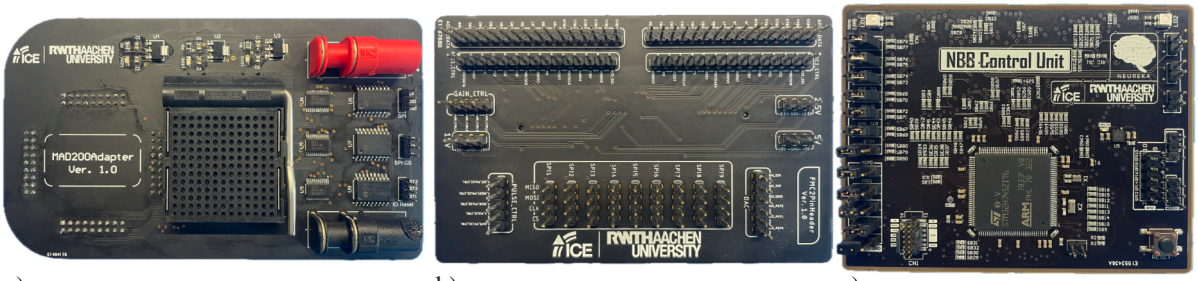
Figure 6.3: Overview of (a) the signal sensing module, which includes a Transimpedance Amplifier (TIA) connected to the Analog-to-Digital Converter (ADC), and (b) the power supply module that offers three distinct power levels.

6.1.3 Signal Sensing

To measure the resistance of a memory cell or the result of a CIM operation, it's necessary to determine the current flowing along a bit line. Therefore, we implemented the signal sensing module which offers a broad measurement range while maintaining a high accuracy. The sensing module consists of a TIA [13] circuit connected to an ADC[6], as depicted in Figure 6.3 (a). Since the ADC is limited to only convert voltages, the implemented TIA circuit translates the cumulative current from the crossbar array's bit lines into a measurable voltage. Given the varying resistances of memristive devices due to their switching processes, the TIA incorporates a dynamically adjustable feedback resistor. This feature allows for the measurement of a wide spectrum of resistances with precise accuracy. The feedback resistor of the TIA can be adjusted via an analog switch, with values ranging from 43 Ω to 100 kΩ [9]. Furthermore, an amplifier is employed to amplify the converted voltages to fit into the measurable range of the ADC. The ADC has eight parallel channels, each with an 18-bit resolution, capable of simultaneous sampling. In total, the NBB facilitates the measurement of 16 parallel channels while the results can be transmitted to the controller through either an SPI bus or a parallel interface. To increase the accuracy of the ADC, a dedicated voltage reference Integrated Circuit (IC) featuring a high precision of up to 0.02% [15]. Additionally, an external voltage reference can be provided via a BNC connector.

6.1.4 Power Supply

To accommodate a wide range of memristive devices, the NBB is capable of supplying a wide set of voltages for both its internal modules and the crossbar array. Figure 6.3 (b) depicts the NBB's power supply, which is designed to operate with a standard 24 V power adapter which ensures an independent usage of the measurement platform from external laboratory equipment. Nevertheless, the NBB also includes an



a) b) c)

Figure 6.4: Extension boards: (a) adapter board linking to the NVM interface of the NBB, providing a dedicated socket and additional external digital I/Os, (b) breakout board for direct access to all control lines connected to the control interface, and (c) controller unit featuring a microcontroller that executes the implemented firmware.

additional socket for connecting external high-precision voltage sources. The PCB supports three distinct voltage domains: −2.5 V to 2.5 V, −5 V to 5 V, and −15 V to 15 V, each supplied by dedicated DC-DC converters, enabling a versatile power supply for various memristive device requirements [191, 190, 11, 10, 208].

6.1.5 Non-Volatile Memory (NVM) Interface

The NVM interface primarily consists of four pin headers located at the center of the NBB, as depicted in Figure 6.1 (a). This interface facilitates straightforward connections to adapter boards, thus enabling the integration of additional circuitry and their corresponding sockets. Figure 6.4 (a) presents an adapter board designed for a memristive crossbar housed in a PGA100 package. Notably, the adapter board provides extra digital I/O pins, which are instrumental in managing the digital control logic embedded in the chip under test. The adapter board's I/O expanders are linked to the NBB's controller via external wires, ensuring the synchronization of all input signals to the chip. The interconnection matrix significantly enhances flexibility in signal routing. Utilizing the NVM interface in conjunction with specially designed adapter boards allows the NBB to interface with any chip package and pin configuration available. However, the application of the NVM interface is constrained by the configuration of the sensing module. Given the limited number of parallel sensing channels—specifically, 16—these independent channels are restricted to routing exclusively to the north pin header of the NVM interface. Nonetheless, the interconnection matrix provides two additional sensing wires, which can be utilized to connect with external measurement equipment.

6.1.6 Platform Orchestration

While the NVM interface serves as a universal abstraction for connecting to crossbar arrays, the controller interface acts as an abstract interface for managing the NBB platform. An FPGA Mezzanine Card (FMC) connector bundles control lines from all

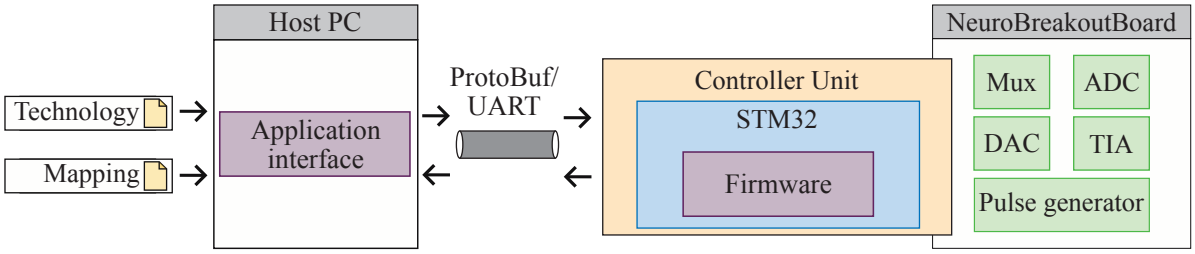


Figure 6.5: Overview of the system architecture: the NeuroBreakoutBoard consists of hardware modules orchestrated by firmware running on the controller unit. This controller unit communicates via a UART/ProtoBuf [72] protocol with the provided application interfaces.

modules on the NBB, utilizing a standardized connector primarily employed in Field-Programmable Gate Arrays (FPGAs) to provide a vast array of I/O pins. To facilitate easy integration with various development platforms, an FMC breakout board has been developed.

This breakout board, depicted in Figure 6.4 (b), allocates specific pins for each SPI interface of the I/O expanders, as well as the DAC and ADC. It also includes the ADC's parallel interface, complete with all control pins, enabling full access to the ADC's diverse functionalities. Additionally, gain control pins are linked to analog switches that adjust the feedback resistances of the TIAs, as well as the pulse control pins responsible for initiating pulses during read, write, and compute operations.

Figure 6.4 (c) illustrates a control unit equipped with a STM32 microcontroller from STMicroelectronics [181], directly connected to the controller interface via the FMC connector at the PCB's bottom side. The microcontroller interfaces with all components through the SPI bus, while employing the ADC's parallel interface for enhanced performance during repetitive measurements. The unit is designed for flexibility, featuring pin headers and jumpers for straightforward modification of fixed configuration pins and includes status LEDs to signal the measurement process or detect errors. Moreover, the control unit is equipped with a Joint Test Action Group (JTAG)/Serial Wire Debug (SWD) connector for programming the microcontroller and a Universal Asynchronous Receiver/Transmitter (UART) bus interface for communication with the host PC.

6.2 Software

This section discusses the specifics of the software implemented on the control unit and the host PC, designed to conduct measurements and execute CIM operations on physical crossbar arrays. The architecture of the overall system is depicted in Figure 6.5, showcasing the NeuroBreakoutBoard with the integrated control unit connected to a host PC via a UART bus. Communication between the control unit and the host PC employs the Protocol Buffers framework [72], which serializes data in a

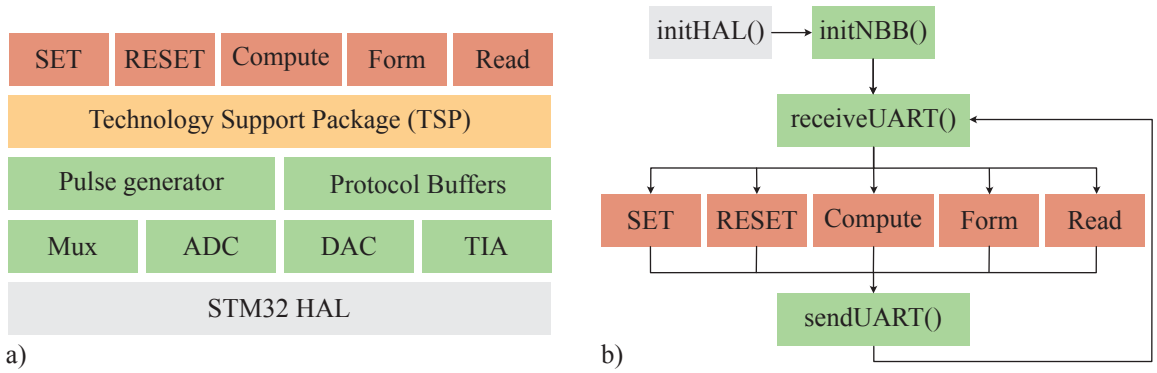


Figure 6.6: NeuroBreakoutBoard firmware: (a) software modules implemented to offer various abstraction levels for interfacing with the hardware modules, and (b) flowchart depicting the main loop.

platform-independent manner. The application interface, executed on the host PC, is developed in Python and boasts a robust Application Programming Interface (API) that facilitates the execution of experiments. Additionally, it supports the incorporation of more sophisticated tools, enhancing its functionality for advanced experimental setups.

6.2.1 Firmware

The firmware architecture, as shown in Figure 6.6 (a), is organized in layers and developed in ANSI-C. The STM32 Hardware Abstraction Layer (HAL) offers extensive APIs for interfacing with the microcontroller's peripherals, simplifying the development of user applications through efficient management of communication peripherals, data transfers, interrupts, Direct Memory Access (DMA), and error handling. Moreover, HAL incorporates runtime failure detection to improve firmware robustness and facilitate debugging. On top of HAL, software modules abstract the NeuroBreakoutBoard's peripherals, providing interfaces for controlling the pulse generator, multiplexers, ADCs, DACs, TIAs, and for encoding/decoding protocol buffer messages, which are elaborated in the following.

Pulse generator: The pulse generator is designed to produce the necessary pulses for read, write, and compute operations. The pulses' amplitude is adjusted by the DAC, while the generator itself uses an analog switch activated by a digital input. These inputs are directly linked through the control interface to the STM32's General-Purpose Input/Output (GPIO) pins. The HAL library facilitates interaction with the GPIOs using the `HAL_GPIO_WritePin()` function, and the duration of the pulses is controlled by a hardware timer in the STM32, which triggers an interrupt event.

Protocol buffers: Protocol Buffers (ProtoBuf) is an open-source project that aims to provide a framework for serializing structured data that is both language and platform-independent [72]. Within our measurement platform, ProtoBuf is employed to serialize data for transmission over UART between the host PC's instrumentation

application and the STM32. To utilize ProtoBuf, a proto file is required to outline the structure and size of the messages sent across the bus. The proto files, detailed in Appendix B, define operation, request, and response messages, facilitating the exchange of all necessary information to execute operations on the NBB and relay measurement results back to the host PC.

Multiplexer: The interconnection matrix consists of 68 multiplexers, managed by a total of 204 control signals connected to I/O expanders. These I/O expanders are addressed over a SPI bus allowing a straightforward orchestration of the whole interconnection matrix. The STM32 HAL library provides the `HAL_SPI_Transmit()` function, enabling message transmission over the SPI bus interface. Beyond enabling the interaction with I/O expanders, the multiplexer module also introduces a mapping function. This function allows for the configuration of any pin within the interconnection matrix to connect to an output pin of the DAC, significantly simplifying algorithm implementation at a higher abstraction level.

Analog-Digital Converters (ADCs): ADCs serve as the core of the sensing module, offering both parallel and serial interfaces for data transmission to the control unit. With the SPI controllers of the STM32 microcontroller already occupied by the I/O expanders and the DAC, the ADC's parallel interface connects to GPIO pins for data transmission. The module features an `adcInit()` function for ADC and GPIO setup. The `calculateResistance()` function computes the resistance of a memristive device as follows:

$$R_{\text{cell}} = \frac{3V_{\text{Read}}}{V_{\text{ADC}}R_{\text{Feedback}}}, \tag{6.1}$$

where R_{cell} represents the memristive cell's resistance, V_{Read} the read voltage, V_{ADC} the TIA's output voltage, and R_{Feedback} the TIA's feedback resistance. Additionally, the `calculateCurrent()` function determines the cumulative current along a specified bit line, essential for analog CIM operation.

Digital-Analog Converters (DACs): The DAC, pivotal for generating the required pulses for reading, writing, and computing, connects to the microcontroller via the SPI bus. The DAC firmware module enables users to set the reference voltage range and modify output voltages. Given the microcontroller's limited SPI interfaces, the DAC shares an SPI module with the I/O expanders, necessitating the setting of the respective chip select pin connected to a GPIO pin on the microcontroller before message transmission.

Before setting the actual output voltage, the DAC's voltage reference must be configured. The `setSoftSpanRange()` function sends the appropriate SPI message to adjust the DAC's reference voltage. The `setupDACVoltageByChannel()` function then establishes the desired output voltage for a specific channel.

Transimpedance Amplifiers (TIAs): The transimpedance amplifier module regulates the feedback resistance, determining the sensing module's measurement resolution. The feedback resistances are connected to analog switches, managed by four digital control bits. On the NBB, these control bits for all sixteen TIAs are routed in parallel and mapped to the controller interface, setting the feedback resistance uniformly

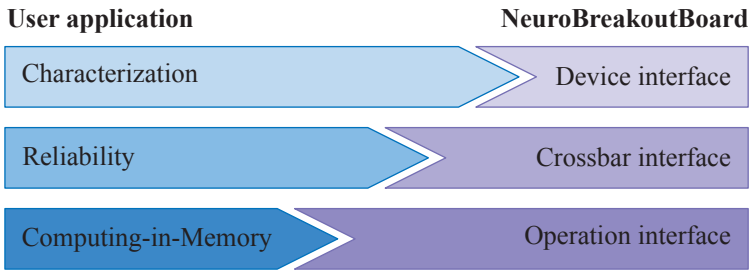


Figure 6.7: Application interfaces: the provided application interfaces facilitate interaction with the NeuroBreakoutBoard across three distinct abstraction levels, enabling everything from device characterization to the execution of CIM operations.

across all TIA modules. The module includes the `setTIAFeedbackResistance()` function for adjusting the feedback resistance of the TIA modules.

Technology Support Package (TSP): Following Figure 6.6 (a), situated above the core firmware modules, the TSP incorporates routines specific to various technologies. Memristive devices, for example, necessitate distinct writing schemes to enhance their reliability and extend their lifespan. Consequently, the TSP is tailored for each type of memristor technology, refining the foundational access functions at this level. The TSP offers an interface comprising five functions, enabling the mapping of both memory and CIM operations on crossbar arrays. This flexibility not only broadens the measurement platform's range of applications but also allows for extensive customization to accommodate various memristive device technologies.

The primary routine of the NBB firmware is outlined in Figure 6.6 (b). Upon the platform's startup, the firmware initializes the STM32 HAL library, followed by the configuration of all implemented modules. Specifically, multiplexers are set to their initial states, and both the ADC and DAC are reset. The main loop includes a blocking `receiveUART()` function call, waiting on a ProtoBuf message from the host PC. After decoding the message's contents, the corresponding function is executed. The response message, consisting of a status flag alongside the resistances or currents measured by the sensing module, is then transmitted.

6.2.2 Application Interfaces

The application interface, designed to run on a host PC, is developed in the platform-independent programming language, Python. This interface connects to the NBB firmware as depicted in Figure 6.6 (a), offering a comprehensive middle layer for conducting both device/crossbar measurements and executing analog/digital CIM operations. The application interface, shown in Figure 6.7 provides three distinct abstraction layers through which external applications can interact with the NBB, facilitating a unique method to interact with neuromorphic crossbar arrays. This approach enables early adoption and seamless integration into the software development lifecycle. The specifics of these interfaces are elaborated in the following.

Device interface: The device interface serves as the most fundamental layer, designed to facilitate direct interaction with memristive devices within a crossbar array. The `formCell()` function executes the forming process essential for initializing a memristive cell. Given the variety of memristive devices, this routine may be adjusted to align with the specifications of the memristive crossbar array connected to the NeuroBreakoutBoard. Currently, we have incorporated the Incremental Form and Verify (IFV) algorithm, which enhances the switching characteristics and post-forming yield, as highlighted in [75]. The IFV algorithm progressively increases the forming voltage by a given ΔV_{form}, conducting a read operation after each pulse. The process continues until the cell resistance meets a predefined target current i_{target}.

Subsequent to forming, the `readCell()` function facilitates cell reading, whereas SET/RESET operations are managed by the `setCell()` and `resetCell()` functions, respectively. The write functions employs the Incremental Step Pulse with Verify Algorithm (ISPVA), adopting a similar write/verify approach as the IFV algorithm. Notably, the read function leverages an automatic feedback resistance algorithm to select the TIA's feedback resistance based on the measurement outcome. This algorithm iteratively adjusts the feedback resistor, in case the operational amplifier is saturated, by choosing the next feedback resistance and recursively invoking itself.

Device addressing is managed through a configuration file that maps the physical chip pins to the corresponding pins of the NeuroBreakoutBoard's interconnection matrix, which must be manually created before employing the instrumentation application.

In addition to the provided functions, the device interface offers an advanced logging feature that records all executed commands for each device, providing a comprehensive history of the cells within the crossbar array. This data is stored in a platform-independent CSV file, encapsulating all necessary parameters to replicate operations and analyze Cycle-to-Cycle (C2C) variations, Device-to-Device (D2D) differences, and the yield before and after forming.

Crossbar interface: Compared to the device interface, the crossbar interface encompasses `read()`, `set()`, `reset()`, and `form()` functions that operate on an entire crossbar array. These are crucial for initializing the crossbar for various experiments, including analog/digital CIM operations. Inputs to these functions are matrices corresponding to the crossbar's dimensions, efficiently utilizing the underlying device interface for executing write and read operations. Therefore, all executed actions are automatically tracked in the provided database provided by the device interface.

Operation interface: The operation interface represents the highest level of abstraction, enabling the execution of CIM operations on a crossbar array. Given that memristive crossbar arrays can process data in both digital and analog modes, this interface is separated into two distinct modules.

For conducting analog vector/matrix multiplications within a crossbar structure, the matrix $\boldsymbol{B}$ must be encoded as conductance values across the memristive devices, with the input vector $\boldsymbol{a}$ applied as voltages across the word lines. This generates an output vector $\boldsymbol{c}$, represented by the cumulative current along the bit lines. Assuming $a_i \in \{0,1\}$ and $B_{i,j} \in \{0,1\}$, inputs are converted from $\{0,1\} \rightarrow \{0\,\text{V}, V_{\text{read}}\}$ and

matrix values from $\{0,1\} \rightarrow \{G_{\text{min}}, G_{\text{max}}\}$. The resulting vector undergoes a reverse conversion through the `demap()` function, which adjusts each value according to

$$c_m = \frac{1}{V_{\text{read}}(G_{\text{max}} - G_{\text{min}})}(I_{\text{BL},m} - V_{\text{read}} G_{\text{min}} \sum_{0}^{N-1} a_{B,n}), \tag{6.2}$$

where c_m indicates the converted result for bit line m, $I_{\text{BL},m}$ the total current for bit line m, N the row count, and V_{read} the applied read voltage. Currently, this module supports binary vector matrix multiplication only, with further quantization methods requiring additional mapping algorithms.

Executing LIM operations necessitates selecting and implementing a suitable logic family, each significantly differing in operation. The operation interface supports this process by providing utility functions and maintaining a predefined data structure for tracking input and output devices, along with compute voltages and the physical representations of the Boolean values.

Additionally, this interface expands the device interface's tracking capabilities to encompass comprehensive logging of CIM operations. Given the potential engagement of multiple or all cells in a single operation, a specialized operation log file is created and linked to the device-specific history files. This log file records operation types, compute voltages, and result values, offering a detailed account of the experimental procedures and results.

6.3 Case Study: Reliability Assessment of a Commercial ReRAM Technology

This case study aims to showcase the versatility of the NeuroBreakoutBoard by examining the reliability of a commercially available Resistive Random-Access Memory (ReRAM) technology node.

We designed and taped out memristive memory composed of a 1T1R crossbar structure. The memory cell comprises a Metal-Insulator-Metal (MIM) stack, functioning as the memristive device, alongside an access transistor serving as the selection device. The MIM stack consists of top and bottom electrodes, each approximately 150 nm thick, made of TiN, a scavenging layer of about 7 nm Ti, and a dielectric switching layer of 8 nm HfO_2, resulting in a total MIM stack area of 600 nm $\times$ 600 nm. The scavenging layer undergoes oxidation during the memristive device's forming process, introducing oxygen vacancies into the MIM stack and creating a conductive filament within the dielectric layer. The conductive filament's internal structure, manipulated by applying SET or RESET voltages, determines the cell's resistance. The MIM stack is positioned between the second and third metal layers in the Back End Of Line (BEOL) phase of the fabrication process. The accompanying access transistor is manufactured using 130 nm Complementary Metal-Oxide Semiconductor (CMOS) technology, featuring gate dimensions of 130 nm $\times$ 150 nm [153, 146, 155]. While the crossbar array is designed with dimensions of 12 $\times$ 7, this study focuses on a 7 $\times$ 7

Table 6.1: Overview of parameters for the ISPVA and the IFV algorithm.

State	V_{Gate} [V]	I_{Target} [µA]	V_{Start} [V]	V_{Max} [V]	V_{Read} [V]	V_{Step} [V]	T_{Pulse} [µs]
Forming	1.50	30.00	2.00	5.00	0.20	0.01	1.00
HRS	3.30	1.00	0.50	3.00	0.20	0.10	10.00
LRS1	0.50	10.00	0.50	3.00	0.20	0.10	10.00
LRS2	0.70	20.00	0.50	3.00	0.20	0.10	10.00
LRS3	0.90	30.00	0.50	3.00	0.20	0.10	10.00

submatrix. The word, bit, and gate lines are directly connected to the chip's pads and wire-bonded to a Quad Flat Package (QFP). An adapter board, tailored for the chip's socket, links directly to the NVM interface of the NeuroBreakoutBoard.

In the following, we assess the memristive crossbar structures for their ability to perform digital and analog CIM operations. Initially, we evaluate the yield before and after forming, switching endurance, and binary and multi-level switching characteristics of the fabricated memory cells. Upon investigating these properties, we explore the feasibility of executing CIM operations. This involves conducting a vector/matrix multiplication to observe the effects of faults on computational results. Finally, we investigate the crossbar array's susceptibility to faults through the execution of logic operations, further elucidating the robustness and versatility of these memristive systems.

6.3.1 Manufacturing Yield

Typically, a memristive device is anticipated to be in the High Resistive State (HRS) before the forming process. Milo et al. [146] define the manufacturing yield of memristive devices as the percentage of cells exhibiting a read current below 1 µA prior to forming. We assessed the resistance before and after forming for each device within the 7×7 crossbar array of five chips, employing the IFV algorithm, with parameters detailed in Table 6.1.

Figure 6.8 displays the number of dysfunctional memory cells (a) before and (b) after the forming process. The individual yields of the chips were 85.71%, 83.67%, 87.76%, 81.63%, and 95.92%, leading to an average yield of 86.94%. While Milo et al. [146] categorize a cell as formed and functional if it exhibits a read current greater than 18 µA post-forming, we adopt a functional criterion based on a cell's ability to execute at least 50 consecutive switching operations between the HRS and the Low Resistive State (LRS). For instance, Figure 6.8 compares the current measurements post-write operation of a dysfunctional (c) and a functional (d) cell. All read operations were performed using a 200 mV read voltage, where a low current indicates the HRS, and a high current indicates the LRS. The dysfunctional cell managed three con-

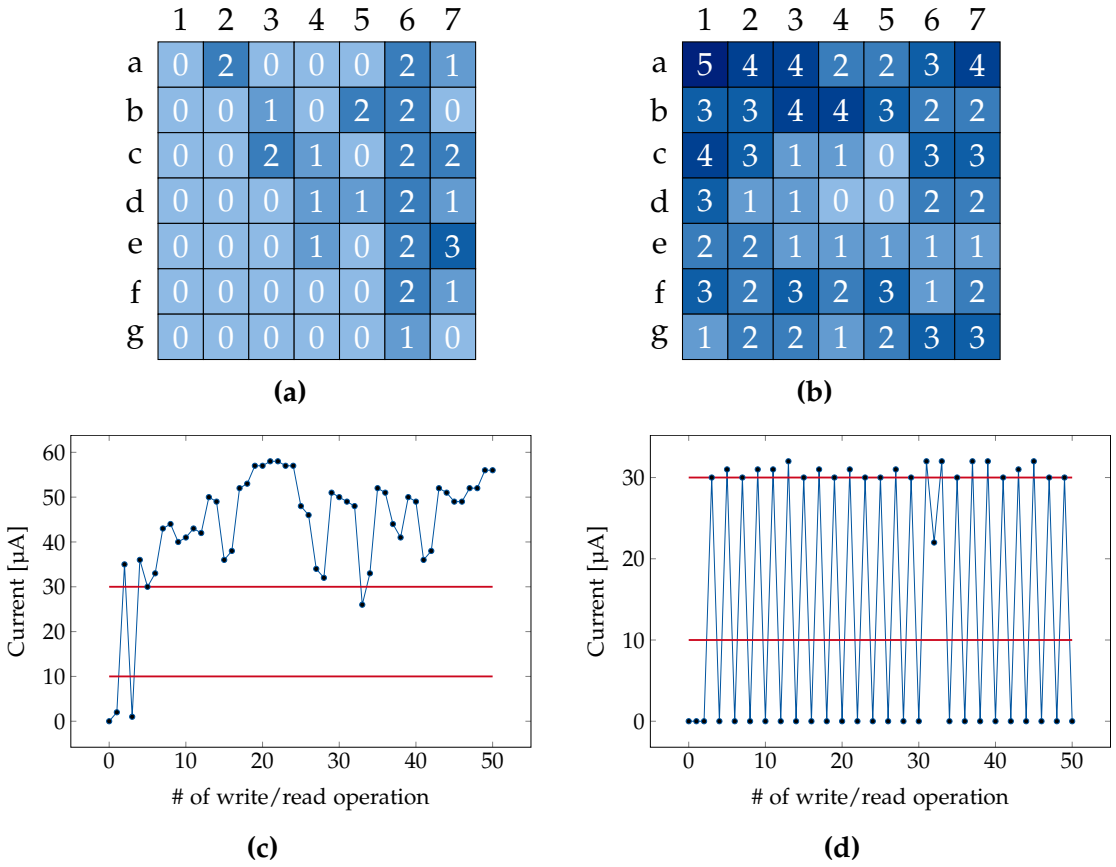


Figure 6.8: Analysis of the manufacturing yield: heatmap depicting the number of dysfunctional memory cells (a) prior to and (b) following the forming process across 5 chips. Comparison of switching characteristics between a (c) dysfunctional and (d) functional memory cell.

secutive switches before it failed, exhibiting a stuck-at LRS fault behavior. Conversely, the functional cell demonstrated stable switching throughout the experiment.

Overall, our findings indicate that a significant number of cells, both before and after forming, are prone to a stuck-at LRS fault. Specifically, we noticed cells with low resistance before forming that did not alter their internal state despite repeated forming attempts. Although correctly formed, some cells could only switch between the HRS and LRS a limited number of times before irreversibly transitioning to the LRS state. In rare instances, this failure occurred after a single reset operation.

6.3.2 Programming Characteristics

After identifying the number of functional cells, we investigate the programming characteristics directly influencing the feasibility and accuracy of CIM operations. To perform a SET or RESET operation, the device interface of the NeuroBreakoutBoard is utilized implementing ISPVA (see Section 6.2.2). The respective parameters are given in Table 6.1.

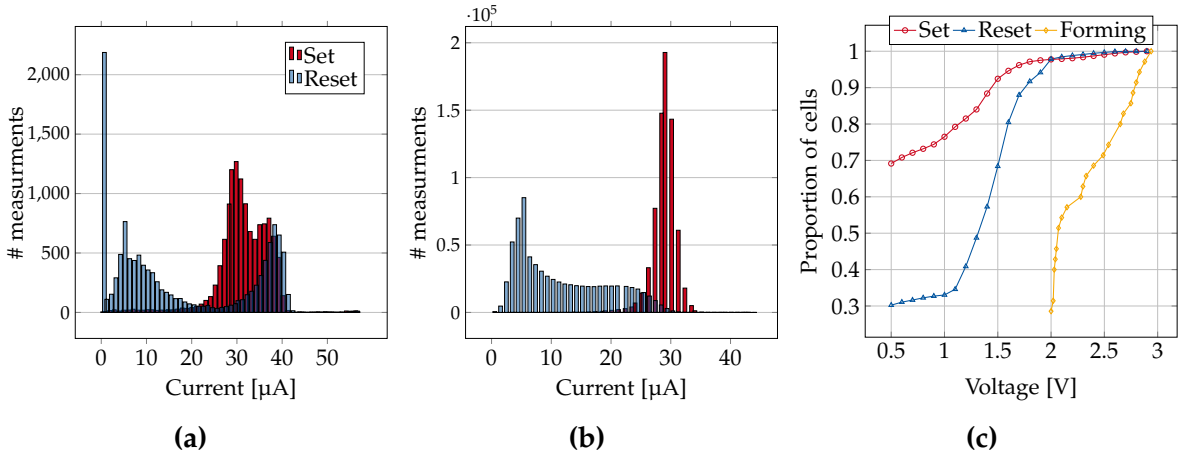


Figure 6.9: Binary switching characteristics: (a-b) distribution of the HRS and the LRS across two distinct cells. (c) Cumulative Distribution Function (CDF) analysis for SET, RESET, and forming voltages.

Binary states: Initially, the binary switching characteristics is investigated, focusing on the HRS and the LRS3 as the two distinct states. Figure 6.9 (a-b) presents histograms of consecutive alternating SET/RESET operations followed by a read operation for two different cells, highlighting C2C and D2D variability. *While both cells generally exhibit the HRS around 10 µA and the LRS at 30 µA, the cell depicted in Figure 6.9 (a) demonstrates significant overlap between these states due to two phenomena. Occasionally, although the RESET operation as indicated by the ISPVA algorithm appears successful, the subsequent read operation inadvertently flips the cell back to its previous state. Additionally, a substantial number of cells display stuck-at faults, leading to insignificant or no changes in resistance following the write algorithm. Overall, variability is more substantial in the HRS compared to the LRS.*

Figure 6.9 (c) depicts the CDF of the SET, RESET, and forming voltages for all characterized cells, underscoring the D2D variability observed during programming and forming. *The SET process demonstrates the greatest stability, with over 90% of cells successfully switching at 1.5 V. The CDF for forming voltages highlights the need for an incremental forming algorithm, requiring a forming voltage between 2 V and 3 V to successfully form 90% of the memory cells. The RESET operation, however, exhibits significant variability, ranging from 0.5 V to nearly 3 V, with the lower bound particularly concerning given the typical selection of reading voltages between 0.2 V and 0.5 V.*

Multi-level states: The ability to program memristive devices into multiple levels offers a significant advantage over traditional memory technologies by increasing memory density and enhancing the accuracy of analog vector/matrix multiplication.

The memristive device under study can be programmed into four states (HRS, LRS1, LRS2, and LRS3) by adjusting I_{Target}, as detailed in Table 6.1. However, the inherent variability in memristive devices poses a challenge to multi-level programming by causing state overlaps, making them indistinguishable. To assess the feasibility of achieving distinct states with the provided devices, we cycled the devices through

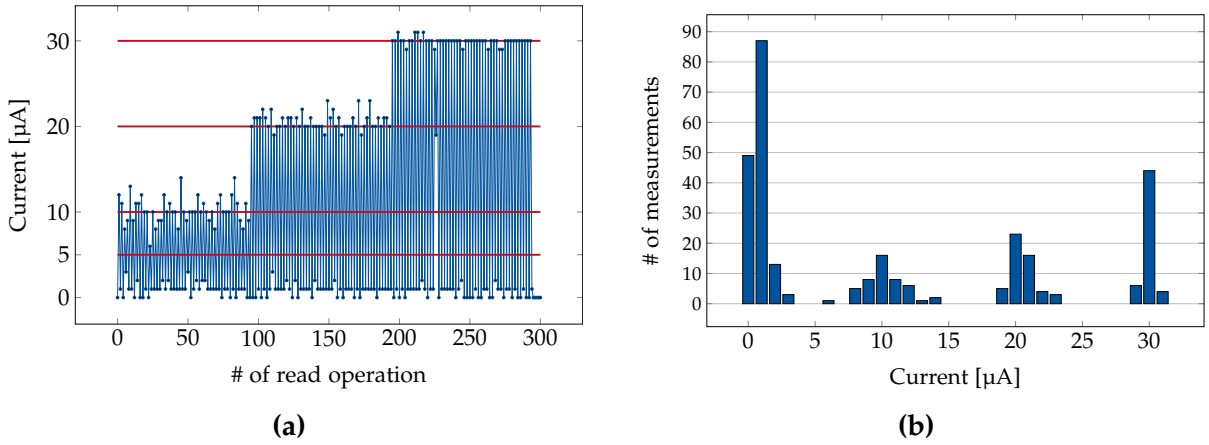


Figure 6.10: Multi-level switching characteristics: (a) repetitive cycling through all applicable resistance states and (b) histogram depicting the variability of states.

each state 50 times to examine the impact of variability on the different states. Figure 6.10 (a) shows the switching characteristics over 300 programming cycles, with red lines marking I_{Target} for each state. *Generally, LRS1 exhibits considerable variability, sometimes nearly merging with the HRS, attributed to the narrow gap between HRS, defined at a maximum current of 5 µA, and LRS1, set at 10 µA. By contrast, LRS3 shows notably less variation, establishing it as the most stable state, which is why it was selected for binary operation. Figure 6.10 (b) features a histogram of the four states for all measured devices, reinforcing the observation that LRS1 and LRS2 exhibit higher variability compared to LRS3 and HRS.*

6.3.3 Endurance Characteristics

While the programming characteristics impact the precision of operations executed on memristive crossbar arrays, the endurance determines the lifespan of a device, or the number of operations it can perform before failure. Hence, in this section, we delved into the switching endurance of the provided memristive devices, with a focus on contrasting the behaviors of two specific memory cells.

Figure 6.11 (a) displays the programming voltage from the ISPVA for the first device. *Overall, the device sustained approximately 50,000 cycles, equally divided among 12,500 SET and RESET operations, with 25,000 intermediate read operations. A sudden increase in the read current indicates the device's failure, becoming permanently stuck in the LRS.*

To offer a detailed examination of the device behavior over time, Figure 6.11 (b) and (c) depict the cell state after SET (LRS) and RESET (HRS) operations. *While the LRS exhibits reasonable stability, the HRS shows increased variability.* Typically, for the HRS to be effectively distinguishable from the LRS, it should be below 5 µA. *However, for this cell, the HRS frequently overlaps with the LRS, complicating its utility for data storage and CIM operations.*

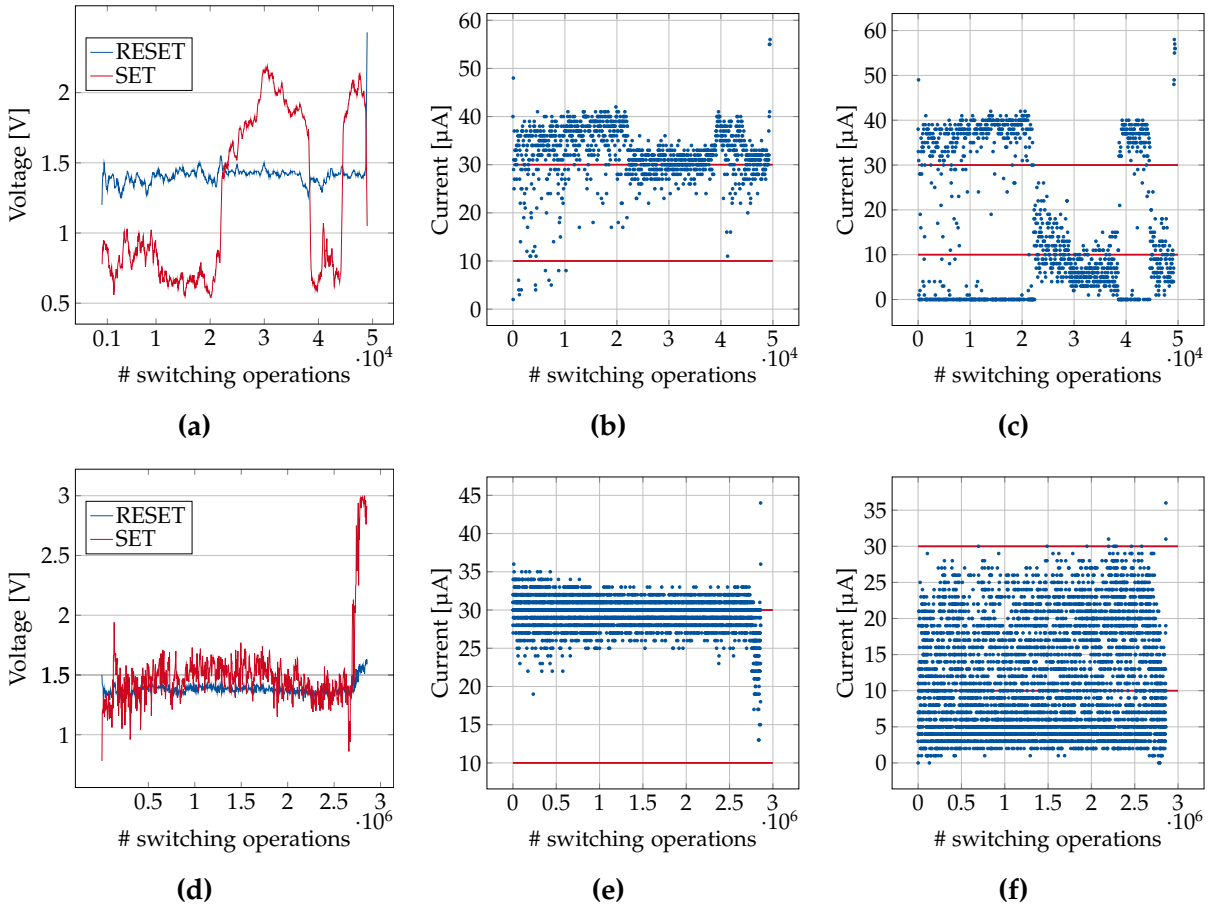
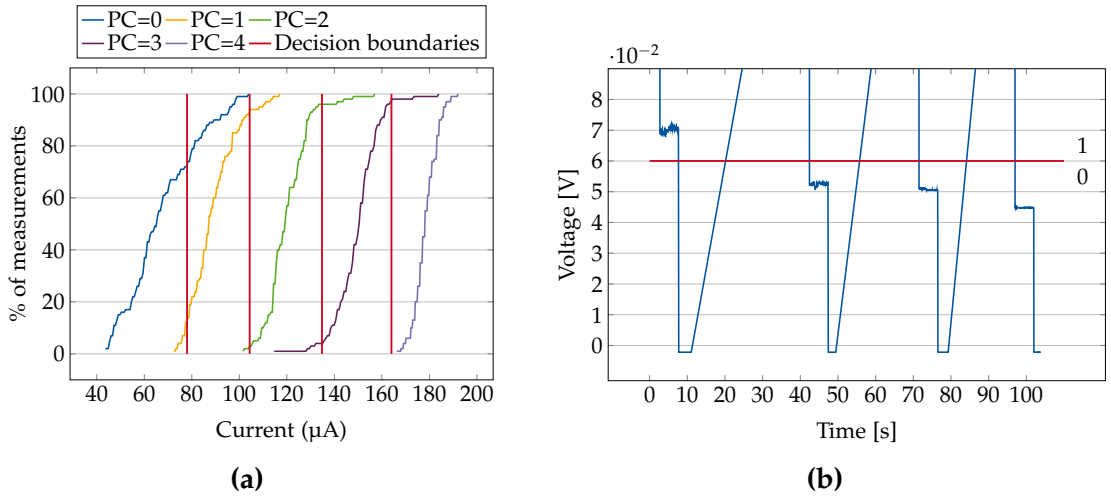


Figure 6.11: Switching endurance of two memory cells: (a/d) programming voltage for SET/RESET operations and programmed state after (b/e) SET and (c/f) RESET operations.

In comparison, the second cell significantly exceeds the first in terms of cycle count, nearly by two orders of magnitude, as illustrated in Figure 6.11 (d). Similar to the first, the LRS of this second cell displays high stability, but fluctuations in the HRS compromise the device's suitability for both memory and CIM applications. Notably, the RESET voltage of the second cell, as shown in Figure 6.11 (d), seems more consistent than that of the first cell, suggesting that the instability lies within the HRS, which unintentionally performs a SET operation while its current resistance is being measured.

6.3.4 Computing-in-Memory (CIM)

Leveraging its flexible interconnection matrix and signal generation/sensing modules, the NeuroBreakoutBoard excels in executing CIM operations on memristive crossbar arrays. As detailed in Section 2.2, CIM operations can be categorized into two types: analog CIM and binary LIM operations. In this section, we evaluate the reliability of both types of operations in the context of the nonidealities inherent in memristive devices.



(a) **(b)**

Figure 6.12: Resilience of CIM operations: (a) CDF of analog CIM operations and (b) digital NOR operation executed on the NeuroBreakoutBoard.

Analog Computing-in-Memory (CIM): The analog CIM approach involves performing Multiply–Accumulate (MAC) operations that require an input vector and a corresponding matrix. For this experiment, the input vector consists of four elements, and the matrix dimensions are set to 4×7, constrained by the number of functional cells. Throughout the experiment, we dynamically reprogram the matrix values, represented by the resistance of the memristive devices, with random values. Meanwhile, the input vector is fixed at $\mathbf{v}_{\text{in}} = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}^{\text{T}}$ to ensure comprehensive coverage of all possible output values. Generally, a binary MAC operation involves performing an XNOR operation between the input vector and each matrix column, followed by a Popcount (PC) function that counts the number of 1s in the result vector, representing the MAC operation's outcome.

Figure 6.12 (a) displays the CDF of the accumulated bit line currents, corresponding to the respective popcount values. Given that the input vector comprises four elements, the resulting popcount ranges from 0 (indicating all elements are 0s) to 4 (all elements are 1s). The decision boundaries for these counts are marked with red lines, while the colored traces represent the four distinct popcount outcomes derived from a total of 500 MAC operations. *Overall, the data suggests that the accuracy of the measured accumulated current aligns more closely with the correct result as the popcount increases. For example,* 100% *of the instances with a popcount of 4 were accurate, whereas for popcounts of 2 and 3, over* 90% *of the results were correct. However, the scenarios with the lowest popcount (indicating all elements are 0s) exhibited the highest incidence of errors. This finding is consistent with our previous experiments that indicated a higher error susceptibility of the HRS compared to the LRS.*

Binary Logic-in-Memory (LIM): For executing binary logic operations on memristive crossbar arrays, various logic families have been introduced, outlining the blueprints for constructing logic gates (refer to Section 2.2). Initially, our goal was

to implement Memristor-Aided Logic (MAGIC) and Memristor-Based Material Implication (IMPLY) gates on the provided crossbar array. However, the effectiveness of both logic families is significantly influenced by specific device characteristics, particularly requiring a dedicated ratio of $\frac{R_{\mathrm{OFF}}}{R_{\mathrm{ON}}}$ to the memristor's voltage thresholds $V_{\mathrm{T,OFF}}$ and $V_{\mathrm{T,ON}}$, as detailed in Equation (2.2). Unfortunately, the memristive devices available to us do not meet these criteria, making these logic families impractical for our purposes.

As a result, we opted to explore the reliability of LIM operations using a Memristor Ratioed Logic (MRL) NOR gate [56, 60, 55] with its working principle detailed in Section 2.2. Figure 6.12 (b) demonstrates the output voltage of the NOR gate in response to various input combinations $\{(0,0),(0,1),(1,0),(1,1)\}$, given a computation voltage of 0.2 V. The red line indicates the threshold used to differentiate between logical 1 and 0. *The findings validate the NOR gate's functionality, indicating an output of 0 exclusively for the input pair* $(0,0)$. *Moreover, the gap between the different output voltages is approximately 2 mV, signifying a robust decision margin, especially when considering the significant variability associated with the HRS.*

6.4 Limitations and Outlook

The NeuroBreakoutBoard serves as a flexible and versatile instrumentation platform designed to investigate the impact of memristive devices' inherent characteristics at the device, crossbar, and operation levels. It consists of hardware modules, firmware that operates on the microcontroller, and application interfaces executed on a host PCs, While these components are intricately linked to ensure the platform's easy use, they also introduce certain limitations. The hardware modules support up to 68 analog signals, with 16 of these being assignable to the signal sensing module. Consequently, the size of the crossbar arrays that can be tested is restricted. This constraint extends to the driver and sensing circuitry's maximum ratings, which limit the total current through a bit line and thus the number of rows that can be utilized in a single MAC operation. The firmware and application interfaces are built on a modular architecture. However, the UART/Protobuf communication protocol significantly restricts the overall performance of the NeuroBreakoutBoard. Endurance testing, requiring a comprehensive series of read/write operations, is particularly affected by this bottleneck, leading to extended testing durations. At first glance, these limitations might seem restrictive for experiments across all levels. Yet, the platform plays a pivotal role by facilitating early integration of real devices into the development of neuromorphic computing systems. Given the unique properties of memristive devices, incorporating real devices early is essential for developing reliable and secure computing platforms for the future. In the future, adding an FPGA to the NeuroBreakoutBoard would not only increase the platform's flexibility but also notable enhance its performance. Moreover, implementing LIM gates would greatly benefit from the capability to switch I/Os to a high impedance state, effectively isolating them from the crossbar array and thus reducing unintentional sneak path currents.

6.5 Synopsis

This chapter focuses on the development and capabilities of the NeuroBreakoutBoard, a versatile platform designed for characterizing and performing CIM operations on memristive crossbar arrays. The NBB's design encompasses hardware modules for signal generation, routing, sensing, and interfaces for the device, crossbar, and operation level. The provided case study showcases the NeuroBreakoutBoard's application in evaluating the reliability of a commercially available ReRAM technology, focusing on the manufacturing yield, the programming characteristics, and the execution of CIM operations. The chapter concludes by acknowledging the platform's limitations while emphasizing its crucial role in advancing neuromorphic computing through early real-device integration and potential future improvements.

Chapter 7

Conclusion

Neuromorphic computing emerges as a promising solution to the von Neumann bottleneck, which constrains the computing performance and energy efficiency of traditional computing systems. However, the realization of neuromorphic computing hinges on the development of novel devices that can seamlessly integrate computing and data storage within the same location, embodying the computing-in-memory (CIM) paradigm. Memristors, recognized as the fourth fundamental circuit element, stand as the cornerstone for Computing-in-Memory (CIM) yet grapple with reliability issues due to their immature development stage. To overcome the von Neumann bottleneck, neuromorphic computing must demonstrate its efficacy and dependability, ensuring the creation of trustworthy computing systems for future applications.

Therefore, this thesis sought to investigate the reliability concerns associated with neuromorphic systems, specifically targeting memristor-based CIM architectures.

At the heart of this exploration lies the introduced fault injection platform designed to evaluate the resilience of Logic-in-Memory (LIM) operations to various in-field faults. This platform serves as an essential instrument for examining the effects of imperfections in memristor-based systems and formulating methods to bolster their reliability. The exhaustive evaluation facilitated by this platform illuminated the vulnerabilities of LIM operations to faults, revealing that a certain threshold of in-field faults could be withstood. Notably, the findings indicated that stuck-at faults more severely impact the reliability of emergent applications than bit-flip faults. The presented insights underscore the necessity of implementing fault-tolerant strategies alongside measures to monitor and mitigate the degradation of the memristive devices throughout their lifespan.

Beyond the reliability issues of memristive devices, their distinctive features unveil a new frontier for hardware security threats. This dissertation introduces NeuroHammer, a significant hardware security attack leveraging these unique attributes to undermine the integrity of computing systems. Specifically, this attack takes advantage on the high memory density to induce deliberate bit-flips via thermal crosstalk in neighboring memory cells. The provided case study underscores the imperative need for countermeasures to shield against such hardware security threats, advocating for a holistic approach to system design that prioritizes security and reliability from the beginning.

Lastly, the NeuroBreakoutBoard (NBB) is presented as a pivotal platform for the detailed examination of memristive devices from both a reliability and hardware security standpoint. Through enabling the investigation of device characteristics across multiple dimensions, the NeuroBreakoutBoard (NBB) underscores the importance of

understanding the underlying mechanisms of memristive behavior to guide the design of more reliable and secure neuromorphic systems.

Based on the presented results, several key requirements can be identified to ensure dependable and secure neuromorphic computing platforms. First, reliability must be addressed across the entire stack, from the device level to the system architecture. While improvements in memristive devices are expected over time, certain non-idealities will remain, requiring solutions at higher abstraction levels. Second, the application itself has a significant impact on reliability. Our results demonstrate that the specific application being executed can accelerate wear on memristive devices, increasing the fault rate. Lastly, hardware security must be a priority during the design phase. Attacks like NeuroHammer, which exploit internal malfunctions, need to be mitigated early in the design process to prevent serious security vulnerabilities.

Conclusively, this dissertation not only addresses the critical challenges of neuromorphic computing but also paves the way for future research directions. By emphasizing the significance of reliability and hardware security in the evolution of memristive-based systems, this work lays the groundwork for the essential progress required to develop trustworthy neuromorphic computing solutions. In addition to the improvement points outlined in the thesis, the following research directions show promise:

- Advanced Error Mitigation Techniques: Error Correcting Codes (ECCs) are a crucial part of modern memories for detecting and mitigating the impact of faults. These well-established techniques cannot simply be applied to Resistive Random-Access Memories (ReRAMs) due to their use of CIM. Therefore, novel ECCs must be developed that account for both memory and CIM applications, while also considering device wear.

- Reliability-aware Mapping Algorithms: Mapping algorithms are a crucial piece of software that assign a given neural network to a specific system architecture. While these algorithms already take into account factors like data flow and the number of compute cores, memristor-based accelerators could benefit from reliability-aware mapping. Such algorithms could optimize device usage and data flow to significantly reduce wear over time.

- Security-Enhanced Architectures: As neuromorphic computing systems continue to evolve, addressing security becomes critical to defend against advanced attacks like NeuroHammer and other exploits. Mitigation strategies must carefully balance the trade-offs between area, power, and performance overhead against the security benefits. Ensuring this balance is essential to maintaining the overall performance and efficiency of future neuromorphic systems while providing robust security protections. Research should focus on developing lightweight, efficient security mechanisms that can be integrated without compromising system resources or computational capabilities.

Appendix A

Simulation Details

This appendix provides detailed information on the JART simulation model discussed in Chapter 5. Additionally, it includes documentation of the alpha matrices essential for conducting circuit-level simulations to verify the NeuroHammer attack.

A.1 Model Parameter

Table A.1: JART VCM v1b model parameters.

$A_{\text{det}} = \pi r^2 = 6.36 \times 10^{-15}\,\text{m}^2$	$r_{\text{det}} = 45\,\text{nm}$
$N_{\text{disc, min, det}} = 0.008 \times 10^{26}\,\text{m}^{-3}$	$N_{\text{disc, max, det}} = 20 \times 10^{26}\,\text{m}^{-3}$
$l_{\text{cell}} = 3\,\text{nm}$	$l_{\text{det}} = 0.4\,\text{nm}$
$l_{\text{plug}} = 2.6\,\text{nm}$	$a = 0.25\,\text{nm}$
$R_{\text{series}} = 1.37\,\text{k}\Omega$ (I = 0 µA)	$R_{\text{series}} = 1.46\,\text{k}\Omega$ (I = 700 µA)
$R_{\text{line}} = 719\,\Omega$ (I = 0 µA)	$R_{\text{line}} = 810\,\Omega$ (I = 700 µA)
$R_{\text{TiOx}} = 650\,\Omega$	$\Delta W_{\text{A}} = 1.35\,\text{eV}$
$\nu_0 = 2 \times 10^{13}\,\text{Hz}$	$\mu_{\text{n}} = 4 \times 10^{-6}\,\text{m}^2/(\text{V s})$
$R_{\text{th0, SET}} = 15.72 \times 10^6\,\text{K/W}$	$R_{\text{th0, RESET}} = 4.24 \times 10^6\,\text{K/W}$
$e\phi_{\text{n0}} = 0.18\,\text{eV}$	$e\phi_{\text{n}} = 0.1\,\text{eV}$
$R_{\text{th,line}} = 90\,471.47\,\text{K/W}$	$R_0 = 719.24\,\Omega$
$\alpha_{\text{line}} = 3.92 \times 10^{-3}/\text{K}$	$m^* = 9.11 \times 10^{-31}\,\text{kg}$
$e = 1.6 \times 10^{-19}\,\text{C}$	$T_0 = 293\,\text{K}$
$A^* = 6.01 \times 10^5\,\text{A}/(\text{m}^2\,\text{K}^2)$	$z_{\text{Vo}} = 2$
$k_{\text{B}} = 1.38 \times 10^{-23}\,\text{J/K}$	$\varepsilon_0 = 8.854 \times 10^{-12}\,\text{A s}/(\text{V m})$
$\varepsilon_{\phi_B} = 5.5\varepsilon_0$	$\varepsilon = 17\varepsilon_0$
$h = 6.626 \times 10^{-34}\,\text{J s}$	

A.2 Alpha Matrices

Table A.2: 9×9 alpha matrix determined for a 10 nm electrode spacing.

0.0233	0.0285	0.0333	0.0369	0.0383	0.0369	0.0333	0.0285	0.0233
0.0290	0.0365	0.0442	0.0505	0.0532	0.0505	0.0442	0.0366	0.0291
0.0357	0.0466	0.0597	0.0729	0.0801	0.0729	0.0598	0.0467	0.0358
0.0417	0.0573	0.0791	0.1084	0.1397	0.1085	0.0791	0.0574	0.0419
0.0444	0.0628	0.0930	0.1519	1.0000	0.1520	0.0932	0.0630	0.0446
0.0416	0.0572	0.0789	0.1082	0.1392	0.1082	0.0790	0.0573	0.0418
0.0355	0.0464	0.0594	0.0725	0.0794	0.0725	0.0594	0.0465	0.0356
0.0288	0.0362	0.0438	0.0499	0.0523	0.0499	0.0438	0.0363	0.0289
0.0231	0.0282	0.0328	0.0362	0.0373	0.0362	0.0329	0.0282	0.0231

Table A.3: 9×9 alpha matrix determined for a 20 nm electrode spacing.

0.0158	0.0201	0.0241	0.0270	0.0283	0.0271	0.0241	0.0201	0.0158
0.0212	0.0278	0.0346	0.0402	0.0427	0.0403	0.0346	0.0279	0.0213
0.0278	0.0380	0.0505	0.0631	0.0700	0.0632	0.0506	0.0381	0.0279
0.0341	0.0494	0.0714	0.1016	0.1324	0.1017	0.0715	0.0496	0.0343
0.0370	0.0557	0.0879	0.1526	1.0000	0.1528	0.0881	0.0560	0.0373
0.0340	0.0493	0.0712	0.1013	0.1317	0.1014	0.0713	0.0494	0.0342
0.0276	0.0378	0.0501	0.0625	0.0690	0.0626	0.0502	0.0379	0.0278
0.0210	0.0275	0.0341	0.0395	0.0416	0.0395	0.0341	0.0275	0.0210
0.0156	0.0197	0.0235	0.0262	0.0270	0.0262	0.0236	0.0198	0.0156

Table A.4: 9×9 alpha matrix determined for a 30 nm electrode spacing.

0.0108	0.0143	0.0176	0.0201	0.0212	0.0202	0.0177	0.0144	0.0109
0.0156	0.0213	0.0273	0.0325	0.0347	0.0325	0.0274	0.0213	0.0156
0.0216	0.0310	0.0427	0.0549	0.0615	0.0549	0.0428	0.0311	0.0217
0.0276	0.0423	0.0641	0.0949	0.1257	0.0950	0.0642	0.0425	0.0278
0.0304	0.0488	0.0819	0.1510	1.0000	0.1512	0.0822	0.0491	0.0308
0.0275	0.0421	0.0638	0.0945	0.1249	0.0946	0.0640	0.0423	0.0277
0.0214	0.0307	0.0423	0.0542	0.0603	0.0543	0.0424	0.0308	0.0215
0.0154	0.0210	0.0268	0.0316	0.0333	0.0316	0.0268	0.0210	0.0154
0.0106	0.0140	0.0171	0.0192	0.0197	0.0192	0.0171	0.0140	0.0107

Table A.5: 9×9 alpha matrix determined for a 40 nm electrode spacing.

0.0074	0.0102	0.0129	0.0149	0.0158	0.0150	0.0129	0.0102	0.0074
0.0113	0.0162	0.0215	0.0260	0.0281	0.0261	0.0215	0.0162	0.0114
0.0166	0.0250	0.0358	0.0474	0.0538	0.0475	0.0359	0.0251	0.0167
0.0220	0.0357	0.0568	0.0877	0.1185	0.0878	0.0570	0.0359	0.0222
0.0246	0.0421	0.0751	0.1473	1.0000	0.1476	0.0755	0.0425	0.0250
0.0219	0.0355	0.0566	0.0872	0.1175	0.0873	0.0567	0.0357	0.0221
0.0164	0.0247	0.0354	0.0466	0.0523	0.0466	0.0355	0.0248	0.0165
0.0111	0.0158	0.0209	0.0251	0.0265	0.0251	0.0209	0.0159	0.0112
0.0072	0.0098	0.0123	0.0139	0.0141	0.0139	0.0123	0.0098	0.0072

Table A.6: 9×9 alpha matrix determined for a 50 nm electrode spacing.

0.0050	0.0071	0.0092	0.0109	0.0117	0.0109	0.0093	0.0071	0.0050
0.0081	0.0121	0.0166	0.0206	0.0224	0.0206	0.0166	0.0122	0.0082
0.0125	0.0198	0.0296	0.0403	0.0464	0.0404	0.0297	0.0199	0.0126
0.0172	0.0295	0.0496	0.0798	0.1103	0.0799	0.0498	0.0298	0.0174
0.0194	0.0356	0.0677	0.1412	1.0000	0.1415	0.0681	0.0361	0.0200
0.0171	0.0294	0.0493	0.0792	0.1090	0.0794	0.0495	0.0296	0.0173
0.0123	0.0195	0.0291	0.0394	0.0447	0.0395	0.0292	0.0196	0.0124
0.0079	0.0118	0.0160	0.0195	0.0206	0.0196	0.0160	0.0118	0.0080
0.0048	0.0068	0.0086	0.0098	0.0097	0.0098	0.0086	0.0068	0.0048

Table A.7: 9×9 alpha matrix determined for a 60 nm electrode spacing.

0.0033	0.0049	0.0066	0.0080	0.0086	0.0080	0.0066	0.0050	0.0033
0.0058	0.0090	0.0128	0.0162	0.0179	0.0163	0.0128	0.0091	0.0058
0.0093	0.0156	0.0243	0.0342	0.0399	0.0343	0.0244	0.0157	0.0095
0.0133	0.0243	0.0430	0.0723	0.1023	0.0725	0.0432	0.0245	0.0136
0.0152	0.0298	0.0606	0.1346	1.0000	0.1350	0.0611	0.0304	0.0158
0.0132	0.0241	0.0426	0.0717	0.1009	0.0718	0.0429	0.0244	0.0135
0.0092	0.0153	0.0238	0.0332	0.0380	0.0333	0.0239	0.0154	0.0093
0.0056	0.0087	0.0122	0.0151	0.0158	0.0151	0.0122	0.0087	0.0056
0.0032	0.0046	0.0060	0.0068	0.0064	0.0068	0.0060	0.0046	0.0032

Table A.8: 9×9 alpha matrix determined for a 70 nm electrode spacing.

0.0022	0.0034	0.0047	0.0058	0.0064	0.0058	0.0047	0.0034	0.0022
0.0041	0.0067	0.0098	0.0128	0.0142	0.0128	0.0098	0.0067	0.0041
0.0069	0.0122	0.0199	0.0289	0.0342	0.0290	0.0200	0.0123	0.0071
0.0102	0.0199	0.0372	0.0655	0.0950	0.0656	0.0374	0.0202	0.0105
0.0118	0.0249	0.0541	0.1282	1.0000	0.1286	0.0547	0.0256	0.0125
0.0102	0.0197	0.0368	0.0647	0.0933	0.0649	0.0371	0.0200	0.0104
0.0068	0.0119	0.0194	0.0279	0.0321	0.0279	0.0195	0.0121	0.0069
0.0039	0.0064	0.0092	0.0116	0.0119	0.0116	0.0092	0.0064	0.0040
0.0021	0.0031	0.0041	0.0046	0.0040	0.0046	0.0041	0.0031	0.0021

Table A.9: 9×9 alpha matrix determined for a 80 nm electrode spacing.

0.0015	0.0024	0.0034	0.0043	0.0048	0.0043	0.0035	0.0024	0.0015
0.0029	0.0050	0.0077	0.0102	0.0115	0.0103	0.0077	0.0051	0.0030
0.0052	0.0097	0.0166	0.0249	0.0300	0.0250	0.0167	0.0098	0.0053
0.0080	0.0165	0.0326	0.0602	0.0897	0.0604	0.0329	0.0168	0.0083
0.0092	0.0210	0.0490	0.1240	1.0000	0.1245	0.0497	0.0218	0.0100
0.0079	0.0163	0.0323	0.0594	0.0877	0.0596	0.0325	0.0166	0.0082
0.0051	0.0095	0.0160	0.0238	0.0275	0.0238	0.0161	0.0096	0.0052
0.0028	0.0047	0.0070	0.0089	0.0089	0.0090	0.0071	0.0048	0.0028
0.0014	0.0021	0.0028	0.0030	0.0021	0.0030	0.0028	0.0022	0.0014

Table A.10: 9×9 alpha matrix determined for a 90 nm electrode spacing.

0.00097	0.00164	0.00240	0.00308	0.00351	0.00309	0.00241	0.00165	0.00098
0.00202	0.00362	0.00574	0.00788	0.00899	0.00790	0.00577	0.00367	0.00205
0.00378	0.00740	0.01321	0.02059	0.02516	0.02065	0.01330	0.00752	0.00388
0.00592	0.01310	0.02745	0.05317	0.08136	0.05334	0.02774	0.01341	0.00621
0.00680	0.01697	0.04253	0.11498	1.0000	0.11558	0.04332	0.01783	0.00764
0.00586	0.01294	0.02709	0.05232	0.07916	0.05251	0.02738	0.01326	0.00614
0.00367	0.00715	0.01264	0.01937	0.02245	0.01943	0.01274	0.00727	0.00377
0.00189	0.00333	0.00510	0.00651	0.00614	0.00652	0.00513	0.00337	0.00193
0.00086	0.00138	0.00181	0.00172	0.00057	0.00172	0.00181	0.00139	0.00087

Table A.11: The 1×32 alpha matrix is utilized for simulating 1-Transistor 1-Resistor (1T1R) structures. For visualization purposes, the line vector is presented across multiple rows.

0.2463	0.2381	0.2234	0.2069	0.1909	0.1759	0.1623	0.1501
0.1394	0.1303	0.1234	0.1205	0.1301	0.1652	0.2638	0.5242
1.0000	0.4763	0.1871	0.0743	0.0301	0.0127	0.0059	0.0031
0.0020	0.0016	0.0013	0.0012	0.0012	0.0011	0.0010	0.0009

Appendix B

Communication Details

This appendix provides detailed information on the defined Protocol Buffers messages used to establish the communication between the host and the STM32 microcontroller.

```
enum _Operation{
 NONE           = 0;
 read           = 1;
 form           = 2;
 set            = 3;
 reset          = 4;
 multiplication = 5;
 readRow        = 6;
 NOR            = 7;
 NOT            = 8;
}
```

Figure B.1: Protocol buffers file specifying the message structure to determine the operational mode of the NeuroBreakoutBoard.

```
message Response{
        _Operation operation = 1;
        uint32 bitLine       = 2; // Bitline index of the target cell
        uint32 sourceLine    = 3; // Sourceline index of the target cell
        uint32 wordLine      = 4; // Wordline index of the target cell
        uint32 resistance    = 5; // Measured resistance of the cell
        int32 vSet           = 6; // Final programming voltage (ISPVA)
        repeated int32 currentList = 7; // List of all measured currents
}
```

Figure B.2: Protocol buffers file defining the response message sent from the STM32 microcontroller to the host Popcounts (PCs),

```
message Request{
 _Operation operation  = 1;
 uint32 bitLine    = 2; // Bitline index of the target cell
 uint32 sourceLine   = 3; // Sourceliune index of the target cell
 uint32 wordLine   = 4; // Wordline index of the target cell
 float vSet     = 5; // Start voltage of the ISPVA [V]
 float vStop    = 6; // Stop voltage of the ISPVA [V]
 float vStep    = 7; // Voltage difference in every step of ISPVA [V]
 float tStep    = 8; // Pulse duration of the ISPVA [us]
 float vGate    = 9; // Gate voltage for the set/form operation [V]
 float vRead    = 10; // Read voltage for the verify step in ISPVA [V]
 float iTarget    = 11; // Target current of the ISPVA [uA]
 float vGateReset  = 12; // Gate voltage for the reset operation [V]
 uint32 vBL_0    = 13; // Binary input vector for multiplication
 uint32 vBL_1   = 14;
 uint32 vBL_2   = 15;
 uint32 vBL_3   = 16;
 uint32 vBL_4   = 17;
 uint32 vBL_5   = 18;
 uint32 vBL_6   = 19;
 uint32 vHigh   = 20; // Voltage representing logic 1
 uint32 vLow    = 21; // Voltage representing logic 0
 float v0    = 22; // Computation voltage of the logic gate
 uint32 logicBL0   = 23; // 1st bitline index of the logic gate
 uint32 logicSL0   = 24; // 2nd bitline index of the logic gate
 uint32 logicSL1   = 25; // Sourceline index of the logic gate
 float vGateRead   = 26; // Gate voltage for the verify step in ISPVA[V]
}
```

Figure B.3: Protocol buffers file specifying the request message sent from the host PCs to the STM32 microcontroller.

Glossary

Acronyms

1T1R	1-Transistor 1-Resistor
ADC	Analog-to-Digital Converter
ASIC	Application-Specific Integrated Circuit
AES	Advanced Encryption Standard
API	Application Programming Interface
BNN	Binary Neural Network
BL	Bit Line
BEOL	Back End Of Line
BE	Bottom Electrode
BLC	Bit Line Controller
CIM	Computing-in-Memory
CMOS	Complementary Metal-Oxide Semiconductor
C2C	Cycle-to-Cycle
CRT	Chinese Remainder Theorem
CDF	Cumulative Distribution Function
DAC	Digital-to-Analog Converter
DRAM	Dynamic Random-Access Memory
D2D	Device-to-Device
DFS	Design For Security
DNN	Deep Neural Network
DMA	Direct Memory Access
DoS	Denial-of-Service
eNVM	Emerging Non-Volatile Memory
ECC	Error Correcting Code
ENTT	Emerging NVM-based Trojan Trigger
EFI	Enhanced Fault Injection Attack
FLIM	Faulty Logic-in-Memory
FEOL	Front End Of Line
FPGA	Field-Programmable Gate Array
FCM	Fast Crossbar Model
FMC	FPGA Mezzanine Card
FTJ	Ferroelectric Tunnel Junction
GND	Ground
GLC	Gate Line Controller
GPDK	Generic Process Design Kit
GPIO	General-Purpose Input/Output
GUI	Graphical User Interface
HRS	High Resistive State
HAL	Hardware Abstraction Layer

IC	Integrated Circuit
IMPLY	Memristor-Based Material Implication
IP	Intellectual Property
IMP	Material Implication
IFV	Incremental Form and Verify
ISPVA	Incremental Step Pulse with Verify Algorithm
JTAG	Joint Test Action Group
LIM	Logic-in-Memory
LRS	Low Resistive State
LLC	Last Level Cache
LSB	Least Significant Bit
MAC	Multiply–Accumulate
MAGIC	Memristor-Aided Logic
MRL	Memristor Ratioed Logic
mCAT	memristor Characterization And Testing
MIM	Metal-Insulator-Metal
NBB	NeuroBreakoutBoard
NMOS	N-Type Metal–Oxide Semiconductor
NN	Neural Network
PUF	Physical Unclonable Function
PCB	Printed Circuit Board
PC	Popcount
PCM	Phase Change Memory
QFP	Quad Flat Package
ReRAM	Resistive Random-Access Memory
RSA	Rivest–Shamir–Adleman
RNG	Random Number Generation
SRAM	Static Random-Access Memory
STT-RAM	Spin-Transfer Torque Random-Access Memory
SMU	Source-Meter Unit
SPI	Serial Peripheral Interface
SWD	Serial Wire Debug
TIA	Transimpedance Amplifier
TE	Top Electrode
UART	Universal Asynchronous Receiver/Transmitter
VCM	Valence Change Material
VADER	Variation-oriented Adversarial Attack
WL	Word Line
WLC	World Line Controller

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[2] S. Adee, "The Hunt For The Kill Switch," *IEEE Spectrum*, vol. 45, no. 5, pp. 34–39, 2008.

[3] aixACCT Systems GmbH, "aixMATRIX – Lightning-Quick Testing Of Neuromorphic Memory Systems," https://www.aixacct.com/testsysteme/paralleltestsysteme/aixmatrix, Accessed: 2023-11-14.

[4] A. Al-Shaarawy, A. Amirsoleimani, and R. Genov, "PRUNIX: Non-Ideality Aware Convolutional Neural Network Pruning for Memristive Accelerators," in *IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2022.

[5] K. A. Ali, M. Rizk, A. Baghdadi, J.-P. Diguet, J. Jomaah, N. Onizawa, and T. Hanyu, "Memristive Computational Memory Using Memristor Overwrite Logic (MOL)," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 11, pp. 2370–2382, 2020.

[6] Analog Devices Inc., "AD7606 - 8-Channel DAS with 16-Bit, Bipolar Input, Simultaneous Sampling ADC," https://www.analog.com/media/en/technical-documentation/data-sheets/ad7606_7606-6_7606-4.pdf, accessed: 2024-01-20.

[7] Analog Devices Inc., "AD826 - Low Cost, High Speed, Low Power Dual Operational Amplifier," https://www.analog.com/media/en/technical-documentation/data-sheets/AD826.pdf, accessed: 2024-01-20.

[8] Analog Devices Inc., "ADG1408 - iCMOS Multiplexers," https://www.analog.com/media/en/technical-documentation/data-sheets/adg1408_1409.pdf, accessed: 2024-01-20.

[9] Analog Devices Inc., "ADG613 - 1 pC Charge Injection, 100 pA Leakage, CMOS, ±5 V, +5 V, +3 V, Quad SPST Switches," https://www.analog.com/media/en/technical-documentation/data-sheets/ADG611_612_613.pdf, accessed: 2024-01-20.

[10] Analog Devices Inc., "LT1963A - 1.5A, Low Noise, Fast Transient Response LDO Regulators," https://www.analog.com/media/en/technical-documentation/data-sheets/1963aff.pdf, accessed: 2024-01-20.

[11] Analog Devices Inc., "LT3015 - 1.5A, Low Noise, Negative Linear Regulator with Precision Current Limit," https://www.analog.com/media/en/technical-documentation/data-sheets/3015fb.pdf, accessed: 2024-01-20.

[12] Analog Devices Inc., "LTC2668 - 16-Channel 16-/12-Bit," https://www.analog.com/media/en/technical-documentation/data-sheets/ltc2668.pdf, accessed: 2024-01-20.

[13] Analog Devices Inc., "LTC6269 - Dual 500MHz Ultra-Low Bias Current FET Input Op Amp," https://www.analog.com/media/en/technical-documentation/data-sheets/62689f.pdf, accessed: 2024-01-20.

[14] Analog Devices Inc., "MAX4564 - Low-Voltage, Dual-Supply, SPDT Analog Switch," https://www.analog.com/media/en/technical-documentation/data-sheets/MAX4564.pdf, accessed: 2024-01-20.

[15] Analog Devices Inc., "MAX6126 - Ultra-High-Precision, Ultra-Low-Noise,Series Voltage Reference," https://www.analog.com/media/en/technical-documentation/data-sheets/MAX6126.pdf, accessed: 2024-01-20.

[16] L. Anghel, A. Bernasconi, V. Ciriani, L. Frontini, G. Trucco, and I. Vatajelu, "Stuck-At Fault Mitigation of Emerging Technologies Based Switching Lattices," *Journal of Electronic Testing*, vol. 36, no. 3, pp. 313–326, 2020.

[17] ARC Instruments Ltd., "Arc Instruments - High Performance Array Control Instruments," https://www.whitehouse.gov/briefing-room/statements-releases/2022/08/09/fact-sheet-chips-and-science-act-will-lower-costs-create-jobs-strengthen-supply-chains-and-counter-china/, Accessed: 2023-11-13.

[18] J. Arlat, Y. Crouzet, J. Karlsson, P. Folkesson, E. Fuchs, and G. Leber, "Comparison of Physical and Software-Implemented Fault Injection Techniques," *IEEE Transactions on Computers*, vol. 52, no. 9, pp. 1115–1133, 2003.

[19] C. Bengel, A. Siemon, F. Cuppers, S. Hoffmann-Eifert, A. Hardtdegen, M. von Witzleben, L. Hellmich, R. Waser, and S. Menzel, "Variability-Aware Modeling of Filamentary Oxide-Based Bipolar Resistive Switching Cells Using SPICE Level Compact Models," *Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 12, pp. 4618–4630, 2020.

[20] R. Berdan, A. Serb, A. Khiat, A. Regoutz, C. Papavassiliou, and T. Prodromakis, "A u-Controller-Based System for Interfacing Selectorless RRAM Crossbar Arrays," *IEEE Transactions on Electron Devices*, vol. 62, no. 7, pp. 2190–2196, 2015.

[21] J. Bethge, C. Bartz, H. Yang, Y. Chen, and C. Meinel, "MeliusNet: An Improved Network Architecture for Binary Neural Networks," in *Conference on Applications of Computer Vision*, 2021, pp. 1439–1448.

[22] J. Bethge, H. Yang, M. Bornstein, and C. Meinel, "Back to Simplicity: How to Train Accurate BNNs From Scratch?" *arXiv preprint arXiv:1906.08637*, 2019.

[23] D. Bhattacharjee, A. Dutt, and A. Chattopadhyay, "MAMI: Majority and Multi-Input Logic on Memristive Crossbar Array," in *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*. IEEE, 2018.

[24] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware Trojan Attacks: Threat Analysis and Countermeasures," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1229–1247, 2014.

[25] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The Gem5 Simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[26] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the Importance of Eliminating Errors in Cryptographic Computations," *Journal of Cryptology*, vol. 14, no. 2, pp. 101–119, 2000.

[27] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "Memristive Switches Enable Stateful Logic Operations Via Material Implication," *Nature*, vol. 464, no. 7290, pp. 873–876, 2010.

[28] G. W. Burr, R. M. Shelby, A. Sebastian, S. Kim, S. Kim, S. Sidler, K. Virwani, M. Ishii, P. Narayanan, A. Fumarola, L. L. Sanches, I. Boybat, M. L. Gallo, K. Moon, J. Woo, H. Hwang, and Y. Leblebici, "Neuromorphic Computing Using Non-volatile Memory," *Advances in Physics: X*, vol. 2, no. 1, pp. 89–124, 2016.

[29] Cadence Design Systems Inc., "Virtuoso System Design Platform," https://www.cadence.com/en_US/home/tools/custom-ic-analog-rf-design/virtuoso-studio.html, Accessed: 2024-01-14.

[30] L. Caviglione, "Trends and Challenges in Network Covert Channels Countermeasures," *Applied Sciences*, vol. 11, no. 4, p. 1641, 2021.

[31] I. Chakraborty, M. F. Ali, D. E. Kim, A. Ankit, and K. Roy, "GENIEx: A Generalized Approach to Emulating Non-Ideality in Memristive XBars using Neural Networks," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020.

[32] M.-F. Chang, W. Fuchs, and J. Patel, "Diagnosis and Repair of Memory with Coupling Faults," *IEEE Transactions on Computers*, vol. 38, no. 4, pp. 493–500, 1989.

[33] A. Chaudhuri and K. Chakrabarty, "Analysis of Process Variations, Defects, and Design-Induced Coupling in Memristors," in *IEEE International Test Conference (ITC)*. IEEE, 2018.

[34] M. Chaudhuri, "Zero Inclusion Victim: Isolating Core Caches from Inclusive Last-level Cache Evictions," in *Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021.

[35] C.-Y. Chen, H.-C. Shih, C.-W. Wu, C.-H. Lin, P.-F. Chiu, S.-S. Sheu, and F. T. Chen, "RRAM Defect Modeling and Failure Analysis Based on March Test and a Novel Squeeze-Search Scheme," *IEEE Transactions on Computers*, vol. 64, no. 1, pp. 180–190, 2015.

[36] P.-Y. Chen, X. Peng, and S. Yu, "Neurosim+: An Integrated Device-To-Algorithm Framework For Benchmarking Synaptic Devices And Array Architectures," in *IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2017.

[37] P.-Y. Chen, X. Peng, and S. Yu, "NeuroSim: A Circuit-Level Macro Model for Benchmarking Neuro-Inspired Architectures in Online Learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 12, pp. 3067–3080, 2018.

[38] Y.-X. Chen and J.-F. Li, "Fault Modeling and Testing of 1T1R Memristor Memories," in *IEEE VLSI Test Symposium (VTS)*. IEEE, 2015.

[39] Z. Chen, H. Wu, B. Gao, D. Wu, N. Deng, H. Qian, Z. Lu, B. Haukness, M. Kellam, and G. Bronner, "Performance Improvements by SL-Current Limiter and Novel Programming Methods on 16MB RRAM Chip," in *International Memory Workshop (IMW)*. IEEE, 2017.

[40] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[41] C. Chou, Z. Lin, P. Tseng, C. Li, C. Chang, W. Chen, Y. Chih, and T. J. Chang, "An N40 256k×44 Embedded RRAM Macro with Sl-precharge SA and Low-Voltage Current Limiter to Improve Read and Write Performance," in *IEEE International Solid - State Circuits Conference - (ISSCC)*. IEEE International Solid - State Circuits Conference - (ISSCC), 2018, pp. 478–480.

[42] C.-C. Chou, Z.-J. Lin, C.-A. Lai, C.-I. Su, P.-L. Tseng, W.-C. Chen, W.-C. Tsai, W.-T. Chu, T.-C. Ong, H. Chuang, Y.-D. Chih, and T.-Y. J. Chang, "A 22nm 96kx144 RRam Macro with a Self-Tracking Reference and a Low Ripple Charge Pump to Achieve a Configurable Read Window and a Wide Operating Voltage Range," in *Symposium on VLSI Circuits*. IEEE, 2020.

[43] O. Choudary and M. G. Kuhn, "Template Attacks on Different Devices," in *Constructive Side-Channel Analysis and Secure Design*. Springer International Publishing, 2014, pp. 179–198.

[44] L. Chua, "Memristor-The Missing Circuit Element," *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971.

[45] COMSOL AB, "COMSOL Multiphysics®," https://www.comsol.com/comsol-multiphysics, accessed: 2023-09-14.

[46] J. R. Contreras, J. Schubert, H. Kohlstedt, and R. Waser, "Memory Device Based on a Ferroelectric Tunnel Junction," in *Device Research Conference, Santa Barbara, CA, USA, 24/06/2002-26/06/2002*, Inst fur Festkorperforschung, Forschungszentrum Julich GmbH, Germany. Piscataway, NJ, USA: IEEE, 2002, pp. 97–8.

[47] F. Cüppers, S. Menzel, C. Bengel, A. Hardtdegen, M. von Witzleben, U. Böttger, R. Waser, and S. Hoffmann-Eifert, "Exploiting the Switching Dynamics of HfO2-Based ReRAM Devices for Reliable Analog Memristive Behavior," *APL Materials*, vol. 7, no. 9, 2019.

[48] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A Large-Scale Hierarchical Image Database," in *Conference on Computer Vision and Pattern Recognition*. IEEE, 2009.

[49] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[50] L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, S. Borri, and M. Hage-Hassan, "Dynamic Read Destructive Fault in Embedded-SRAMs: Analysis and March Test Solution," in *IEEE European Test Symposium (ETS)*. IEEE, 2004.

[51] V. Dumoulin and F. Visin, "A Guide to Convolution Arithmetic for Deep Learning," *arXiv preprint arXiv:1603.07285*, 2016.

[52] A. S. Emara, A. H. Madian, H. H. Amer, S. H. Amer, and M. B. Abdelhalim, "Testing Of Memristor Ratioed Logic (MRL) XOR Gate," in *International Conference on Microelectronics (ICM)*. IEEE, 2016.

[53] S. S. Ensan, K. Nagarajan, M. N. I. Khan, and S. Ghosh, "SCARE: Side Channel Attack on In-Memory Computing for Reverse Engineering," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 12, pp. 2040–2051, 2021.

[54] T. K. Esatu, A. Prakash, Z. Li, D. Lau, S. H. Jo, and T.-J. K. Liu, "Highly Reliable and Secure PUF Using Resistive Memory Integrated Into a 28 nm CMOS Process," *Transactions on Electron Devices*, vol. 70, no. 5, pp. 2291–2296, 2023.

[55] M. Escudero, I. Vourkas, A. Rubio, and F. Moll, "Variability-Tolerant Memristor-based Ratioed Logic in Crossbar Array," in *International Symposium on Nanoscale Architectures*, ser. NANOARCH '18. ACM, 2018.

[56] M. Escudero, I. Vourkas, A. Rubio, and F. Moll, "Memristive Logic in Crossbar Memory Arrays: Variability-Aware Design for Higher Reliability," *IEEE Transactions on Nanotechnology*, vol. 18, pp. 635–646, 2019.

[57] M. Eslami, B. Ghavami, M. Raji, and A. Mahani, "A Survey on Fault Injection Methods of Digital Integrated Circuits," *Integration*, vol. 71, pp. 154–163, 2020.

[58] B. Feinberg, S. Wang, and E. Ipek, "Making Memristive Neural Network Accelerators Reliable," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018.

[59] N. Fern, S. Kulkarni, and K.-T. T. Cheng, "Hardware Trojans Hidden In RTL Don't Cares - Automated Insertion And Prevention Methodologies," in *IEEE International Test Conference (ITC)*. IEEE, 2015.

[60] C. Fernandez and I. Vourkas, "Reliability-Aware Ratioed Logic Operations for Energy-Efficient Computational ReRAM," in *International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2022.

[61] M. Fieback, "Testing RRAM and Computation-in-Memory Devices," Ph.D. dissertation, 2022.

[62] M. Fieback, G. C. Medeiros, L. Wu, H. Aziza, R. Bishnoi, M. Taouil, and S. Hamdioui, "Defects, Fault Modeling, and Test Development Framework for RRAMs," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 18, no. 3, pp. 1–26, 2022.

[63] M. Fieback, M. Taouil, and S. Hamdioui, "Testing Resistive Memories: Where are We and What is Missing?" in *IEEE International Test Conference (ITC)*. IEEE, 2018.

[64] P. Foster, J. Huang, A. Serb, T. Prodromakis, and C. Papavassiliou, "An FPGA Based System for Interfacing with Crossbar Arrays," in *International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020.

[65] P. Foster, J. Huang, A. Serb, S. Stathopoulos, C. Papavassiliou, and T. Prodromakis, "An FPGA-Based System For Generalised Electron Devices Testing," *Scientific Reports*, vol. 12, no. 1, 2022.

[66] C. Funck and S. Menzel, "Comprehensive Model of Electron Conduction in Oxide-based Memristive Devices," *ACS Applied Electronic Materials*, vol. 3, pp. 3674–3692, 2021.

[67] M. L. Gallo and A. Sebastian, "An Overview of Phase-change Memory Device Physics," *Journal of Physics D: Applied Physics*, vol. 53, no. 21, p. 213002, 2020.

[68] A. Gangolli, Q. H. Mahmoud, and A. Azim, "A Systematic Review of Fault Injection Attacks on IoT Systems," *Electronics*, vol. 11, no. 13, p. 2023, 2022.

[69] L. Geiger and P. Team, "Larq: An Open-Source Library for Training Binarized Neural Networks," *Journal of Open Source Software*, vol. 5, no. 45, p. 1746, 2020. [Online]. Available: https://doi.org/10.21105/joss.01746

[70] R. Gennaro, H. Krawczyk, and T. Rabin, *RSA-Based Undeniable Signatures*. Springer Berlin Heidelberg, 1997, pp. 132–149.

[71] Google Inc., "FlatBuffers," https://flatbuffers.dev, Accessed: 2024-01-24.

[72] Google Inc., "Protocol Buffers Documentation," https://protobuf.dev/, Accessed: 2024-01-24.

[73] L. Grenouillet, N. Castellani, A. Persico, V. Meli, S. Martin, O. Billoint, R. Segaud, S. Bernasconi, C. Pellissier, C. Jahan, C. Charpin-Nicolle, P. Dezest, C. Carabasse, P. Besombes, S. Ricavy, N.-P. Tran, A. Magalhaes-Lucas, A. Roman, C. Boixaderas, T. Magis, M. Bedjaoui, M. Tessaire, A. Seignard, F. Mazen, S. Landis, E. Vianello, G. Molas, F. Gaillard, J. Arcamone, and E. Nowak, "16kbit 1T1R Oxram Arrays Embedded in 28nm Fdsoi Technology Demonstrating Low Ber, High Endurance, and Compatibility with Core Logic Transistors," in *International Memory Workshop (IMW)*. IEEE, 2021.

[74] A. Grossi, E. Nowak, C. Zambelli, C. Pellissier, S. Bernasconi, G. Cibrario, K. E. Hajjam, R. Crochemore, J. Nodin, P. Olivo, and L. Perniola, "Fundamental Variability Limits Of Filament-Based RRAM," in *IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2016.

[75] A. Grossi, C. Zambelli, P. Olivo, E. Miranda, V. Stikanov, C. Walczyk, and C. Wenger, "Electrical Characterization and Modeling of Pulse-Based Forming Techniques in RRAM Arrays," *Solid-State Electronics*, vol. 115, pp. 17–25, 2016.

[76] L. Guckert and E. E. Swartzlander, "MAD Gates - Memristor Logic Design Using Driver Circuitry," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 2, pp. 171–175, 2017.

[77] S. Hamdioui, Z. Al-Ars, A. J. van de Goor, and M. Rodgers, "Dynamic Faults in Random-Access-Memories: Concept, Fault Models and Tests," *Journal of Electronic Testing*, vol. 19, no. 2, pp. 195–205, 2003.

[78] S. Hamdioui, M. Taouil, and N. Z. Haron, "Testing Open Defects in Memristor-Based Memories," *IEEE Transactions on Computers*, vol. 64, no. 1, pp. 247–259, 2015.

[79] R. W. Hamming, "Error Detecting and Error Correcting Codes," *Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[80] N. Z. Haron and S. Hamdioui, "DfT Schemes for Resistive Open Defects in RRAMs," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2012.

[81] N. Z. Haron and S. Hamdioui, "On Defect Oriented Testing for Hybrid CMOS/Memristor Memory," in *Asian Test Symposium*. IEEE, 2011.

[82] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Conference On Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[83] Z. He, J. Lin, R. Ewetz, J.-S. Yuan, and D. Fan, "Noise Injection Adaption: End-to-End ReRAM Crossbar Non-ideal Effect Adaption for Neural Network Mapping," in *Design Automation Conference*. ACM, 2019.

[84] B. Hoffer, N. Wainstein, C. M. Neumann, E. Pop, E. Yalon, and S. Kvatinsky, "Stateful Logic Using Phase Change Memory," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 8, no. 2, pp. 77–83, 2022.

[85] W. Hu, C.-H. Chang, A. Sengupta, S. Bhunia, R. Kastner, and H. Li, "An Overview of Hardware Security and Trust: Threats, Countermeasures, and Design Tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 6, pp. 1010–1038, 2021.

[86] D. Ielmini and H. P. Wong, "In-memory Computing with Resistive Switching Devices," *Nature Electronics*, vol. 1, no. 6, pp. 333–343, 2018.

[87] D. Ielmini and R. Waser, *Resistive Switching: From Fundamentals of Nanoionic Redox Processes to Memristive Device Applications*. Wiley-VCH Verlag GmbH & Co. KGaA, 2016.

[88] Intel Corporation, "Intel Optane Memory Series," https://www.intel.com/content/www/us/en/products/sku/97544/intel-optane-memory-series-16gb-m-2-80mm-pcie-3-0-20nm-3d-xpoint/specifications.html, Accessed: 2023-05-22.

[89] P. Jain, U. Arslan, M. Sekhar, B. C. Lin, L. Wei, T. Sahu, J. Alzate-vinasco, A. Vangapaty, M. Meterelliyoz, N. Strutt, A. B. Chen, P. Hentges, P. A. Quintero, C. Connor, O. Golonzka, K. Fischer, and F. Hamzaoglu, "13.2 A 3.6Mb 10.1Mb/mm2 Embedded Non-Volatile ReRAM Macro in 22nm FinFET Technology with Adaptive Forming/Set/Reset Schemes Yielding Down to 0.5V with Sensing Time of 5ns at 0.7V," in *International Solid- State Circuits Conference (ISSCC)*. IEEE, 2019.

[90] S. Jain, A. Sengupta, K. Roy, and A. Raghunathan, "RxNN: A Framework for Evaluating Deep Neural Networks on Resistive Crossbars," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 2, pp. 326–338, 2021.

[91] Y. Jin, "Introduction to Hardware Security," *Electronics*, vol. 4, no. 4, pp. 763–784, 2015.

[92] S. H. Jo, T. Kumar, C. Zitlaw, and H. Nazarian, "Self-Limited Rram with On/off Resistance Ratio Amplification," in *Symposium on VLSI Technology (VLSI Technology)*. IEEE, 2015.

[93] G. Joy Persial, M. Prabhu, and R. Shanmugalakshmi, "Side Channel Attack-Survey," *International Journal of Scientific Research and Reviews*, vol. 1, no. 4, pp. 54–57, 2011.

[94] S. Kannan, N. Karimi, R. Karri, and O. Sinanoglu, "Detection, Diagnosis, And Repair Of Faults In Memristor-Based Memories," in *IEEE VLSI Test Symposium (VTS)*. IEEE, 2014.

[95] S. Kannan, N. Karimi, R. Karri, and O. Sinanoglu, "Modeling, Detection, and Diagnosis of Faults in Multilevel Memristor Memories," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 5, pp. 822–834, 2015.

[96] S. Kannan, R. Karri, and O. Sinanoglu, "Sneak Path Testing And Fault Modeling For Multilevel Memristor-Based Memories," in *IEEE International Conference on Computer Design (ICCD)*. IEEE, 2013.

[97] S. Kannan, J. Rajendran, R. Karri, and O. Sinanoglu, "Sneak-path Testing of Memristor-based Memories," in *International Conference on VLSI Design and International Conference on Embedded Systems*. IEEE, 2013.

[98] Z. E. Kaya, S. B. Tekin, and S. Kalem, "Design Of An FPGA-Based RRAM Parameter Measurement Platform," in *International Conference on Industrial Technology (ICIT)*. IEEE, 2018.

[99] M. N. I. Khan, S. Bhasin, A. Yuan, A. Chattopadhyay, and S. Ghosh, "Side-Channel Attack on STTRAM Based Cache for Cryptographic Application," in *IEEE International Conference on Computer Design (ICCD)*. IEEE, 2017.

[100] M. N. I. Khan and S. Ghosh, "Analysis of Row Hammer Attack on STTRAM," in *IEEE International Conference on Computer Design (ICCD)*. IEEE, 2018.

[101] M. N. I. Khan and S. Ghosh, "Fault Injection Attacks On Emerging Non-Volatile Memory And Countermeasures," in *International Workshop on Hardware and Architectural Support for Security and Privacy*. ACM, 2018.

[102] M. N. I. Khan, K. Nagarajan, and S. Ghosh, "Hardware Trojans in Emerging Non-Volatile Memories," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019.

[103] H. Kim, M. R. Mahmoodi, H. Nili, and D. B. Strukov, "4k-Memristor Analog-Grade Passive Crossbar Circuit," *Nature Communications*, vol. 12, no. 1, 2021.

[104] K.-H. Kim, S. Gaba, D. Wheeler, J. M. Cruz-Albrecht, T. Hussain, N. Srinivasa, and W. Lu, "A Functional Hybrid Memristor Crossbar-Array/cmos System for Data Storage and Neuromorphic Applications," *Nano Letters*, vol. 12, no. 1, pp. 389–395, 2011.

[105] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping Bits in Memory Without Accessing Them," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 361–372, 2014.

[106] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre Attacks: Exploiting Speculative Execution," in *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2019.

[107] M. Kooli and G. D. Natale, "A Survey on Simulation-Based Fault Injection Tools for Complex Systems," in *IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*. IEEE, 2014.

[108] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification With Deep Convolutional Neural Networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[109] K. J. Kuhn, M. D. Giles, D. Becher, P. Kolar, A. Kornfeld, R. Kotlyar, S. T. Ma, A. Maheshwari, and S. Mudanai, "Process Technology Variation," *IEEE Transactions on Electron Devices*, vol. 58, no. 8, pp. 2197–2208, 2011.

[110] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MAGIC - Memristor-Aided Logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.

[111] S. Kvatinsky, A. Kolodny, U. C. Weiser, and E. G. Friedman, "Memristor-based IMPLY Logic Design Procedure," in *IEEE International Conference on Computer Design (ICCD)*. IEEE, 2011.

[112] S. Kvatinsky, N. Wald, G. Satat, A. Kolodny, U. C. Weiser, and E. G. Friedman, "MRL - Memristor Ratioed Logic," in *International Workshop on Cellular Nanoscale Networks and their Applications*. IEEE, 2012.

[113] C. Lammie and M. R. Azghadi, "MemTorch: A Simulation Framework for Deep Memristive Cross-Bar Architectures," in *IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020.

[114] C. Lammie, W. Xiang, and M. R. Azghadi, "Modeling And Simulating In-Memory Memristive Deep Learning Systems: An Overview Of Current Efforts," *Array*, vol. 13, p. 100116, 2022.

[115] B. Q. Le, A. Levy, T. F. Wu, R. M. Radway, E. R. Hsieh, X. Zheng, M. Nelson, P. Raina, H.-S. P. Wong, S. Wong, and S. Mitra, "Radar: A Fast and Energy-efficient Programming Technique for Multiple Bits-per-cell Rram Arrays," *IEEE Transactions on Electron Devices*, vol. 68, no. 9, pp. 4397 – 4403, 2021.

[116] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Handwritten Digit Recognition with a Back-Propagation Network," in *International Conference on Neural Information Processing Systems*, ser. NIPS'89. Cambridge, MA, USA: MIT Press, 1989, p. 396–404.

[117] M.-J. Lee, C. B. Lee, D. Lee, S. R. Lee, M. Chang, J. H. Hur, Y.-B. Kim, C.-J. Kim, D. H. Seo, S. Seo, U.-I. Chung, I.-K. Yoo, and K. Kim, "A Fast, High-Endurance and Scalable Non-Volatile Memory Device Made from Asymmetric Ta2O5X/TaO2X Bilayer Structures," *Nature Materials*, vol. 10, no. 8, pp. 625–630, 2011.

[118] E. Lehtonen and M. Laiho, "Stateful Implication Logic With Memristors," in *IEEE/ACM International Symposium on Nanoscale Architectures*. IEEE, 2009.

[119] R. K. Lenka, S. Padhi, and K. M. Nayak, "Fault Injection Techniques - A Brief Review," in *International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*. IEEE, 2018.

[120] Y. Levy, J. Bruck, Y. Cassuto, E. G. Friedman, A. Kolodny, E. Yaakobi, and S. Kvatinsky, "Logic Operations In Memory Using A Memristive Akers Array," *Microelectronics Journal*, vol. 45, no. 11, pp. 1429–1437, 2014.

[121] B. Li, H. Lv, Y. Wang, and Y. Chen, "Security Threat to the Robustness of RRAM-based Neuromorphic Computing System," in *International Symposium on Smart Electronic Systems (iSES)*. IEEE, 2022.

[122] C. Li, Y. Li, H. Jiang, W. Song, P. Lin, Z. Wang, J. J. Yang, Q. Xia, M. Hu, E. Montgomery, J. Zhang, N. Davila, C. E. Graves, Z. Li, J. P. Strachan, R. S. Williams, N. Ge, M. Barnell, and Q. Wu, "Large Memristor Crossbars for Analog Computing," in *International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018.

[123] H. Li, W. Chen, A. Levy, C. Wang, H. Wang, P. Chen, W. Wan, W. Khwa, H. Chuang, Y. Chih, M. Chang, H. P. Wong, and P. Raina, "Sapiens: A 64-Kb RRAM Non-Volatile Associative Memory For One-Shot Learning And Inference At The Edge," *IEEE Transactions on Electron Devices*, pp. 1–7, 2021.

[124] H. H. Li, Y. Chen, C. Liu, J. P. Strachan, and N. Davila, "Looking Ahead for Resistive Memory Technology: A Broad Perspective on Rera Technology for Future Storage and Computing," *IEEE Consumer Electronics Magazine*, vol. 6, no. 1, pp. 94–103, 2017.

[125] H. Li, S. Wang, X. Zhang, W. Wang, R. Yang, Z. Sun, W. Feng, P. Lin, Z. Wang, L. Sun, and Y. Yao, "Memristive Crossbar Arrays for Storage and Computing Applications," *Advanced Intelligent Systems*, vol. 18, pp. 309–323, 2021.

[126] J.-F. Li, K.-L. Cheng, C.-T. Huang, and C.-W. Wu, "March-Based RAM Diagnosis Algorithms for Stuck-At and Coupling Faults," in *Proceedings International Test Conference*. IEEE, 2001.

[127] C. Liaw, "Integrated Semiconductor Memory with an Arrangement of Non-volatile Memory Cells, and Method," *United States Patent 7277312*, 2007.

[128] M.-Y. Lin, H.-Y. Cheng, W.-T. Lin, T.-H. Yang, I.-C. Tseng, C.-L. Yang, H.-W. Hu, H.-S. Chang, H.-P. Li, and M.-F. Chang, "DL-RSIM: A Simulation Framework to Enable Reliable ReRAM-based Accelerators for Deep Learning," in *ACM International Conference on Computer-Aided Design*, 2018.

[129] E. Linn, R. Rosezin, C. Kügeler, and R. Waser, "Complementary Resistive Switches for Passive Nanocrossbar Memories," *Nature Materials*, vol. 9, no. 5, pp. 403–406, 2010.

[130] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading Kernel Memory from User Space," in *USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, 2018, pp. 973–990. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/lipp

[131] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, M. Hamburg, and R. Strackx, "Meltdown: Reading Kernel Memory from User Space," *Communications of the ACM*, vol. 63, no. 6, pp. 46–56, 2020.

[132] C. Liu, M. Hu, J. P. Strachan, and H. H. Li, "Rescuing Memristor-based Neuromorphic Design with High Defects," in *Proceedings of the Design Automation Conference*. ACM, 2017.

[133] M. Liu and K. Chakrabarty, "Online Fault Detection in ReRAM-Based Computing Systems for Inferencing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 4, pp. 392–405, 2022.

[134] P. Liu, Z. You, J. Wu, B. Liu, Y. Han, and K. Chakrabarty, "Fault Modeling and Efficient Testing of Memristor-Based Memory," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 11, pp. 4444–4455, 2021.

[135] Y. Liu, Y. Jin, A. Nosratinia, and Y. Makris, "Silicon Demonstration of Hardware Trojan Design and Detection in Wireless Cryptographic ICs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 4, pp. 1506–1519, 2017.

[136] Z. Liu, W. Luo, B. Wu, X. Yang, W. Liu, and K.-T. Cheng, "Bi-Real Net: Binarizing Deep Network Towards Real-Network Performance," *International Journal of Computer Vision*, vol. 128, no. 1, pp. 202–219, 2020.

[137] C. Lopez-Ongil, L. Entrena, M. Garcia-Valderas, M. Portela, M. Aguirre, J. Tombs, V. Baena, and F. Munoz, "A Unified Environment for Fault Injection at Any Design Level Based on Emulation," *IEEE Transactions on Nuclear Science*, vol. 54, no. 4, pp. 946–950, 2007.

[138] Y. Lyu and P. Mishra, "A Survey of Side-Channel Attacks on Caches and Countermeasures," *Journal of Hardware and Systems Security*, vol. 2, no. 1, pp. 33–50, 2017.

[139] M. Mao, Y. Cao, S. Yu, and C. Chakrabarti, "Optimizing Latency, Energy, And Reliability Of 1T1R ReRAM Through Appropriate Voltage Settings," in *IEEE International Conference on Computer Design (ICCD)*. IEEE, 2015.

[140] M. Mao, Y. Cao, S. Yu, and C. Chakrabarti, "Optimizing Latency, Energy, And Reliability of 1T1R ReRAM Through Cross-Layer Techniques," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 3, pp. 352–363, 2016.

[141] A. Marchewka, B. Roesgen, K. Skaja, H. Du, C. L. Jia, J. Mayer, V. Rana, R. Waser, and S. Menzel, "Nanoionic Resistive Switching Memories: On the Physical Nature of the Dynamic Reset Process," *ACS Applied Electronic Materials*, vol. 2, no. 1, pp. 1 500 233/1–13, 2016.

[142] B. Martinez, J. Yang, A. Bulat, and G. Tzimiropoulos, "Training Binary Neural Networks with Real-To-Binary Convolutions," *arXiv preprint arXiv:2003.11535*, 2020.

[143] S. Menzel, M. von Witzleben, V. Havel, and U. Boettger, "The Ultimate Switching Speed Limit of Redox-based Restive Switching Devices," *Faraday Discussions*, vol. 213, pp. 197–213, 2019.

[144] S. Menzel, M. Waters, A. Marchewka, U. Böttger, R. Dittmann, and R. Waser, "Origin of the Ultra-nonlinear Switching Kinetics in Oxide-based Resistive Switches," *Advanced Functional Materials*, vol. 21, no. 23, pp. 4487–4492, 2011.

[145] S. Menzel, "Juelich Aachen Resistive Switching Tools (JART)," http://www.emrl.de/Jart.html, Accessed: 2024-01-14.

[146] V. Milo, C. Zambelli, P. Olivo, E. Pérez, M. K. Mahadevaiah, O. G. Ossorio, C. Wenger, and D. Ielmini, "Multilevel HfO2-Based Rram Devices for Low-Power Neuromorphic Networks," *APL Materials*, vol. 7, no. 8, 2019.

[147] S. Motaman and S. Ghosh, "Dynamic Computing in Memory (DCIM) in Resistive Crossbar Arrays," in *IEEE International Conference on Computer Design (ICCD)*. IEEE, 2018.

[148] S. N. Mozaffari, S. Tragoudas, and T. Haniotakis, "Fast March Tests for Defects in Resistive Memory," in *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. IEEE, 2015.

[149] K. Nagarajan, M. N. I. Khan, and S. Ghosh, "ENTT: A Family of Emerging NVM-based Trojan Triggers," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2019.

[150] C. Nguyen, C. Cagli, G. Molas, B. Sklenard, C. Nail, K. E. Hajjam, J.-F. Nodin, C. Charpin, S. Bernasconi, and G. Reimbold, "Study of Forming Impact on 4Kbit RRAM Array Performances and Reliability," in *IEEE International Memory Workshop (IMW)*. IEEE, 2017.

[151] W. Otsuka, K. Miyata, M. Kitagawa, K. Tsutsui, T. Tsushima, H. Yoshihara, T. Namise, Y. Terao, and K. Ogata, "A 4mb Conductive-Bridge Resistive Memory with 2.3gb/s Read-Throughput and 216mb/s Program-Throughput," in *International Solid-State Circuits Conference*. IEEE, 2011.

[152] S. Parkin, X. Jiang, C. Kaiser, A. Panchula, K. Roche, and M. Samant, "Magnetically Engineered Spintronic Sensors and Memory," *Proceedings of the IEEE*, vol. 91, no. 5, pp. 661–680, 2003.

[153] S. Pechmann, T. Mai, M. Völkel, M. K. Mahadevaiah, E. Perez, E. Perez-Bosch Quesada, M. Reichenbach, C. Wenger, and A. Hagelauer, "A Versatile, Voltage-Pulse Based Read and Programming Circuit for Multi-Level RRAM Cells," *Electronics*, vol. 10, no. 5, p. 530, 2021.

[154] X. Peng, S. Huang, Y. Luo, X. Sun, and S. Yu, "DNN+NeuroSim: An End-to-End Benchmarking Framework for Compute-in-Memory Accelerators with Versatile Device Technologies," in *IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2019.

[155] E. Perez, A. Grossi, C. Zambelli, P. Olivo, R. Roelofs, and C. Wenger, "Reduction of the Cell-to-Cell Variability in Hf1-xAlxOyBased RRAM Arrays by Using Program Algorithms," *IEEE Electron Device Letters*, vol. 38, no. 2, pp. 175–178, 2017.

[156] S. Qadir and S. M. K. Quadri, "Information Availability: An Insight into the Most Important Attribute of Information Security," *Journal of Information Security*, vol. 07, no. 03, pp. 185–194, 2016.

[157] C. Quan, M. E. Fouda, S. Lee, and J. Lee, "Multi-Fidelity Nonideality Simulation and Evaluation Framework for Resistive Neuromorphic Computing," in *Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2022.

[158] S. Rai, S. Garg, C. Pilato, V. Herdt, E. Moussavi, D. Sisejkovic, R. Karri, R. Drechsler, F. Merchant, and A. Kumar, "Vertical IP Protection of the Next-Generation

Devices: Quo Vadis?" in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, Feb. 2021.

[159] M. J. Rasch, D. Moreda, T. Gokmen, M. L. Gallo, F. Carta, C. Goldberg, K. E. Maghraoui, A. Sebastian, and V. Narayanan, "A Flexible and Fast PyTorch Toolkit for Simulating Training and Inference on Analog Crossbar Arrays," in *IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2021.

[160] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: Imagenet Classification Using Binary Convolutional Neural Networks," in *European Conference On Computer Vision*. Springer, 2016, pp. 525–542.

[161] V. Ravi and S. R. S. Prabaharan, "Memristor Based Memories: Defects, Testing, And Testability Techniques," *Far East Journal of Electronics and Communications*, vol. 17, no. 1, pp. 105–125, 2017.

[162] J. Reuben, R. Ben-Hur, N. Wald, N. Talati, A. H. Ali, P.-E. Gaillardon, and S. Kvatinsky, "Memristive Logic: A Framework For Evaluation And Comparison," in *International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. IEEE, 2017.

[163] R. L. Rivest, A. Shamir, and L. M. Adleman, "Cryptographic Communications System and Method," 1983, uS Patent 4,405,829.

[164] S. Roy, S. Sridharan, S. Jain, and A. Raghunathan, "TxSim: Modeling Training of Deep Neural Networks on Resistive Crossbar Systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 4, pp. 730–738, 2021.

[165] S. Samonas and D. Coss, "The CIA Strikes Back: Redefining Confidentiality, Integrity and Availability in Security." *Journal of Information System Security*, vol. 10, no. 3, 2014.

[166] S. Schechter, G. H. Loh, K. Strauss, and D. Burger, "Use ECP, Not ECC, For Hard Failures In Resistive Memories," in *ACM International Symposium On Computer Architecture*, 2010.

[167] M. Seaborn and T. Dullien, "Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges," 2015. [Online]. Available: http://googleprojectzero.blogspot.com.tr/2015/03/exploiting-dram-rowhammer-bug-to-gain.html

[168] C. R. (Senate), "The National Security Aspects Of The Global Migration Of The U.S. Semiconductor Industry," https://irp.fas.org/congress/2003_cr/s060503.html, 2003, Accessed: 2023-08-24.

[169] S. H. H. Shadmehri, A. BanaGozar, M. Kamal, S. Stuijk, A. Afzali-Kusha, M. Pedram, and H. Corporaal, "SySCIM: SystemC-AMS Simulation of Memristive Computation In-Memory," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022.

[170] D. Sisejkovic and R. Leupers, *Logic Locking: A Practical Approach to Secure Hardware*. Springer International Publishing, 2023.

[171] S. Son, C. La Torre, A. Kindsmüller, V. Rana, and S. Menzel, "A Study of the Electroforming Process in 1T1R Memory Arrays," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[172] W. Song and P. Liu, "Dynamically Finding Minimal Eviction Sets Can Be Quicker Than You Think for Side-Channel Attacks against the LLC," pp. 427–442, 2019.

[173] R. Spreitzer, V. Moonsamy, T. Korak, and S. Mangard, "Systematic Classification of Side-Channel Attacks: A Case Study for Mobile Devices," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 465–488, 2018.

[174] F. Staudigl, H. Al Indari, D. Schön, H.-Y. Chen, D. Sisejkovic, J. M. Joseph, V. Rana, S. Menzel, A. Hagelauer, and R. Leupers, "It's Getting Hot in Here: Hardware Security Implications of Thermal Crosstalk on ReRAMs," *IEEE Transactions on Reliability*, pp. 1–15, 2024.

[175] F. Staudigl, T. Fetz, R. Pelke, D. Sisejkovic, J. M. Joseph, L. Bolzani Pöhls, and R. Leupers, "Fault Injection in Native Logic-in-Memory Computation on Neuromorphic Hardware," in *Design Automation Conference (DAC)*. IEEE, 2023.

[176] F. Staudigl, T. Fetz, R. Pelke, D. Sisejkovic, J. M. Joseph, L. B. Pöhls, and R. Leupers, "Invited Paper: A Holistic Fault Injection Platform for Neuromorphic Hardware," in *Latin American Test Symposium (LATS)*. IEEE, 2023.

[177] F. Staudigl, M. Hossein, T. Ziegler, H. Al Indari, R. Pelke, S. Siegel, D. J. Wouters, D. Sisejkovic, J. M. Joseph, and R. Leupers, "Work-in-Progress: A Universal Instrumentation Platform for Non-Volatile Memories," in *International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES/ISSS '23 Companion. ACM, Sep. 2023.

[178] F. Staudigl, H. A. Indari, D. Schon, D. Sisejkovic, F. Merchant, J. M. Joseph, V. Rana, S. Menzel, and R. Leupers, "NeuroHammer: Inducing Bit-Flips in Memristive Crossbar Memories," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022.

[179] F. Staudigl, F. Merchant, and R. Leupers, "A Survey of Neuromorphic Computing-in-Memory: Architectures, Simulators, and Security," *IEEE Design & Test*, vol. 39, no. 2, pp. 90–99, 2022.

[180] F. Staudigl, K. J. X. Sturm, M. Bartel, T. Fetz, D. Sisejkovic, J. M. Joseph, L. B. Pohls, and R. Leupers, "X-Fault: Impact of Faults on Binary Neural Networks in Memristor-Crossbar Arrays with Logic-in-Memory Computation," in *International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2022.

[181] N. STMicroelectronics, "STM32H745ZI - High-performance and DSP with DP-FPU, Dual core Arm Cortex-M7+ Cortex-M4 MCU with 2MBytes of Flash memory, 1MB RAM, 480 MHz CPU, Art Accelerator, L1 cache, external memory interface, large set of peripherals, SMPS," https://www.st.com/resource/en/datasheet/stm32h745zi.pdf, accessed: 2024-01-20.

[182] J. Szefer, "Survey of Microarchitectural Side and Covert Channels, Attacks, and Defenses," *Journal of Hardware and Systems Security*, vol. 3, no. 3, pp. 219–234, 2018.

[183] M. Tahoor, "Reliable Computing I, Lecture 3: Faults, Errors, Failures," https://cdnc.itec.kit.edu/downloads/lecture3-reliable-computing-1-2016-2017.pdf, 2016, Accessed: 2023-08-14.

[184] A. Tang, S. Sethumadhavan, and S. Stolfo, "CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management," in *USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017, pp. 1057–1074.

[185] X. Tang, J. Liu, Y. Shen, S. Li, L. Shen, A. Sanyal, K. Ragab, and N. Sun, "Low-Power SAR ADC Design: Overview and Survey of State-of-the-Art Techniques," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 6, pp. 2249–2262, 2022.

[186] M. Tehranipoor and F. Koushanfar, "A Survey of Hardware Trojan Taxonomy and Detection," *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 10–25, 2010.

[187] U. G. The White House, "CHIPS and Science Act Will Lower Costs, Create Jobs, Strengthen Supply Chains, and Counter China," https://www.whitehouse.gov/briefing-room/statements-releases/2022/08/09/fact-sheet-chips-and-science-act-will-lower-costs-create-jobs-strengthen-supply-chains-and-counter-china/, 2022, Accessed: 2023-08-23.

[188] J. P. Thoma and T. Güneysu, "Write Me and I'll Tell You Secrets - Write-After-Write Effects On Intel CPUs," in *International Symposium on Research in Attacks, Intrusions and Defenses, RAID*. ACM, 2022.

[189] A. C. Torrezan, J. P. Strachan, G. Medeiros-Ribeiro, and R. S. Williams, "Sub-nanosecond Switching of a Tantalum Oxide Memristor," *Nanotechnology*, vol. 22, p. 485203, 2011.

[190] Traco Power North America Inc., "THL 25-2423," https://www.tracopower.com/sites/default/files/products/datasheets/thl25_datasheet.pdf, accessed: 2024-01-20.

[191] Traco Power North America Inc., "TMR 3-2421," https://www.tracopower.com/sites/default/files/products/datasheets/tmr3_datasheet.pdff, accessed: 2024-01-20.

[192] E. I. Vatajelu, P. Prinetto, M. Taouil, and S. Hamdioui, "Challenges and Solutions in Emerging Memory Testing," *IEEE Transactions on Emerging Topics in Computing*, vol. 7, no. 3, pp. 493–506, 2019.

[193] P. Vila, B. Kopf, and J. F. Morales, "Theory and Practice of Finding Eviction Sets," in *IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019.

[194] M. von Witzleben, K. Fleck, C. Funck, B. Baumkötter, M. Zuric, A. Idt, T. Breuer, R. Waser, U. Böttger, and S. Menzel, "Investigation of the Impact of High Temperatures on the Switching Kinetics of Redox-based Resistive Switching Cells Using a Highspeed Nanoheater," *Advanced Electronic Materials*, vol. 3, no. 12, p. 1700294, 2017.

[195] M. von Witzleben, K. Fleck, C. Funck, B. Baumkötter, M. Zuric, A. Idt, T. Breuer, R. Waser, U. Böttger, and S. Menzel, "Investigation of the Impact of High Temperatures on the Switching Kinetics of Redox-based Resistive Switching Cells Using a Highspeed Nanoheater," *Adv. Electron. Mat.*, vol. 3, no. 12, p. 1700294, 2017.

[196] M. von Witzleben, T. Hennen, A. Kindsmüller, S. Menzel, R. Waser, and U. Böttger, "Study of the Set Switching Event of VCM-based Memories on a Picosecond Timescale," *Journal of Applied Physics*, vol. 127, no. 20, p. 204501, 2020.

[197] W. Wan, R. Kubendran, C. Schaefer, S. B. Eryilmaz, W. Zhang, D. Wu, S. Deiss, P. Raina, H. Qian, B. Gao, S. Joshi, H. Wu, H.-S. P. Wong, and G. Cauwenberghs, "A Compute-In-Memory Chip Based on Resistive Random-Access Memory," *Nature*, vol. 608, no. 7923, pp. 504–512, 2022.

[198] Z. Wang, F. hsuan Meng, Y. Park, J. K. Eshraghian, and W. D. Lu, "Side-Channel Attack Analysis on In-Memory Computing Architectures," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–13, 2023.

[199] R. Waser and M. Aono, "Nanoionics-based Resistive Switching Memories," *Nature Materials*, vol. 6, no. 11, pp. 833–840, 2007.

[200] R. Waser, R. Dittmann, G. Staikov, and K. Szot, "Redox-based Resistive Switching Memories - Nanoionic Mechanisms, Prospects, and Challenges," *Advanced Materials*, vol. 21, no. 25-26, pp. 2632–2663, 2009.

[201] J. Wen, A. Baroni, E. Perez, M. Uhlmann, M. Fritscher, K. KrishneGowda, M. Ulbricht, C. Wenger, and M. Krstic, "Towards Reliable and Energy-Efficient RRAM Based Discrete Fourier Transform Accelerator," in *Design, Automation &; Test in Europe Conference &; Exhibition (DATE)*. IEEE, 2024, pp. 1–6.

[202] L. Xia, W. Huangfu, T. Tang, X. Yin, K. Chakrabarty, Y. Xie, Y. Wang, and H. Yang, "Stuck-at Fault Tolerance in RRAM Computing Systems," *IEEE Journal*

on Emerging and Selected Topics in Circuits and Systems, vol. 8, no. 1, pp. 102–115, 2018.

[203] L. Xia, B. Li, T. Tang, P. Gu, P.-Y. Chen, S. Yu, Y. Cao, Y. Wang, Y. Xie, and H. Yang, "MNSIM: Simulation Platform for Memristor-based Neuromorphic Computing System," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2017.

[204] Q. Xia, W. Robinett, M. W. Cumbie, N. Banerjee, T. J. Cardinali, J. J. Yang, W. Wu, X. Li, W. M. Tong, D. B. Strukov, G. S. Snider, G. Medeiros-Ribeiro, and R. S. Williams, "Memristor-CMOS Hybrid Integrated Circuits for Reconfigurable Logic," *Nano Letters*, vol. 9, no. 10, pp. 3640–3645, 2009.

[205] Q. Xia and J. J. Yang, "Memristive Crossbar Arrays for Brain-Inspired Computing," *Nature materials*, vol. 18, no. 4, pp. 309–323, 2019.

[206] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware Trojans: Lessons Learned after One Decade of Research," *Transactions on Design Automation of Electronic Systems*, vol. 22, no. 1, pp. 1–23, 2016.

[207] T. P. Xiao, C. H. Bennett, B. Feinberg, M. J. Marinella, and S. Agarwal, "CrossSim: Accuracy Simulation Of Analog In-Memory Computing." [Online]. Available: https://github.com/sandialabs/cross-sim

[208] XP Power Ltd., "JCH1024," https://www.xppower.com/portals/0/pdfs/SF_JCH10.pdf, accessed: 2024-01-20.

[209] C. Xu, D. Niu, Y. Zheng, S. Yu, and Y. Xie, "Impact of Cell Failure on Reliable Cross-Point Resistive Memory Design," *ACM Transactions on Design Automation of Electronic Systems*, vol. 20, no. 4, pp. 1–21, 2015.

[210] M. Xue, C. Gu, W. Liu, S. Yu, and M. O'Neill, "Ten Years of Hardware Trojans: A Survey From The Attacker's Perspective," *Computers & Digital Techniques IET*, vol. 14, no. 6, pp. 231–246, 2020.

[211] J. J. Yang, M. D. Pickett, X. Li, D. A. A. Ohlberg, D. R. Stewart, and R. S. Williams, "Memristive Switching Mechanism for Metal/oxide/metal Nanodevices," *Nature Nanotechnology*, vol. 3, no. 7, pp. 429–433, 2008.

[212] K. Yang, M. Hicks, Q. Dong, T. Austin, and D. Sylvester, "A2: Analog Malicious Hardware," in *IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016.

[213] S. Yasuda, K. Ohba, T. Mizuguchi, H. Sei, M. Shimuta, K. Aratani, T. Shiimoto, T. Yamamoto, T. Sone, S. Nonoguchi, J. Okuno, A. Kouchiyama, W. Otsuka, and K. Tsutsui, "A Cross Point Cu-ReRAM with a Novel Ots Selector for Storage Class Memory Applications," in *Symposium on VLSI Technology*. IEEE, 2017.

[214] W. Ye, L. Wang, Z. Zhou, J. An, W. Li, H. Gao, Z. Li, J. Yue, H. Hu, X. Xu, J. Yang, J. Liu, D. Shang, F. Zhang, J. Tian, C. Dou, Q. Liu, and M. Liu, "A 28-nm RRAM Computing-in-Memory Macro Using Weighted Hybrid 2T1R Cell Array and Reference Subtracting Sense Amplifier for AI Edge Inference," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 10, pp. 2839–2850, 2023.

[215] S. Yoneda, S. Ito, Y. Hayakawa, Z. Wei, S. Muraoka, R. Yasuhara, K. Kawashima, A. Himeno, and T. Mikawa, "Newly Developed Process Integration Technologies for Highly Reliable 40 Nm ReRAM," *Japanese Journal of Applied Physics*, vol. 58, pp. SBBB06/1–8, 2019.

[216] B. Zhang, N. Uysal, D. Fan, and R. Ewetz, "Handling Stuck-At-Faults In Memristor Crossbar Arrays Using Matrix Transformations," in *ACM Asia and South Pacific Design Automation Conference*, 2019.

[217] Y. Zhang, D. Feng, W. Tong, Y. Hua, J. Liu, Z. Tan, C. Wang, B. Wu, Z. Li, and G. Xu, "CACF: A Novel Circuit Architecture Co-Optimization Framework for Improving Performance, Reliability and Energy of ReRAM-based Main Memory System," *ACM Transactions on Architecture and Code Optimization*, vol. 15, no. 2, pp. 1–26, 2018.

[218] Y. Zhang, Y. Shen, X. Wang, and Y. Guo, "A Novel Design for a Memristor-Based OR Gate," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 8, pp. 781–785, 2015.

[219] Y. Zhang, Z. Yu, L. Gu, C. Wang, and D. Feng, "EnTiered-ReRAM: An Enhanced Low Latency and Energy Efficient TLC Crossbar ReRAM Architecture," *IEEE Access*, vol. 9, pp. 167 173–167 189, 2021.

[220] Y. Zheng, C. Xu, and Y. Xie, "Modeling Framework For Cross-Point Resistive Memory Design Emphasizing Reliability And Variability Issues," in *IEEE Asia and South Pacific Design Automation Conference*, 2015.

[221] Z. Zhu, H. Sun, K. Qiu, L. Xia, G. Krishnan, G. Dai, D. Niu, X. Chen, X. S. Hu, Y. Cao, Y. Xie, Y. Wang, and H. Yang, "MNSIM 2.0: A Behavior-Level Modeling Tool for Memristor-based Neuromorphic Computing Systems," in *Great Lakes Symposium on VLSI*. ACM, 2020.

[222] H. Ziade, R. A. Ayoubi, and R. Velazco, "A Survey On Fault Injection Techniques," *The International Arab Journal of Information Technology*, vol. 1, no. 2, pp. 171–186, 2004.

Curriculum Vitae

Name	Felix Staudigl
Geburtsdatum	13.01.1994
Geburtsort	Ingolstadt
2000–2004	Grundschule
2004–2010	Realschule
2010–2012	Fachoberschule
2012–2016	Duales Studium (Siemens AG & Technische Hochschule Nürnberg)
Juli 2015	IHK-Abschluss zum Elektroniker für Automatisierungstechnik
August 2016	Abschluss B.Eng Elektro- und Informationstechnik
2016–2021	T.I.M.E.-Programm (RWTH Aachen & CTU Prag)
Juni 2019	Abschluss M.Sc (CTU)
August 2021	Abschluss M.Sc (RWTH)
Seit Januar 2020	Wissenschaftlicher Angestellter am Lehrstuhl für Software für Systeme auf Silizium (SSS) an der RWTH Aachen