Exploration of security threats in In-Memory Computing Paradigms

Pietro Inglese

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : EEATS - Electronique, Electrotechnique, Automatique, Traitement du Signal (EEATS)
Spécialité : Nano électronique et Nano technologies
Unité de recherche : Techniques de l'Informatique et de la Microélectronique pour l'Architecture des systèmes intégrés

Exploration des menaces de sécurité dans les paradigmes de calcul en mémoire

Exploration of security threats in In-Memory Computing Paradigms

Présentée par :

Pietro INGLESE

Direction de thèse :

Giorgio DI NATALE DIRECTEUR DE RECHERCHE, Université Grenoble Alpes	Directeur de thèse
Elena-Ioana VATAJELU CHARGEE DE RECHERCHE, Université Grenoble Alpes	Co-directrice de thèse

Rapporteurs :

Alberto BOSIO
PROFESSEUR DES UNIVERSITES, École centrale de Lyon
Jean-Michel PORTAL
PROFESSEUR DES UNIVERSITES, IM2NP, Aix-Marseille Université

Thèse soutenue publiquement le **7 décembre 2023**, devant le jury composé de :

Giorgio DI NATALE DIRECTEUR DE RECHERCHE, CNRS	Directeur de thèse
Alberto BOSIO PROFESSEUR DES UNIVERSITES, École centrale de Lyon	Rapporteur
Jean-Michel PORTAL PROFESSEUR DES UNIVERSITES, IM2NP, Aix-Marseille Université	Rapporteur
Marie-lise FLOTTES CHARGEE DE RECHERCHE, CNRS	Examinatrice
Vincent BEROULLE PROFESSEUR DES UNIVERSITES, Grenoble INP - Esisar, UGA	Président

Invités :

Arnaud VIRAZEL
PROFESSEUR DES UNIVERSITES, Université de Montpellier / LIRMM
Bastien GIRAUD
SCIENTIST, Commissariat à l'énergie atomique et aux énergies alternatives

To all those who have enabled my personal growth,
encouraged me to strive for betterment,
and provided me with unwavering support.

Pietro

Contents

List of Figures

List of Tables

Summary

Computation in Memory (CIM) is a groundbreaking concept that involves performing computations directly within the memory itself, eliminating the need to transfer data back and forth between the memory and Central Processing Unit (CPU). This approach deviates from the traditional Von Neumann architecture, aiming to overcome its limitations and bottlenecks, ultimately driving technological advancement.

Various CIM solutions leverage existing memory technologies, capitalising on physical attributes, organisation, peripheral components, and control logic of the memory. These paradigms enable logic and arithmetic operations within the memory, significantly reducing latency and energy consumption by eliminating data transfers to the CPU. Furthermore, they pave the way for enhanced parallelism through dense arrays of memory elements that support computation.

The thesis explores the potential of memristive-based memories to redefine the integration of logic and memory. The research delves into different LIM implementations, analysing their advantages and disadvantages, with a particular focus on security considerations.

The primary objective is to efficiently synthesise a range of Boolean operations within the LIM framework, from basic 2-bit operations to the Full Adder. Simultaneously, the thesis introduces a Simulation and Analysis environment to support parallel simulations and parametric sweep. The research work compares various LIM technologies in terms of memory resource requirements and the number of operations needed to implement fundamental Boolean functions. These preliminary findings highlight the potential of memristive-based LIM technologies, but they also emphasise the importance of the respect of electrical characteristics and operation times.

To gain a deeper understanding of LIM solutions, we developed a toolkit to easily analyse electrical behaviour, generate netlist inputs, run Cadence simulations, and collect simula-

tion data. Thanks to this automatic tool, we revealed some weaknesses of certain solutions, particularly concerning deviations in output memristor resistance from ideal values. The research explored therefore operations under non-ideal conditions, focusing on the role of input memristor resistance in determining correct operation ranges. It highlights the potential for incorrect results when chaining operations, especially from non-ideal inputs, prompting the consideration of refresh cycles for stability.

The thesis finally explored security properties of such LIM solutions, by looking at side-channel and fault analyses. The study revealed several vulnerabilities, mainly due to the large resistive difference between Low Resistive State (LRS) and High Resistive State (HRS), and the lack of robustness when operations are performed under non-ideal conditions.

In summary, the research explores the exciting possibilities of memristive-based LIM, offering insights into their potential, but also showing their limits and vulnerabilities.

Sommaire

Le calcul en mémoire est un concept révolutionnaire qui consiste à effectuer des calculs directement dans la mémoire elle-même, éliminant ainsi la nécessité de transférer des données entre la mémoire et l'unité centrale. Cette approche s'écarte de l'architecture Von Neumann traditionnelle et vise à en surmonter les limites et les goulets d'étranglement, pour finalement favoriser le progrès technologique.

Diverses solutions de calcul en mémoire exploitent les technologies de mémoire existantes, en capitalisant sur les attributs physiques, l'organisation, les composants périphériques et la logique de contrôle de la mémoire. Ces paradigmes permettent d'effectuer des opérations logiques et arithmétiques au sein de la mémoire, ce qui réduit considérablement la latence et la consommation d'énergie en éliminant les transferts de données vers l'unité centrale de traitement. En outre, ils ouvrent la voie à un parallélisme accru grâce à des réseaux denses d'éléments de mémoire qui prennent en charge le calcul.

La thèse explore le potentiel des mémoires à base de memristors pour redéfinir l'intégration de la logique et de la mémoire. La recherche plonge dans différentes implémentations de Logique en Mémoire (LIM), analysant leurs avantages et leurs inconvénients, en mettant particulièrement l'accent sur les aspects de sécurité.

L'objectif principal est de synthétiser efficacement une gamme d'opérations booléennes. Parallèlement, la thèse introduit un environnement de simulation et d'analyse pour permettre les simulations parallèles et les balayages paramétriques. Le travail de recherche compare diverses technologies LIM en termes de besoins en ressources mémoire et du nombre d'opérations nécessaires pour mettre en œuvre des fonctions booléennes fondamentales. Ces résultats préliminaires mettent en lumière le potentiel des technologies LIM à base de memristors, tout en soulignant l'importance du respect des caractéristiques électriques et des temps d'opération.

Pour mieux comprendre les solutions LIM, nous avons développé une boîte à outils permettant d'analyser facilement le comportement électrique, de générer des entrées de netlist, d'exécuter des simulations Cadence et de recueillir des données de simulation. Grâce à cet outil automatisé, nous avons révélé certaines faiblesses de certaines solutions, en particulier en ce qui concerne les écarts de résistance des memristors par rapport aux valeurs idéales.

La recherche a donc exploré les opérations dans des conditions non idéales, en mettant l'accent sur le rôle de la valeur de la résistance initiale du memristor dans la détermination des plages d'opération correctes. Elle met en évidence le risque d'obtenir des résultats incorrects lors de l'enchainement d'opérations, en particulier à partir de valeurs initiaux non idéaux des memristors, ce qui incite à envisager des cycles de rafraîchissement pour la stabilité des opérations.

Finalement, la thèse a exploré les propriétés de sécurité de ces solutions LIM, en examinant les analyses des canaux auxiliaires et des attaques en fautes. L'étude a révélé plusieurs vulnérabilités, principalement dues à la grande différence de résistance entre les états de faible résistance (LRS) et les états de haute résistance (HRS), ainsi qu'au manque de robustesse lorsque les opérations sont effectuées dans des conditions non idéales.

En résumé, la recherche explore les possibilités de la technologie LIM à base de memristors, en offrant des perspectives sur leur potentiel, tout en montrant également leurs limites et leurs vulnérabilités.

1

Introduction

1.1 Background & Context

The progress in the semiconductor industry has stimulated numerous innovations and improvements in all of its domains, from devices to computing paradigms. The evolution of transistors has been described by Gordon Moore, the co-founder of Intel and Fairchild Semiconductors, stating in 1965 that the number of transistors on a microchip doubles about every year (occupying the same area); hence, the cost of microchip is halved [4]. According to Moore, this operating principle and, at the same time, commitment, should have gone on for at least ten years. Ten years later, in 1975, Moore revised the forecast to grow double every almost two years [5]. Over time, this simple prediction became a law, taking the name of its inventor: Moore's law. This law has become a guideline for the semiconductors industry to trace a path for the evolution of computers.

In 1974, Robert H. Dennard at IBM stated the so-called Dennard scaling [6], where, as the size of Metal-Oxide-Semiconductor Field-Effect Transistor (MOS) transistors shrinks, the power density remains constant, and therefore the power consumption will remain proportional with the area. Furthermore, according to Dennard, each new process generation

1

would lead to a 0.7x reduction of the Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET) gate length and width, the power supply voltage, and the gate oxide. The end of Dennard scaling arrived in 2003 when the introduction of multi-core processors guaranteed the continuity of Moore's law. Presently, CMOS technology is close to its physical limits and, therefore, it is harder to scale down the size and improve the performance: we are witnessing the end of Moore law [7].

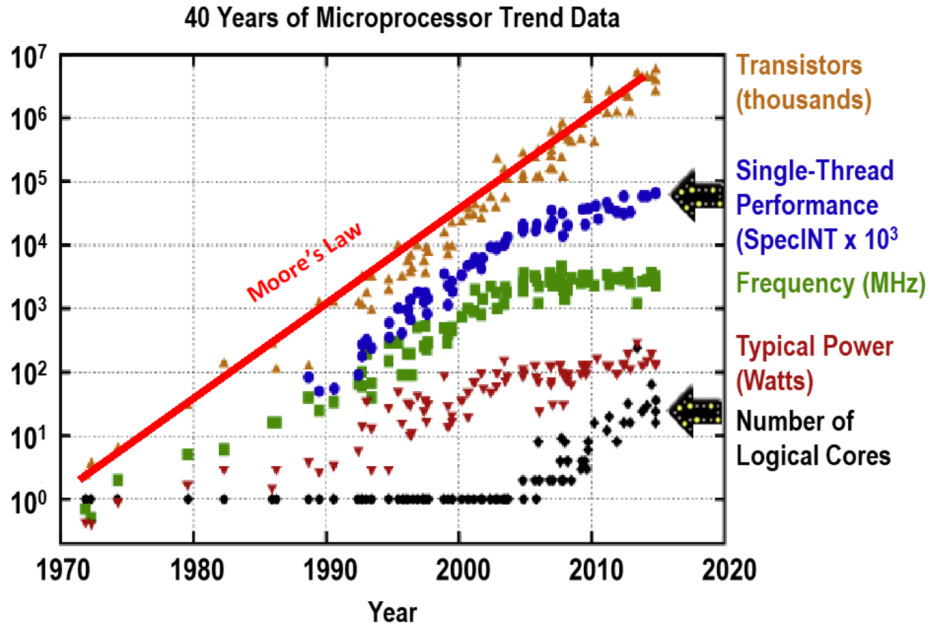


Figure 1.1: IRDS - 40 years of Microprocessor Trend Data. From [1].

In parallel to the MOSFETs, computer architectures evolved as well: since the appearance of modern computers, the widely adopted computing architecture has been based on the separation between the computing unit (or processor) and the memory, i.e., the Von Neumann architecture [8]. This architecture implies that a single bus is connecting the processor and the memory, meaning that the memory is accessed at least twice for every instruction (fetch of the instruction, read/write of data). This problem is known as the Von Neumann bottleneck [9].

The first solution that has been adopted to address the Von Neumann bottleneck is the

Harvard architecture [10], where program and data are stored on separate memories that are connected to the processor by two separate buses. However, this solution has limited efficiency, as in modern computers, the separation between processor and memory has become an issue due to the uneven evolution of processing speed and memory access times (also known as the memory wall) [11, 12]. This inability of the memory to match the speed of processors has been an issue since the early '70s and has first been addressed with the introduction of the cache memory [13]: a faster memory used to store a subset of the main memory content. The partitioning of the memory content between the cache (fast and high cost) and main memory (slower and cheaper) is the best approach to guarantee high performance and low cost.

However, with technological advancements, the memory wall continued to be an increasingly important issue. This has led to the introduction of various levels of cache to compensate for the speed disparity while maintaining low cost. The movement of data becomes ubiquitous in CPU operations, where parts of data are moved from memory to processor, processed, and finally sent back. This data movement is currently costlier in terms of power than the computation itself, e.g., a DRAM access consumes 200 times more energy than a floating point operation [14].

Another important issue regarding the power efficiency is related to static power. Today, the main memories are implemented in DRAM (compact and cheap), while caches are implemented in SRAM (fast and expensive). These are both volatile memories, i.e., in order to retain information, they need to be connected to a power supply, thus consuming energy even when the memory is not accessed. In fact, for small technology nodes (below 32 nm), static power consumption has become dominant over dynamic power consumption. These power efficiency issues are known as the power wall [15]. A solution to the static power problem could be to change the technology adopted and switch to the use of Non-Volatile Memories (NVMs), which will significantly reduce the static power consumption since the stored data are kept even in the absence of a power supply. However, traditional flash memories are too slow to replace the DRAM as the main memory or the SRAM as the cache memory. In this context, research has been conducted on different materials and devices to identify NVM solutions with speed matching the requirements of

1

the main memory or even low levels of cache. Among the best candidates, there are the so-called emerging memories, which include Resistive Random Access Memory (RRAM) [16], Phase Change Memory (PCM) [17], Spin-Transfer Torque Magnetic Random Access Memory (STT-MRAM)) [18] and Ferroelectric Field-Effect Transistor (FeFET) [19]. These are all resistive-based memories, i.e., the information is stored as a resistance value and not as a charge value, as is the case for conventional CMOS memories. The NVMs, already tackling issues such as power consumption, also showed interesting physical capabilities. Indeed, changing from charge-based (as SRAM or DRAM) to resistive-based memories enables new scenarios where the physical capabilities of the memory can be exploited to perform logic or arithmetic operations directly inside the memory array, therefore bypassing the memory wall via the CIM paradigm. Emerging memory technologies, along with new computation paradigms, can in fact provide a convenient way to mitigate the limitations of traditional memories: moving the computation inside the memory tackles both the memory and the power wall. In fact, having the possibility to perform operations directly inside the memory means being able to avoid the data movement to and from the CPU, saving time and energy. Among the existing CIM solutions, the ones based on memristors are of particular interest, since these are the only devices, up to now, which offer the possibility to perform computation and store the result within the memory array without any need for external manipulation of data. In addition, the memristor (used for RRAMs) has the advantage of having a very simple structure, being compatible with the CMOS process, having low area occupancy and theoretical high switching speed, and being non-volatile. Depending on the target application, two main CIM approaches exist: (i) analog-CIM mostly developed for accelerating Multiply-Accumulate (MAC) operations, and (ii) digital-CIM mostly used for performing Boolean operations. The focus of this thesis is on digital-CIM, more precisely on LIM.

The remainder of the chapter is organised as the following: an overview of the basic computer architecture is presented, followed by a foreword to the main memory technologies, and an introduction on the different types of CIM solutions. Finally, we introduce hardware security aspects.

1.1.1 Computer Architecture

The fundamental components of a computer architecture (see figure 1.2) are the CPU, memory, and Input/Output (I/O). The system's main component, the processor (CPU), is in charge of executing instructions. Data and processor instructions are stored in memory. I/O is in charge of transferring data from between the processor and external devices like a printer or a keyboard. Additional parts of a computer system are the bus, which connects the other parts of the architecture, and the instruction pipeline, which tracks the instructions that the processor is following. The CPU is responsible for executing instructions and managing the operations of the system. For our scope, the most interesting part of the computer architecture is the memory.

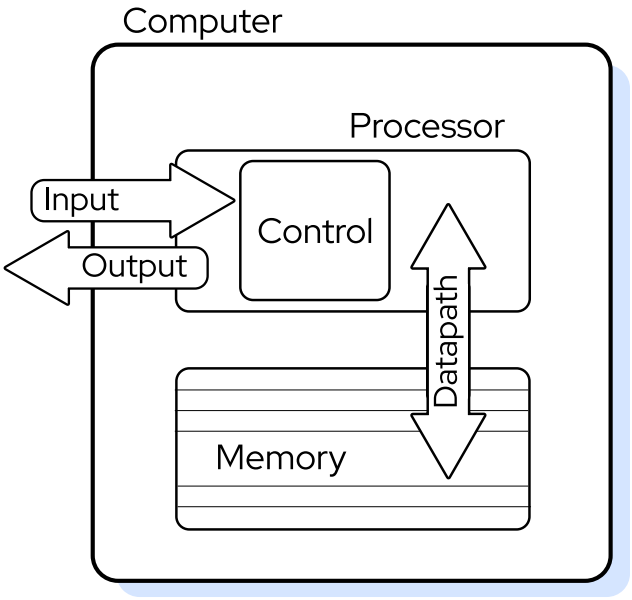


Figure 1.2: Fundamental components in a computer.

1.1.2 Computer Memory Technologies

This section presents the memory, its types, and their usage. It is followed by a brief overview of the various memory technologies and how CIM can be enabled with them. The goal is to highlight the differences and capabilities of the possible implementations,

1

showing the issues and the limitations of each memory. There are several types of memory present inside a computer:

- CPU registers,
- Read Only Memory (ROM),
- Cache memory,
- Primary / Main memory,
- Secondary memory / Mass storage.

The CPU registers are high-speed memories built in the processor to enable the fastest data access. These are groups of flip-flops (latch-type circuit with two stable states, representing logic 1 and 0) [20]. ROM, or permanent memory, is a non-volatile memory. It is non-modifiable and it is used to perform the Power On Self Test (POST): the diagnostic test done at the computer start-up. Finally, secondary memory, also known as external memory, it is a memory that is not directly connected to the CPU and that can often be attached and connected as the user wishes. Common types of secondary memories are Optical Drive memories (CD-ROM, CD-RROM, DVD, etc.), Magnetic storage memories (Hard Disk Drive (HDD), magnetic tape, floppy disk) and Solid-State Drive memories (Universal Serial Bus (USB) memory memory stick or USB flash drive).

In the following, there is a more detailed explanation of certain of the above-mentioned memories.

ROM

ROM (Read Only Memory), or permanent memory is a non-volatile memory that cannot be modified by the user. It is where the operating instructions of the system are stored. The data that is stored inside these memories, as the name says, can only be read. It uses fuses that can be set to store a certain piece of data. Therefore, the data are binary-formatted. It is primarily used to store the Basic Input-Output System (BIOS) and other firmware that are essential for a system to start and function properly. It is typically found in the BIOS chip of a motherboard, which contains the instructions that the computer needs to boot up. ROM memories can be divided into more types, introduced in the next paragraphs.

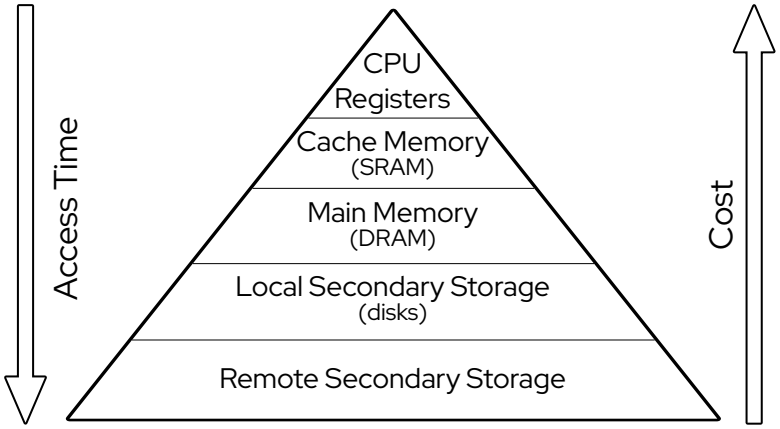


Figure 1.3: Computer Memory types.

PROM Programmable Read-Only Memory (PROM) is a type of ROM that starts without written data on it. It can be written and, therefore, used by means of a PROM programmer. It is only programmable once, and it is typically used to create firmware for a chip.

EPROM Erasable Programmable Read-Only Memory (EPROM) is a type of ROM chip that can be written onto (as the PROM), but it can as well be erased and therefore re-programmed. The erase operation is done by means of Ultraviolet (UV) light exposing it on a quartz crystal window at the top of the chip. The erase operation slightly damages the memory, so there is a limited amount of erase and write operations, and it has to be manually done, making it a time consuming operation.

EEPROM Electrically Erasable Programmable Read-Only Memory (EEPROM) is a type of ROM chip whose goal is substituting the PROM and EPROM chips. It is also erasable, but this time the erasing process is done by means of an electric field: it is quicker and it does not damage the memory. In fact, the chip does not need to be removed from the memory, whereas EPROM chips need to be removed from the computer to be erased.

RAM

Random-Access Memory (RAM) is a vital component of modern computing systems, serving as the primary means of temporary data storage and retrieval. It is a type of volatile

1

memory, meaning that its contents are lost when power is removed from the system.

There are two main types of classic RAM: dynamic RAM (DRAM) and static RAM (SRAM). DRAM is the most widely used type of RAM and is characterised by its ability to be repeatedly written and read. It is also relatively inexpensive compared to SRAM. On the other hand, SRAM is a type of RAM that is faster than DRAM and is primarily used in high-performance systems such as servers and high-end computers.

SRAM is a type of memory that uses a combination of transistors and capacitors to store data. On the other hand, DRAM is a type of memory that uses a capacitor to store data. Unlike SRAM, DRAM needs to be refreshed periodically to retain data, which means that it depends on a supply voltage and that it can preserve information without being refreshed only for a short period of time. This makes DRAM less reliable than SRAM, as it is more susceptible to data loss in the event of a power outage or other failure.

Other RAM types have been studied and developed, to improve performance and reduce costs, and these memories are called Emerging Memories, as introduced.

Among them are RRAM [16], PCM [17], and STT-MRAM) [18].

SRAM SRAM is a bistable circuit that exploits two crossed-coupled inverters to retain an electric charge that represents the stored value. This value will be kept until the opposite value is written in the same cell. The circuit of a 6T SRAM is shown in figure 1.10. It uses two access transistors that allow access to the cell to ensure data retention and perform read and write operations. The SRAM cell is big: its size is more than 60 F^2. As a consequence, the cost of SRAM is high. When the size of the transistors is reduced, the yield drops, and therefore the faster the cell, the more expensive its fabrication is. The read and write access time of an SRAM cell is less than 3 ns. Being a volatile memory, data is retained until the power is on. On the other hand, there is the static power consumption to take into account. The endurance is greater than 10^{15} cycles. SRAM does not need to be refreshed, which means that it can retain data indefinitely as long as power is supplied. This makes SRAM faster and more reliable than DRAM, as it does not suffer from the same performance degradation over time. Another advantage of SRAM is its low power consumption. Because SRAM does not need to be refreshed, it uses less power than DRAM, making it a popular choice for mobile devices such as laptops and smartphones.

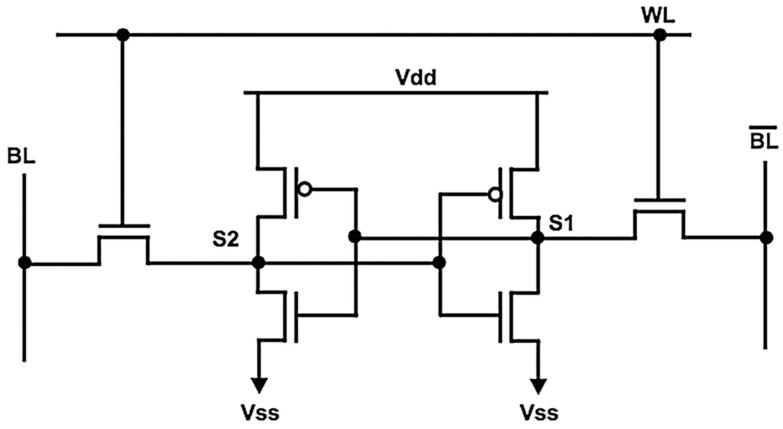


Figure 1.4: A six-transistor (6T) CMOS SRAM cell. From [2].

DRAM DRAM retains the charge thanks to a capacitor whose access is managed through a transistor, as shown in figure 1.10. The capacitor charge represents the value stored in the memory. Although the dimensions of the DRAM circuit are smaller than the SRAM, the access time is higher than the SRAM's, and it suffers of leak currents because of the capacitor, requiring then the value in the memory to be often refreshed (around 50 times per second). The consequence is a slower access time.

The read and write access time of a DRAM cell is between 7 and 20 ns. Being a volatile memory, the data is retained until the power is on. On the other hand, there is the static power consumption to take into account. The endurance is higher than 10^{15} cycles. The cell size is between 4 and 6 F^2.

Despite its drawbacks, DRAM is still the most widely used type of memory in modern computing systems. One of the main reasons for this is its low cost. DRAM is much cheaper to produce than SRAM, which makes it a more cost-effective option for many applications.

Another advantage of DRAM is its high storage capacity. Because DRAM uses a capacitor to store data, it can store more data in a smaller area than SRAM. This makes DRAM a popular choice for use in main memory, where high storage capacity is more important than speed.

In summary, SRAM and DRAM are both RAM types that are used for temporary data

1

storage and retrieval. SRAM is faster and more reliable than DRAM, but it is also more expensive and has lower storage capacity. DRAM is slower and less reliable than SRAM, but is also cheaper and has larger storage capacity. The choice of SRAM or DRAM will depend on the specific needs and requirements of the application. High-performance computing applications such as servers and supercomputers will typically use SRAM, while lower-performance applications such as desktops and laptops will use DRAM.

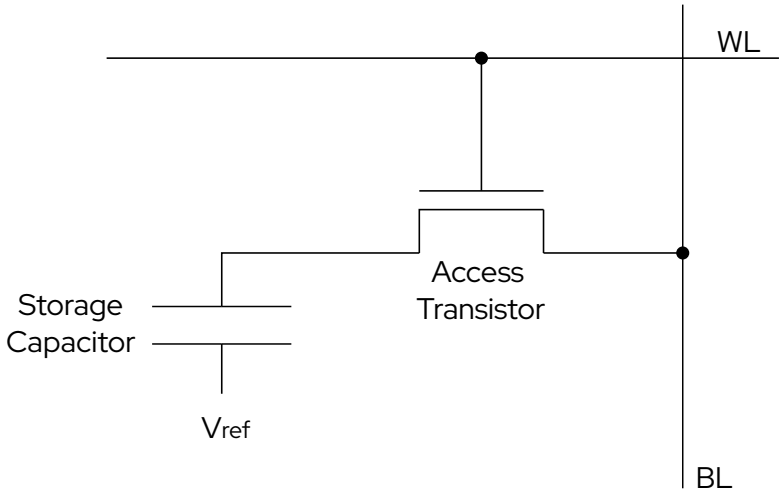


Figure 1.5: DRAM cell.

NVRAM

In the ever-evolving landscape of data storage technologies, the demand for higher capacities, faster access times, and improved energy efficiency has been a driving force behind innovative solutions. Non-Volatile Random-Access Memory (NVRAM) [21] has emerged as a promising class of memory technology that combines the benefits of non-volatility, byte-addressability, and low-latency read/write operations. With the potential to revolutionise data storage and computing systems, NVRAM holds great promise for enabling next-generation persistent memory architectures. NVRAM provides the ability to retain information even in the absence of power. This non-volatility makes NVRAM a desirable option for a wide range of applications, including data centres, enterprise storage, embedded systems, and mobile devices. By bridging the gap between storage and memory,

NVRAM has the potential to overcome the limitations of existing technologies and open up new avenues for data-centric computing.

NVRAM encompasses a variety of memory technologies, each with its own unique characteristics and underlying principles.

Phase Change Memory (PCM): it uses the reversible phase transition of chalcogenide materials between the amorphous and crystalline states. This technology offers fast read and write access times, high endurance, and scalability, making it a promising candidate for future memory systems.

Magnetic Random-Access Memory (MRAM): it uses the magnetic properties of ferromagnetic materials to store data. It offers non-volatility, high endurance, low power consumption, and fast read and write speeds. MRAM has gained significant attention due to its potential for integration with existing complementary metal-oxide-semiconductor (CMOS) processes.

Resistive Random-Access Memory (RRAM): also known as resistive switching memory, it relies on the reversible modulation of resistance in thin film structures. It offers fast switching speeds, low power consumption, and high endurance.

Ferroelectric Random-Access Memory (FeRAM): it utilises ferroelectric materials to store data as polarisation states. It offers non-volatility, low power consumption, fast write speeds, and high endurance. FeRAM holds promise for applications that require frequent data updates and low-latency access.

The emergence of these new NVRAM paradigms presents interesting opportunities to advance the field of persistent data storage. By combining the advantages of non-volatility, byte-addressability, and low-latency access, NVRAM has the potential to redefine memory hierarchies, blur the lines between storage and memory, and fundamentally transform the way data are stored, processed, and retrieved.

RRAM The RRAM element is a variable resistance, whose value represents the bit logic value. It is a 3-layer device, made of two metal electrodes with a dielectric in between. The dielectric is used to conduct by means of a conduction path formed applying a sufficiently high voltage. This is made possible during the fabrication phase, where defects are introduced in a controlled way in the dielectric. There are unipolar and bipolar RRAM:

1

the unipolar switches depending on the amplitude of the voltage applied on the memory, while the bipolar takes into account the voltage polarity as well. The switching, i.e., the change of resistance, takes place with the creation or rupture of conductive filaments inside the dielectric. There are also non-filamentary RRAM devices that exploit different physical systems, like charge trapping and de-trapping and redistribution of oxygen vacancies. Possible operations are set (from HRS to LRS, from logic 0 to logic 1) or reset (from LRS to HRS, from logic 1 to logic 0), which are done, respectively, by forming or breaking the conductive path. A small voltage bias is applied to the cell for reading its data, i.e., to check the resistive state, to then compare the current with a Sense Amplifier (SA). The read and write access time of an RRAM cell is less than 10 ns, with a data retention time greater than 10 years. The endurance is about 10^{12} cycles. The write energy, expressed in Joules per bit, is 0.1 pJ

The RRAM can be implemented via a memristive crossbar. RRAM has garnered interest for its compatibility with CMOS processes and potential for high-density storage.

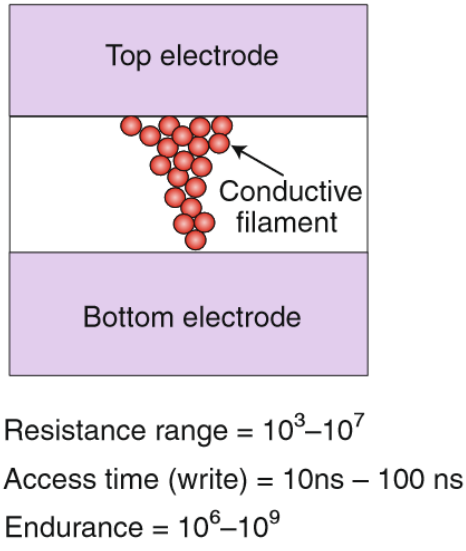


Figure 1.6: An RRAM device at LRS where the Conductive Filament (CF) comprises a large concentration of defects for example oxygen vacancies in metal oxides or metallic ions injected from the electrodes. By the application of appropriate voltage pulses, the defects can be migrated back to the top electrode thus disconnecting the CF and reaching a HRS. From [22].

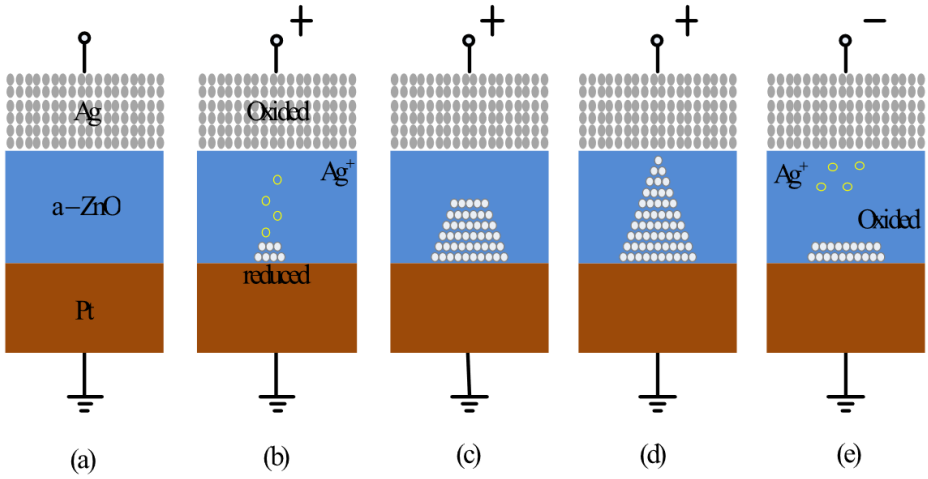


Figure 1.7: Schematic of the switching mechanism of conductive bridge RRAM. a) Pristine state of the RRAM device. b,c) Oxidation of Ag and migration of Ag^+ cations towards cathode and their reduction. d) Accumulation of Ag atoms and Pt electrode leads to growth of highly conductive filament. e) Filament dissolution takes place on applying voltage of opposite polarity. From [16, 23].

PCM The Phase Change Memory (PCM) [17, 24–26] device is, as the RRAM, a variable resistance, and its resistance represents the logic value. It utilises the reversible phase transition of chalcogenide materials between amorphous and crystalline states. It is made by the phase change material, encapsulated inside two electrodes, as shown in figure 1.10. To access the device, a transistor is usually implemented. Phase Change memories exploit materials, typically compounds of Ge, Sb and Te, that can reversibly switch from amorphous and crystalline phases, both having different resistance values. The high resistive state (logic 0) is achieved having the amorphous phase of the memory, while the low resistive state (logic 1) is present during the crystalline phase. The difference between the two states can be up to three or four orders of magnitude. This technology offers fast read and write access times, high endurance, and scalability, making it a promising candidate for future memory systems.

The set and reset operations are performed by applying current pulses through the device. As a consequence, the device heats up. The reset happens once the current pulse that makes the material melt, is stopped abruptly, since it causes the molten material to go

1

into the amorphous phase. The melting temperature is around 600 °C. The set occurs, being the device in amorphous phase, applying a current pulse. This causes the crystallisation of a part of the amorphous region. The crystallisation temperature corresponding the the highest crystallisation rate is ≃ 400 °C. Applying current pulses to a PCM device results in significant heating due to Joule heating. The read happens similarly to the RRAM, applying a small voltage that does not change the inner status of the memory and sensing its current.

The read access time of a Phase Change RAM (PCRAM) cell is less than 10 ns and the write access is 50 ns. The data retention time is higher than 10 years. The endurance is about 10^8 cycles. The write energy, expressed in Joules per bit, is 10 pJ

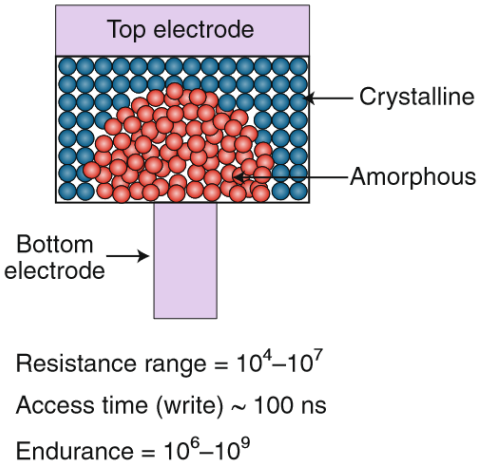


Figure 1.8: A mushroom-type PCM device at HRS where the amorphous phase blocks the bottom electrode. To create this state, a RESET pulse is applied that can melt a significant portion of the phase change material. When the pulse is stopped abruptly, the molten material quenches into the amorphous phase due to glass transition. When a current pulse of lesser amplitude is applied to the PCM device at HRS, a part of the amorphous region crystallises. By fully crystallising the phase change material, the LRS is obtained. From [22].

MRAM The MRAM data storage element is a variable resistance. The Magnetic Tunnelling Junction (MTJ) device is a tunnelling oxide barrier sandwiched in two ferromagnetic layers. One ferromagnetic layer has a fixed magnetic orientation, while the other has free magnetic rotation (free layer).

The conductance, and therefore the logic value, depend on the magnetisation of the layers:

Parallel orientation is the High Conductance State, logic 0, while the Anti-parallel orientation is the Low Conductance State, corresponding to the logic 1. This is the opposite convention with respect to the one used in RRAM and PCM.

To reverse the magnetic orientation (perform write operations), the thermally activated magnetic is used: at finite temperature, there is a finite probability for the magnetisation reverse its direction (Néel-Brown theory). This is also a reliability risk: there is a chance of spontaneous state reversal.

Among the different types of MRAM devices, STT-MRAM is widely used. It exploits a spin-polarised current to flip the magnetisation direction of a ferromagnetic layer. When a current passes through the fixed layer, it becomes spin polarised. Then, passing through the oxide and entering the free layer, it changes the orientation of the polarity of the latter. Writing a logic 0, a current flows from the fixed to the free layer, making the transition from anti-parallel to parallel happen. Writing a logic 1, the current has to flow on the other direction, i.e., from the free to the fixed layer, resulting in a transition from parallel to anti-parallel relative magnetisation.

The read and write access time of a STT-MRAM cell is less than 10 ns. The data retention time is higher than 10 years. The endurance is about 10^{16} cycles. The write energy, expressed in Joules per bit, is 0.1 pJ

1.1.3 Computing in Memory

CIM, as the name suggests, refers to the capability of performing computation directly inside the memory, without therefore resorting to the CPU to have the results and then to write them back to the memory. It represents a non-Von Neumann approach, since it allows to compute in the memory itself. The main idea is to avoid moving data into a processing unit and to therefore break the memory wall. To tackle the issues and bottlenecks we have introduced, a wide variety of solutions exists which are based on existing memory technologies, all of them relying on the physical attributes of the memory, its organisation, its periphery and the control logic. These paradigms enable logic and/or arithmetic operations directly inside the memory boundaries. The operations are indeed performed without the need of transferring data to and from the CPU, thus reducing latency and en-

1

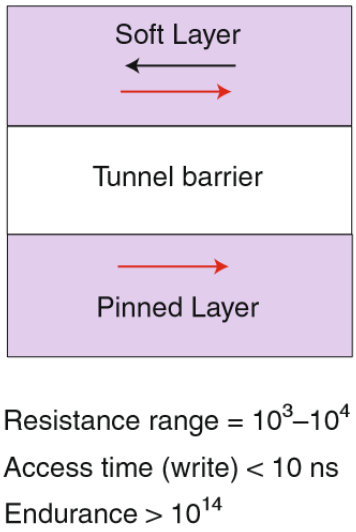


Figure 1.9: An STT-MRAM device with two ferromagnetic layers (pinned and free) separated by a tunnel oxide layer. The magnetic polarisation of the free layer can be changed upon writing. Depending on whether the ferromagnetic polarisations are parallel or anti-parallel, the device assumes a low or high resistance, respectively. The transition to the parallel state takes place directly through conduction electrons, which are previously spin-polarised by the pinned layer. Subsequently, the magnetic polarisation of the free layer is rotated using magnetic momentum conservation. To switch to the anti-parallel state, an opposite voltage, and hence current direction, is employed. From [22].

ergy consumption. Moreover, it lays the foundations for heavily increasing parallelism by having dense arrays of memory elements enabling computation.

Generically, they exploit the physical characteristics of the memory and the insertion of control and, if needed, computational elements in the peripheral logic (e.g., Write Driver (WD) and SA). In fact, the various implementations can be more or less invasive for the architecture and for the memory array: there are CIM solutions that work with any kind of memory, volatile or non-volatile, or that work just with non-volatile memories.

The state of the art is mainly divided on levels of abstraction: (i) device level – to identify the optimum material combination to achieve the desired device behaviour; (ii) circuit level – to design logic gates, a.k.a., LIM [27–31], vector-matrix multiplications [32–34], neuromorphic computing synapse and/or neurons [35–37]; (iii) system level – to map applications on memory arrays, parallelise computations, design accelerators [38]. De-

pending on the solution, CIM can be enabled on different memory technologies: among them, there are the conventional non-volatile and emerging non-volatile memories.

The basic memory array is formed of rows and columns (2-D array) that are independently controlled with memory cells (bit cells) in the cross-points. A simplified view can be seen in figure 1.10. The bit cells contain the information associated to 1 bit of data and the memory cells are connected by means of Wordlines (WLs) and Bitlines (BLs). The bit cells can be connected together in various ways (also depending on the technology used), i.e., there are different types of memory arrays that handle how to access and use the bit cells. The most used types are the 1T1R and the crossbar.

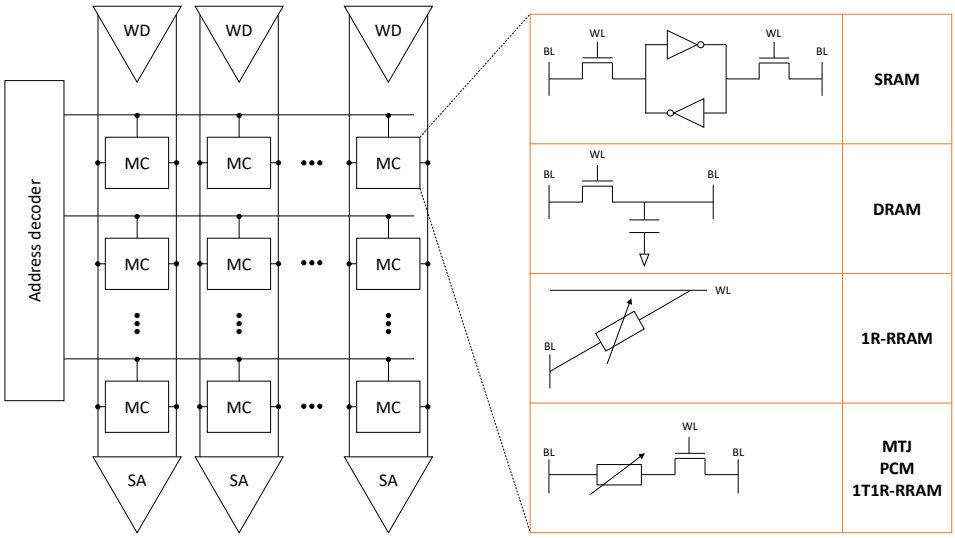


Figure 1.10: Memory array and memory cells.

A generic memory architecture can be described as shown in figure 1.10. A memory cell is selected by means of address decoders, it is written into by the WD and read from with the help of the SA. The voltage levels required to enable the operations on the memory cell are set by the voltage regulators. One memory array communicates with the processor or other memory arrays by means of bus connections. In order to enable CIM, several changes need to be implemented to the memory array or/and to its peripheral circuitry. In this context, the peripheral circuitry consists of standard memory periphery (SA, WD, Address Decoder (AD), etc.) and any additional logic that might be required.

1

Moreover, in some cases, additional logic is added to facilitate computation. Several proposals exist in the literature enabling the computation in memory. Some implementations are generic, and can be used with any memory technology, such as several Near Memory Computing implementations [39]; some can be used with modified CMOS technology to have Computation via Look-up Table (LUT), needing a different memory array; and finally a subcategory of CIM implementations [30, 32, 40–47], modifying the memory periphery, allow to "enhance" a standard memory core with computing capabilities. All the previously said implementations need more or less modifications to parts of the memory array or the periphery, while others take advantage of device physics and are only suitable to be implemented in a specific technology, such as emerging memory technologies.

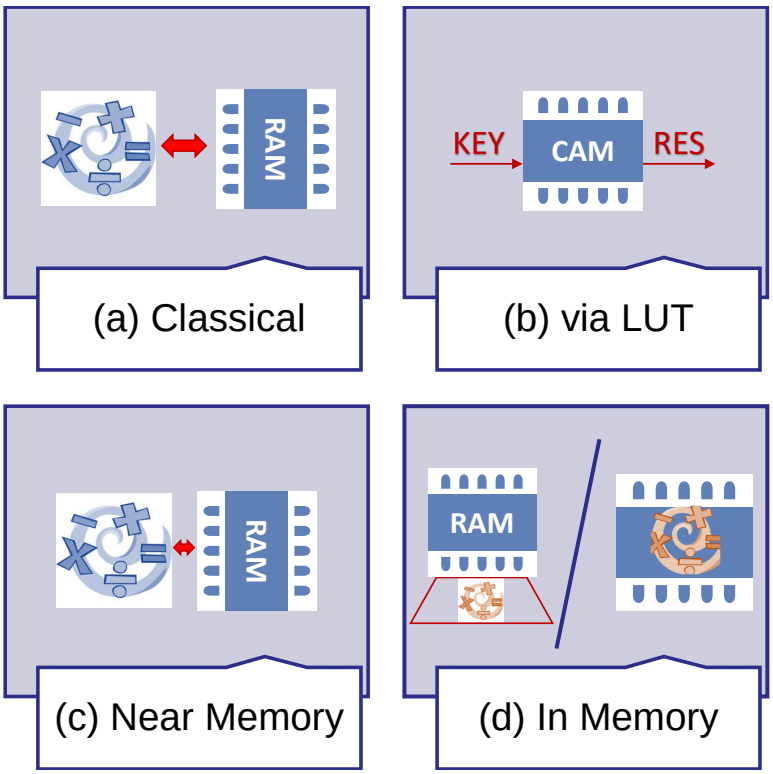


Figure 1.11: (a) Classical Computing, (b) Computation via LUT, (c) Near Memory Computing, (d) CIM (here are illustrated two different paradigms: the one exploiting the periphery and the LIM).

Depending on the way the inputs are stored (the memory content, or an electrical

signal) and where the operations are performed (in the memory array, or in the periphery), the existing solutions can be classified in three main categories:

- Computation via LUT

  A LUT is used to store pre-calculated operations directly in the memory, to avoid moving data to perform operations. This means storing the results of selected operations directly inside the memory, which will be an Application-Specific Integrated Circuit (ASIC). This can be implemented with a CMOS technology. This method can be useful in applications where the same operations are often executed; otherwise, it would be unnecessary and expensive. The state of the art has many scientific articles on this technique and these are the main implementations [48–55].

- Computation Near Memory

  It is often referred to as Processing In Memory (PIM) [56]. The memory core is placed as close as possible to the CPU. It still remains a Von Neumann architecture, but it permits having a shorter bus and therefore decreasing the latency. This can be done by exploiting every main memory technology, but is usually implemented in DRAM technology. The Computation Near Memory reduces the slowdown due to the bus communication speed, but not the speed of the memory itself: the memory wall is therefore not mitigated. The state of the art has many articles and these are the main implementations: [55, 57–63].

- CIM

  CIM enables logic and/or arithmetic operations directly inside the memory boundaries. It allows performing operations without the need of transferring data to/from the CPU, thus saving time and energy. This can be achieved by exploiting the physical characteristics of the memory and/or inserting computational elements into the peripheral logic (SA).

This thesis work is focused on CIM and, more specifically, LIM, since it is the most disruptive and promising solution.

To better explain how a memory can be used to perform CIM operations, in the following are described the operations of a classical memory array and the necessary modi-

fications to make it compatible with CIM.

From the memory array, there are several circuits that enable memory functioning and control.
The WD is in charge of write data on the memory cells, therefore handling the write-1 (SET) and write-0 (RESET) operations at the wanted address. The row and column decoders are used to select the cell corresponding to the wanted address.
The voltage regulator sets the voltages used to carrying out the operations and to assure the correct functioning of the memory. The SAs are used to read the wanted data. Finally, the connection with the external circuitry is allowed by the bus connection, that is composed by the row and column address buses, the data bus and the `write/!read` bit.

The classical memory needs to be modified to enable CIM, based on the technology used and the implemented features. The circuitries that have to be modified (or added) are:

- WD and voltage regulator, to have different voltages to enable CIM operations.
- Row decoder, to simultaneously activate more than one WD.
- SA, to enable logic operations inside it.
- Bus connection to add the `operation` bus to enable the control of logical operations.

1.1.4 Hardware Security

Hardware security is a vital aspect of computer security that involves protecting hardware components from unauthorised access, tampering, and theft. Hardware security measures are designed to ensure that the hardware devices and systems that we rely on every day are secure and can withstand attacks from hackers and cybercriminals.

It is becoming increasingly important, nowadays being crucial for data security and protection; this led to the evolution of hardware-based cryptography and more prominent threats and hardware attacks.

Depending on the type of computing architecture, the cryptographic operations can differ. In classical architectures (see figure 1.13a), data and cryptographic keys are stored

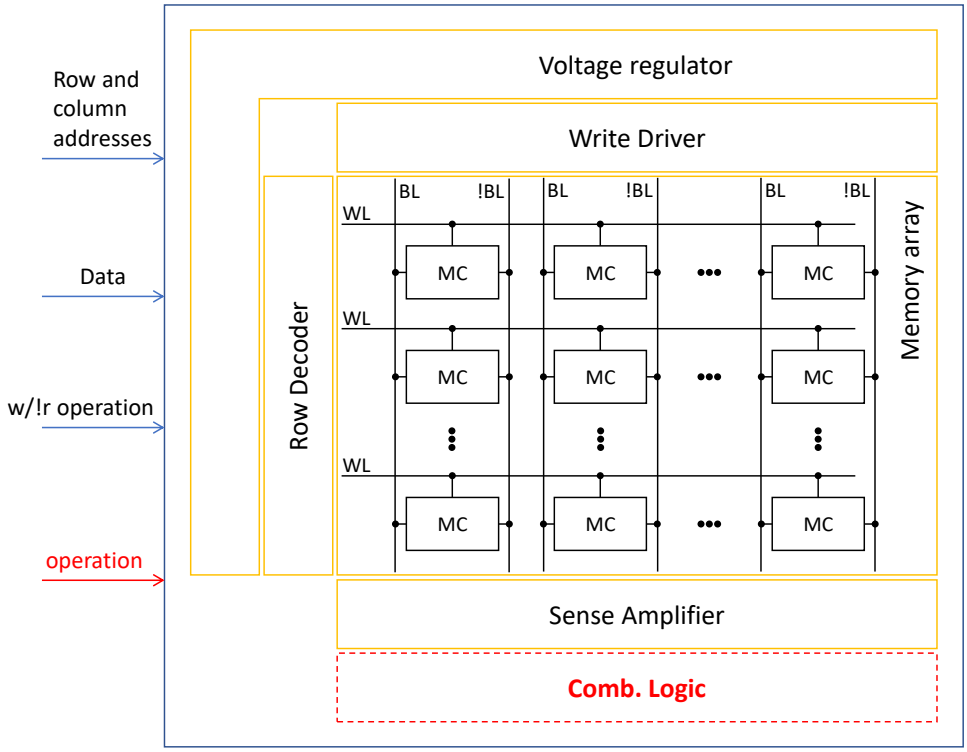


Figure 1.12: Generic memory scheme.

in the main memory, and they are moved towards the processor to execute the cryptographic functions. Therefore, confidential information will transit unencrypted from the memory to the processor through the communication buses, and it could be susceptible of information leakage. In the context of CIM, the encryption and decryption could be performed without resorting to data transfer, therefore mitigating the risk of data leakage and avoiding exposure to attacks. Nonetheless, not all CIM solutions are equal. The CIM solution which works over all technologies, i.e., the Scouting Logic (figure 1.13b) is based on the activation of multiple rows of the memory array to perform logic computation in specially-designed SAs. The result of the computation (that will be an electrical signal, i.e., the output of the SA) will have to be written back to the memory retaining some vulnerability to data leakage. To completely avoid data manipulation outside the memory array, thus reducing observable information leakage, there are in-array computation solutions

1

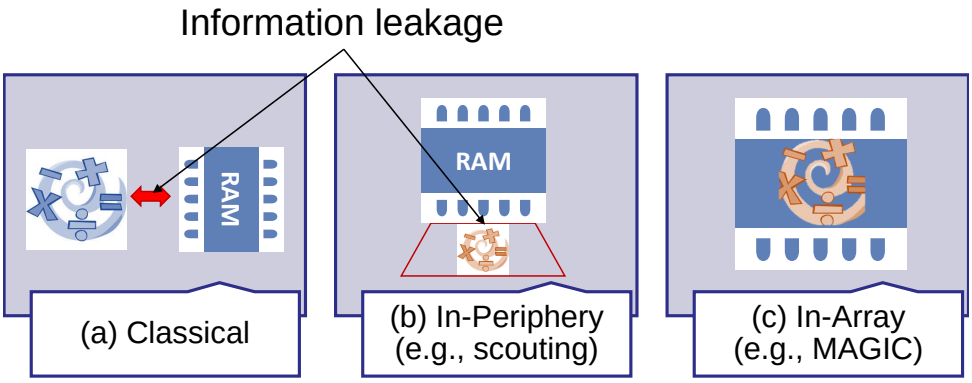


Figure 1.13: (a) Classical computing, (b) IMC In-Periphery, (c) IMC In-Array.

(figure 1.13c). These are the focus of this thesis.

Hardware security threats can take many forms, ranging from physical attacks to software-based attacks. Physical attacks on hardware devices involve attempts to damage or tamper with hardware components, such as memory chips, processors, and storage devices. Physical attacks can include methods such as disassembling a device, exposing it to extreme temperatures or electromagnetic radiation, or using brute force to gain access to the device (among them there are the Fault Attacks (FAs), exploiting erroneous behaviour). On the other hand, side-channel attacks are a type of software-based attack that involves analysing the electromagnetic emissions or power consumption of a device to extract sensitive information, such as cryptographic keys.

Side Channel Attacks

Side Channel Attacks leverage the nuances of physical leakage, exploiting variations in power consumption, electromagnetic emanations, or timing discrepancies to extract sensitive information. Among the Side Channel Attack methodologies there are the Power Analysis Attacks (PAAs), the Electromagnetic Analysis Attacks, Timing Attacks and Acoustic Analysis Attacks.

PAA PAAs [64] constitute one of the most well-studied and impactful categories of side channel attacks. These attacks are rooted in the variations in power consumption exhibited

by a device during its operation. By monitoring the power consumption patterns during cryptographic computations, adversaries can discern critical information, such as secret keys. PAAs are further subdivided into Simple Power Analysis (SPA), Differential Power Analysis (DPA), and Higher-Order Differential Power Analysis, each exploiting different aspects of power consumption fluctuations.

SPA is based on the visual examination of graphs of the current used by the device under exam over time. Variations in power consumption (and therefore in the leakage current) occur as the device performs different operations.

DPA covers a spectrum of methodologies, each exploiting different aspects of power consumption leakage to extract cryptographic secrets. These range from Simple Differential Power Analysis (SDPA) to more advanced approaches such as Higher-Order DPA and Template Attacks. Moreover, the use of machine learning algorithms in DPA is often referred to as Machine Learning DPA and represents the intersection between cutting-edge AI techniques and side channel attacks. SDPA is based on the statistical analysis of the power consumption measured from a device. This attack is more resistant to noise that would make SPA impossible to perform.

Electromagnetic Analysis (EMA) EMA [65, 66] attacks are based on the electromagnetic emanations produced by computing devices during their operation. These emanations, resulting from variations in current flow, can be captured and analysed to extract sensitive information. EMA attacks encompass both electromagnetic radiation analysis and electromagnetic fault injection, demonstrating the potential for adversaries to manipulate hardware behaviour through controlled electromagnetic interference.

Timing Attacks Timing attacks [67] exploit variations in the execution time of cryptographic operations. By meticulously measuring the time it takes a device to perform certain operations, attackers can infer information about the underlying cryptographic computations. Such attacks often exploit subtle timing differences that arise due to conditional branches or memory access patterns.

1

Acoustic analysis attacks Acoustic analysis attacks are a less explored approach and work by analysing the acoustic emissions generated by a computing device during its operation: this can allow adversaries to deduce information about internal computations.

Fault Attacks

FAs [68–71] are a class of attacks that exploit weaknesses in a system by introducing controlled faults or errors in its operation. The goal of a FA is to cause the system to behave in an unintended way, reveal sensitive information, or breach its security measures. In hardware, FAs can be performed by manipulating the physical environment in which the system operates, such as temperature, voltage, electromagnetic radiation, or clock signals. By doing so, an attacker can cause the system to skip or repeat certain operations, change the values of internal variables, or induce hardware failures that can be used to extract secret information. In software, FAs can be performed by injecting faults into the program execution, for example, by modifying the program code, manipulating its input or output data, or exploiting software vulnerabilities. FAs are commonly used to attack cryptographic systems, where they can be used to bypass encryption or signature verification, recover secret keys, or break authentication protocols. To prevent FAs, hardware and software designers need to implement countermeasures that can detect and recover from errors introduced by the attacker or that can make the system less vulnerable to fault injection.

A Differential Fault Attack (DFA) [72, 73] is a type of FAs that is particularly effective against cryptographic systems. DFA exploits the differential behaviour of a system under normal and faulty conditions to recover secret information. In a DFA, the attacker injects a fault into two or more cryptographic computations and compares the results. By analysing the differences between the faulty and correct outputs, the attacker can obtain information about the internal state of the cryptographic algorithm, such as secret keys or intermediate variables. DFA is a powerful attack because it does not require complete knowledge of the system's internal workings but only differential behaviour under faulty conditions. Additionally, DFA can be performed on a single target without requiring multiple attempts or observations. To defend against DFA, cryptographic systems can employ various countermeasures, such as error-correcting codes, redundancy, and randomisation

of inputs and outputs. Hardware and software designers can also use techniques such as duplication, triplication, and comparison of computations to detect and recover from faults. Additionally, cryptographic algorithms can be designed to be resistant to DFA, such as by avoiding branch operations, using constant-time implementations, or limiting the number of computations per key.

Safe Error Attack (SEA) [74, 75] are a type of FA that targets hardware implementations of cryptographic algorithms. In SEA, an attacker introduces faults into the computation of an Error Correcting Code (ECC) algorithm, causing it to generate a ”safe error” that has the effect of reducing the security of the system. The attacker then uses the safe error to recover the private key used by the system. A safe error is a special type of fault that does not cause the system to completely fail or produce incorrect results, but instead introduces a small, intentional error that is difficult to detect. The safe error is designed to bias the internal state of the system in a way that is favourable to the attacker, allowing them to recover the private key more easily. To defend against SEA, hardware designers can use techniques such as error-detection and correction codes, redundancy, and fault tolerance. Additionally, cryptographic algorithms can be designed to be resistant to SEA by avoiding branch operations, using constant-time implementations, or limiting the number of computations per key. Overall, SEA is a relatively new and sophisticated type of fault attack that requires a deep understanding of the internal workings of the cryptographic system. As such, it is important for hardware and software designers to remain vigilant and to implement strong countermeasures to protect against these types of attacks.

1.2 Thesis contributions

As introduced, modern computer architectures have important bottlenecks and issues. The first of them is the Von Neumann Bottleneck, together with the memory and power wall. CIM promises to help solving these issues, and together with emerging non-volatile memories, the resulting technologies can tackle, or at least limit the power wall, together with an increased security. The increased security would come from the whole process hap-

1

pening in the memory instead of resorting to the bus to bring the information back and forth to the CPU, and by having to use the CPU itself to perform the calculation, risking therefore information leakage, hardware trojans and malwares that could sniff and/or observe the data. Therefore, the first choice has been targeting non-volatile memories, with an interested eye on memristive devices. We therefore did a study of the possible CIM implementations, being our goal to choose the most interesting for our use case. Memristive LIM has been the most interesting implementation, for several reasons explained in this manuscript, and we performed several studies on a chosen subset of implementations.

The main thesis contributions are as follows:

- A framework to synthesise Boolean operations on a computing memory array using the state of the art LIM solutions (we show all basic 2-bit operations and the Full Adder) including an efficient and fast method to evaluate the area (number of memristors) and latency (number of cycles and sub-operations needed) of every implementation. This is described in chapter 2.
- Development of a Simulation and Analysis Environment based on an automatic tool that handles parallel simulations to perform parametric sweeps. This is reported in chapter 3.
- Electrical-level design and analysis of selected LIM operations assuming ideal memristors. This is presented in chapter 4.
- Electrical-level design and analysis of selected LIM operations assuming non-ideal memristors and the demonstration, for the first time, of the difficulties and limitations of concatenating logic operation in memory. Details are given in chapter 5.
- Demonstration of MAGIC vulnerabilities to side channel and fault attacks. This is detailed in chapter 6.

2

Logic In Memory (LIM)

This chapter presents an overview of the existing LIM solutions and the technologies enabling them. The chapter starts with a detailed description of the memristive device and its operation. The second part of this chapter presents the state-of-the-art LIM solutions with their descriptions, area and latency evaluation (in terms of required memristors and number of cycles to complete an operation) as well as a comprehensive comparison between them.

2.1 Memristor

In the quest to push the boundaries of electronic memory and computing, researchers have been exploring novel technologies that can overcome the limitations of traditional devices. Among the most promising advancements in this pursuit is the discovery and development of memristors, a fourth fundamental circuit element alongside resistors, capacitors, and inductors. Memristors, short for memory resistors, exhibit unique properties that hold the potential to revolutionise the fields of memory storage and computing systems.

First theorised by Leon Chua in 1971 [76, 77], it wasn't until the early 2000s that mem-

ristors were experimentally realised. As Chua pointed out, the fundamental two-terminal circuit elements are historically three, and they are defined through a relationship between two of the four fundamental circuit variables (the current i, the voltage v, the charge q and the flux-linkage ϕ). In fact, we have the resistor (linking the voltage and the current), the inductor (linking the flux-linkage and the current), and the capacitor (linking the charge and the voltage). Starting from this, Chua noticed that a fourth basic relationship was missing: the relationship between the flux-linkage and the charge, being it the memristor. The first practical implementation of the memristor from the HP labs [78, 79] with the first *Titanium Dioxide* (TiO_2)-based memristor increased the interest in the technology, thanks to its theoretical switching speed, area occupation and non-volatility [80] (and therefore low energy consumption). The research continued and as of today the main metal-oxyde materials employed to manufacture memristive devices are TiO_2 [81, 82], VO_2 [83], NiO [84, 85], SiO_2 [86], Al_2O_3 [87] and Ta_2O_5 [88].

These nanoscale devices are characterised by their ability to change resistance based on the magnitude and direction of the current flowing through them, while retaining this resistance state even when the power is turned off. This inherent non-volatility sets memristors apart from traditional memory technologies such as DRAM or Flash memory, which require constant power supply to retain data. Applications of memristors can span through many areas, from circuits to neuromorphing computing. They can be used to enable CIM [3, 22, 89–92]: the memristor's physical property to change resistive value with current passing through can be exploited to enable operations inside the memory. The memristor-based CIM include Neuromorphing [35–37, 93–99] and Logic Computing [100].

A memristor (or a memristive device) [101] is a two-terminal device characterised by a resistance that varies depending on the amount and direction of the current flowing through it. More specifically, "a two-terminal black box is called a memristor if, and only if, it exhibits a *pinched histeresis loop* for *all bipolar periodic* input current signals (resp., input voltage signals) which result in a *periodic* voltage (resp., current) response of the same frequency, in the voltage-current ($v - i$) plane" [102]. This definition is in line with the original memristor definition and the original definition of memristive device [103].

The memristor has a minimum and maximum resistive value (LRS and HRS, respec-

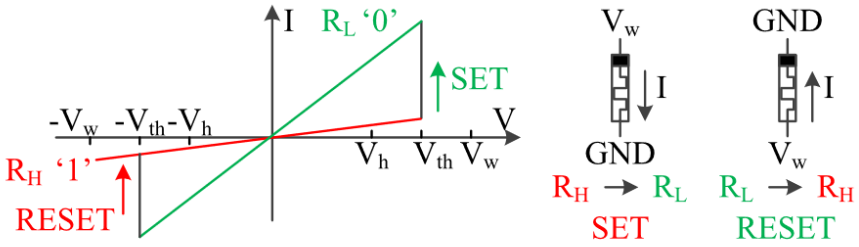


Figure 2.1: Memristor's symbol and convention. From [3].

tively). The write operation is done by applying electrical current to switch from one state to the other. The read operation is done comparing the current flowing through the memristor against a reference current.

Memristors combine non-volatility, low power consumption, high integration, fast write time, and a high ratio between HRS (R_{OFF}) and LRS (R_{ON}). These characteristics, along with LIM capabilities, make them very interesting candidates as Non-Volatile Memories. They are typically inserted in memory crossbars to be used as memory arrays. Resistive memory technologies can be split in three main categories: Electrostatic or Electronic Effect Memories, PCM, and Redox Memories. The first is based on electronics effect (charge injection and trapping), the second exploits the thermal effect to change the resistive state (by, for example, applying an electrical current that, by means of the Joule effect, will form or disrupt a conduction path in an insulating material). The Redox switches between state by exploiting reduction/oxidation (Redox) reactions. Due to the variety of materials used in their fabrication, memristors can have different electric behaviours. They can be controlled in voltage or current, and their resistance can exhibit a continuous transition (from LRS to HRS) or their resistance can remain unchanged until a certain threshold is reached. This last class of memristors is very useful for applications such as data storage and CIM because it allows for non-destructive operations (with voltages or currents below threshold).

In fact, the physical capabilities of the memristor can allow to perform internal computation in memory arrays [104–108]. The various types of computation enabled by these

devices are reported in the next chapter.

2.2 LIM Solutions

CIM, as show in the introduction (chapter 1), promises to solve major bottlenecks found in modern architectures. Memristive devices, such as RRAM, PCM and MRAM, are particularly well suited for CIM, especially for LIM (LIM is part of the CIM techniques and it is based on the execution of Boolean operations inside the memory).

In the traditional use of a memory array, a memory cell is selected by means of address decoders, it is written into by the WD and read from with the help of the SA. The voltage levels required to enable the operations on the memory cell are set by the voltage regulators. One memory array communicates with the processor or other memory arrays by means of bus connections. In order to enable the LIM operations, several changes need to be implemented to the memory array or/and to its peripheral circuitry. In this context, the peripheral circuitry consists of standard memory periphery (SA, WD, AD). Moreover, in some instances, additional logic is added to enable computation.

Several LIM proposals exist in literature: some are general, and can be used with any memory technology, others take advantage of the device physics and are only suitable to be implemented in a specific technology. In addition, some of the existing LIM solutions are designed to implement specific logic functions [27, 28, 31, 109–112] henceforth called "primitive operations", while others propose solutions for the implementation of any Boolean function [30]. The existing LIM solutions can be classified depending on the way the inputs are stored (the memory content, i.e., stateful logic, or an electrical signal, i.e., non-stateful logic) and depending on where the operations are performed (in the memory array, or in the periphery). In this context, three main LIM classes can be distinguished and their characteristics are summarised in table 2.1.

The first class, the Stateful logic in Array, is very interesting because it requires (almost) no modifications of the original memory architecture. The main changes are done in the controller where the compute operation has to be added, besides the read and write, to enable the LIM operations. In the implementations belonging to this class, the input(s) and output logic values are encoded as resistive states and the operations are performed

within the memory array. In the Stateful Logic in Array and its Periphery, the input(s) logic value(s) are encoded inside the memory and the operations are performed in the memory array and with modified logic inside the memory periphery (i.e., SA, WD and AD); in this case the result will be available at the sense amplifier, and, if needed, it will have to be written back to the memory. Finally, in the Non-Stateful Logic in Array and its Periphery, the logic values are encoded as resistive states and operations are performed by translating certain resistive states into voltage levels [113] – as shown in tables 2.1 and 2.2.

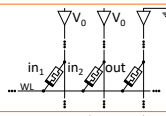
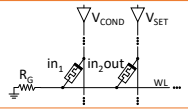
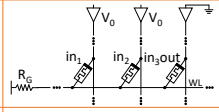
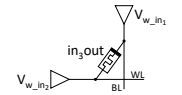
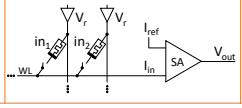
	LIM Solution	*#mem pts*	*Gate*	*Remarks*	*Technology*	*Operations*
LIM Array Stateful Logic	**MAGIC** (NOR2) **FELIX** (NAND2, OR2, XOR2)	3		input non-destructive, parallelizable	Memristive crossbar (1R-RRAM)	1. Initialize R_{out} 2. Apply proper voltages to R_{in}, R_{out} 3. Obtain output in R_{in_out}
	IMPLY $in_1 \rightarrow in_2 out$	2		input destructive, parallelizable		
	Stateful Three-Input Logic (ORNOR3)	3		input destructive		
LIM Array + Periphery – Non Stateful Logic (Hybrid inputs)	**PLiM** (RMAJ3)	3 (1 in the array, 2 external)		input destructive, requires input pre-processing	Memristive crossbar (1R-RRAM)	1. Read inputs R_{in} and convert to voltages 2. Apply the voltages to R_{in_out} 3. Obtain output in R_{in_out}
LIM Array + Periphery Stateful Logic	**Logic in Periphery**	2		input non-destructive, additional step to store output in memory	SRAM 1T1R-RRAM STT-MRAM	1. Apply voltage on multiple rows 2. Obtain output (via SA) 3. [Store output in memory]

Table 2.1: LIM Primitive logic gates. From [113]. Column 2 (CIM Solution) lists the LIM solutions considered and the corresponding primitive operations. The number of memory cells needed to implement a 2-input (1-bit) primitive operation is summarised in column 3 (#mem pts), while the schematic of the primitive operation gate for each solution is illustrated in column 4 (Gate). Column 7 (Operations) lists the algorithm executed to obtain the result of the primitive operation. The executed operation can be input-destructive or not (see column 5, Remarks). An input-destructive operation is an operation that changes the value of the inputs after it is executed.

LIM Array, Stateful Logic The operations are performed within the memory array and the data are coded as memory content. This solution is proposed only for memristive crossbars (1R-RRAM); The input data is stored within the memory array, and the output (computation result) is obtained as memory content within the memory array. In order to

	LIM Array Stateful Logic	LIM Array + Periphery Stateful Logic	LIM Array + Periphery Non Stateful Logic (Hybrid Inputs)
CIM Solution and Primary Function	MAGIC [27, 114] (NOT, NOR2) FELIX [109, 110] (NAND2, OR2, XOR2) IMPLY [28, 111, 116] Stateful three-input logic [112] (ORNOR3)	Any 2-input function	PLiM [31, 115] (RMAJ3) (AND2, OR2, IMPLY2, !IMPLY2)
Technology	Memristive crossbar (1R-RRAM)	SRAM 1T1R-RRAM STT-MRAM	Memristive crossbar (1R-RRAM)
Operations	1. Initialize R_{out} if needed 2. Apply proper voltages to R_{in}, R_{out} 3. Obtain output in R_{in_out} or R_{out}	1. Apply voltage on multiple rows 2. Obtain output (via SA) 3. [Store output in memory]	1. Read inputs R_{in} and convert to voltages 2. Apply the voltages to R_{in_out} 3. Obtain output in R_{in_out}

Table 2.2: LIM implementations overview.

enable LIM operations within the memory array, several conditions need to be respected: (1) the memory cells containing the input data and the memory cells to store the result of the computation must share the same row (column); (2) access to multiple memory cells should be enabled; (3) specific control voltages (different than the memory read/write voltages) to be applied for the completion of logic operations . As a consequence, the WD, the voltage regulator and the address decoders of standard memory array have to be modified to enable LIM. Stateful Logic in Array solutions include IMPLY [28, 111, 116, 117] having the Boolean implication as primitive operation, the Stateful Three-Input Logic [112] with the primitive operation ORNOR3 (i.e., input1 OR (input2 NOR input3)), Memristor-Aided Logic - MAGIC [27, 114], whose primitive operations are NOT and NOR, FELIX [109, 110] whose primitive operations are OR, NAND and XOR, and SIXOR [118], enabling the XOR operation. Some of the existing LIM solutions like IMPLY and ORNOR3 do not preserve the content of all the input memristors after executing the operation (also called input-destructive operation). On the other hand, MAGIC and FELIX promise non-input destructive operations, as classical computation paradigms, allowing therefore the re-use the input data for several operations (and, in some cases, to allow parallelism and save time).

LIM Array + Periphery, Stateful Logic The operations are performed within the memory array periphery or by means of additional logic and the input data are coded as memory content. This solution can be used with any technology (references). The input data

is stored within the memory array, while the output (computation result) is obtained as a voltage (or current) outside of the memory array. In order to enable LIM operations within the periphery of the memory array, several conditions must be respected: (1) the memory cells containing the input data must share the same column; (2) access to multiple memory cells must be enabled; (3) the sense amplifiers should be modified so that different references are allowed. As a consequence, the address decoders and the sense amplifiers of the standard memory array have to be modified to enable LIM. We refer to this class of solutions as Logic in Periphery. The MRIMA architecture [30] is based on the Logic In Periphery: it exploits reconfigurable SAs to perform arithmetic and logic operations on STT-MRAM. All Boolean functions can be implemented with this solution by resorting to additional combinational gates. Other implementations are [40, 119, 120].

LIM Array + Periphery, Non Stateful Logic (Hybrid inputs) The operations are performed within the memory array and by using additional logic, and the data are coded partially as memory content and partially as voltage levels. This solution can be used with resistive technology only. It uses two types of input data: (1) memory content, (2) voltage level, while the output (computation result) is obtained as memory content within the memory array. In order to enable this type of LIM operations, several conditions must be respected: (1) Specific control voltages (different from the memory read/write voltages) to be applied for the completion of logic operations, (2) specific registers to store the inputs to be given as voltage levels. As a consequence, the WD, the voltage regulator and the address decoders of standard memory array have to be modified to enable LIM. An implementation of this solution is PLiM [31], which implements, as primitive operation, a special case of majority voter, where one of the inputs is negated (a.k.a., *Resistive majority*).

The various solutions take inputs that can be resistive or voltages, and the executed operation can be input-destructive or not. An input-destructive operation is an operation that can change the value of the inputs after it is executed. This happens, for example, in the IMPLY logic, explained below, where the result of the operation overwrites one of the two inputs. The non-input-destructive operation makes use of an output memristor to avoid overwriting one of the inputs. This is the case of the basic function of MAGIC,

which makes use of three memristors, two inputs and one output. Depending on the chosen solution, we see an impact on the algorithms and area occupancy and in some cases input destructiveness can save area, while in other cases a copy operation might be needed, increasing the latency. In the following, the introduced classifications are treated. The following syntax will be used to show the number of steps (*i-j*) and the number of memristors used for each operation:

LIM_Boolean - # of steps, # of memristors	
i – j)	LIM_Boolean (used memory cells)

2.2.1 LIM Array Stateful Logic

The LIM Array Stateful logic [121], as previously introduced, makes use of resistive inputs, and it has a resistive output. The operations are enabled connecting together two or more memristors, with specific voltages that permit to change the memristive state of the output memristor in some specific cases. The solutions that we have taken into account are MAGIC, IMPLY and their variations.

MAGIC

The intrinsic logic functions implemented by MAGIC - Memristor-Aided Logic [27, 114] are the NOT and the NOR. Being part of the LIM paradigm, they allow to have the operations directly in the memory array, with the result of the operation finally written onto the output memristor. The NOT operation is executed on 2 memristors, where one is the input (in_1) and the other the output (out). The NOR2 operation is executed on 3 memristors, where 2 of them are the input values (in_1 and in_2) and the third is where the output will be written (out). MAGIC makes use of strict voltage constraints to achieve non-input-destructive operations.

The MAGIC NOR and MAGIC NOT operations consist of two steps: the output memristor initialisation and the execution of the logic operation. The logic operation (either NOR or NOT) is carried out by first setting the memristor to R_{ON} (LRS), hence initialising the memristor wit a SET operation, then providing a voltage V_0 (step 2). The output memristor will switch to HRS if its voltage drop is large enough. The value of V_0 for which

the two operations are performed correctly is computed as in equation (2.1) for the NOT operation and in equation (2.2) for the NOR operation.

$$2V_{T,OFF} < V_0 < \frac{R_{HRS}}{R_{LRS}} \cdot min\left[V_{T,OFF}, \left|V_{T,ON}\right|\right] \tag{2.1}$$

$$2V_{T,OFF} < V_0 < min\left[\frac{R_{HRS}}{2R_{LRS}} V_{T,OFF}, \left|V_{T,ON}\right|\right] \tag{2.2}$$

where $V_{T,OFF}$ and $V_{T,ON}$ are the memristive threshold voltages for switching from LRS to HRS, and from HRS to LRS respectively.

The basic MAGIC NOR configuration is shown in table 2.1.

MAGIC NOT (in_1, out) – 2 steps, 2 memristors	
1)	Write1 (out)
2)	MAGIC NOT (in_1, out)

Table 2.3: MAGIC NOT basic steps. in_1 represents the input memristor, while out represents the output memristor. `Write1` represents the SET operation (that allows to have the output memristor at LRS). `MAGIC NOT` represents the application of the control voltage V_0 to the BL of the input memristor.

MAGIC NOR (in_1, in_2, out) – 2 steps, 3 memristors	
1)	Write1 (out)
2)	MAGIC NOR (in_1, in_2, out)

Table 2.4: MAGIC NOR basic steps. in_1 and in_2 represent the input memristors, while out represents the output memristor. `Write1` represents the SET operation (that allows to have the output memristor at LRS). `MAGIC NOR` represents the application of the control voltage V_0 to the BLs of the input memristors.

More in detail, the output state (set at logic 1), can only change to logic 0 if there is enough voltage applied on it to generate a current flowing towards the ground. The only case when there is not enough voltage on the output memristor happens by having both the input transistors' states at logic 0 (R_{OFF}, HRS), thus allowing the NOR operation execution. In fact, having just as one input memristor with LRS (logic 0, R_{ON}) would allow enough current flowing in the output memristor to change its memristive value

and therefore cover the cases 0-0, 0-1 and 1-0. The case of the NOT gate exploits a voltage divider, and the output state will change from logic 1 to 0 only if the input is a logic 1. There are other possible MAGIC gates (OR, AND, and NAND), but these can't be implemented in a crossbar and hence they are less interesting for our intended LIM applications.

Being the NOR a functionally complete Boolean operator, it can be used to express all the logical operations, as we did in the next section, with the addition of the NOT logic gates that speeds up the operations.

MAGIC has 2 interesting variations, FELIX and X-MAGIC. FELIX [110] adds some basic logic primitive operations on top of the MAGIC ones, while X-MAGIC [109] enables input-destructive operations: it allows to avoid the output memristor initialisation. As an example, the NAND function takes 11 steps for MAGIC, and 2 for FELIX and X-MAGIC. The speed-up of X-MAGIC with respect to FELIX can be seen for more complex operations. The state on the art has many MAGIC-based implementations and studies, such as [122–131]. The analysed functions, together with the number of steps taken and the number of memory elements used, are shown in table 2.11.

FELIX

FELIX [110] is an extension of MAGIC: it adds primitive operations on top of the MAGIC NOR, introducing NAND, OR and XOR (as concatenation of OR and NAND). Tables 2.5 to 2.7 summarise the basic steps of these operations. The FELIX NAND operation exploits the same principle of MAGIC, but with a lower control voltage, which prevents the output from switching when only one input is at LRS. In this case, V_0 can be calculated as (where || represents the parallel resistance configuration):

$$\frac{V_{T,OFF}}{R_{LRS}}\left\{R_{LRS}+\left(R_{HRS}\|\frac{R_{LRS}}{3}\right)\right\} < V_0 < \frac{V_{T,OFF}}{R_{LRS}}\left\{R_{LRS}+\left(R_{HRS}\|\frac{R_{LRS}}{2}\right)\right\} \tag{2.3}$$

To perform a FELIX OR operation, the output memristor is first reset to R_{OFF} (HRS), and then V_0 is applied. The equation defining the control voltage amplitude is:

$$\frac{\left|V_{T,ON}\right|}{R_{HRS}}\left\{R_{HRS}+\left(R_{LRS}\|R_{HRS}\right)\right\} < V_0 < \frac{\left|V_{T,ON}\right|}{R_H RS}\frac{3}{2}R_{HRS} \tag{2.4}$$

The FELIX XOR operation is a three-cycle operation composed of a FELIX OR operation followed by a FELIX NAND operation (without the output initialisation).

FELIX NAND (in_1, in_2, out) – 2 steps, 3 memristors	
1)	Write1 (out)
2)	FELIX NAND (in_1, in_2, out)

Table 2.5: FELIX NAND basic steps. in_1 and in_2 represent the input memristors, while out represents the output memristor. `Write1` represents the SET operation (that allows to have the output memristor at LRS). `FELIX NAND` represents the application of the control voltage V_0 (under the FELIX NAND constraints, as described in equation (2.3)) to the BLs of the input memristors.

FELIX OR (in_1, in_2, out) – 2 steps, 3 memristors	
1)	Write0 (out)
2)	FELIX OR (in_1, in_2, out)

Table 2.6: FELIX OR basic steps. in_1 and in_2 represent the input memristors, while out represents the output memristor. `Write0` represents the RESET operation (that allows to have the output memristor at HRS). `FELIX OR` represents the application of the control voltage V_0 (under the FELIX OR constraints, as described in equation (2.4)) to the BL of the output memristor.

FELIX XOR (in_1, in_2, out) – 3 steps, 3 memristors	
1 – 2)	FELIX OR (in_1, in_2, out)
3)	FELIX NAND V_0 (in_1, in_2, out)

Table 2.7: FELIX XOR basic steps. in_1 and in_2 represent the input memristors, while out represents the output memristor. `FELIX OR` represents the whole FELIX OR operation table 2.6. `FELIX NAND` V_0 represents the application of the NAND control voltage (equation (2.3)) to the BLs of the input memristors, while the BL of the output memristor is connected to ground. The difference between a full FELIX OR followed by a full FELIX NAND and the FELIX XOR steps is the missing initialisation of the output memristor before the FELIX NAND and it is what enables the XOR operation.

IMPLY

The IMPLY [28, 111, 116, 117] solution enables the in-memory primitive of the implication logic (IMPLY), $in_1 \longrightarrow in_2$, which is equivalent to $\neg in_1 \vee in_2$. Its inputs are, as in MAGIC, represented by the memristors' states. The operation is only executed on 2 memristors (the inputs), and the output is stored by overwriting one of the inputs, i.e., it is an input-

destructive logic. The IMPLY operation needs a resistor R_g in series with the gate, to enable the proper functioning of the logic.

IMPLY (in_1, in_2out) – 1 step, 2 memristors
1) IMPLY (in_1, in_2out)

Table 2.8: IMPLY basic steps. in_1 represents the input memristor, while in_2out represents the other input memristor that will be overwritten with the result of the operation (being therefore also the output memristor). `IMPLY` represents the simultaneous application of the control voltages V_{COND} and V_{SET} on, respectively, in_1, and in_2out.

The input/output memristor can only switch from logic 0 to logic 1, and this can happen (having both in_1 and in_2out at logic 0). Switching from logic 1 to logic 0 is not possible. Therefore, to have a logically complete behaviour, the NOT (or FALSE) operation has to be implemented.

The basic IMPLY configuration can be seen in table 2.1. To make the device switch from the low resistive state (logic 1) to the high resistive state (logic 0), a positive voltage higher than the threshold V_{ON} has to be applied. In the same way, applying a negative voltage lower than V_{OFF} will perform the opposite operation. To SET a device, i.e., write a value of logic 1 (TRUE operation), a negative voltage V_{SET}, has to be applied. It must be larger than V_{ON}, to compensate the voltage drop across R_G. To clear a device (FALSE operation), a positive voltage V_{OFF} has to be applied. Finally, to execute the operation, a negative voltage V_{COND} is used, that has a magnitude smaller than V_{SET}, not able to change the state of the devices that is driving. The voltages V_{COND} and V_{SET} are applied simultaneously on, respectively, in_1 and in_2out. The implementation in a memory array is achieved with memristors, using RRAM. It is an input-destructive logic. The typical implementation of the IMPLY logic gate is made through a memristor-based crossbar. To enable this, a resistance must be added for each crossbar row. In the state of the art are present many studies on IMPLY, such as [131–137] To guarantee the correct execution of the IMPLY operation, the V_{SET}, V_{COND} and R_G and computed as follows:

$$R_{ON}\frac{V_{SET}-V_{ON}}{V_{ON}-[V_{SET}-V_{COND}]} < R_G < R_{OFF}\frac{V_{SET}-V_{ON}}{2V_{ON}-[V_{SET}-V_{COND}]} \tag{2.5}$$

$$\frac{V_{SET}}{V_{COND}} < \frac{R_{OFF}}{R_{ON}} \tag{2.6}$$

There is another implementation based on the IMPLY logic, that uses three memristive switches instead of two, and is more efficient with more complex operation: it is the Stateful Three-Input Logic with Memristive Switches [112], that enables the primary function ORNOR3, $in_1 out' = in_1 out \vee \neg(in_2 \vee in_3)$. The three memristors are inputs, and one of them also acts as output. Also for this operation a resistance R_G in series is necessary to ensure the functioning of the logic. Another interesting application of IMPLY is shown in [138]

ORNOR3 ($in_1 out$, in_2, in_3) – 1 step, 3 memristors
1) ORNOR3 ($in_1 out$, in_2, in_3)

Table 2.9: ORNOR3 basic steps. $in_1 out$ is the input memristor that will be overwritten with the result of the operation. in_2 and in_3 represent the other 2 input memristors. ORNOR3 represents the application of the control voltages V_{SET} on the $in_1 out$ memristor and V_{COND} on the other two memristors.

Its circuit is shown in table 2.1. The ORNOR3 primitive logic operation supports the NOR2 operation fixing an input memristor, as well the IMPLY2 operation (performing the operation on only two inputs instead of three). The differences can be seen in table 2.11.

2.2.2 LIM Array + Periphery, Stateful Logic

The LIM Array + Periphery, Stateful logic - also named Scouting Logic [32] - uses as inputs the memory cells, while the output is made available at the end of the sensing. The Logic in Periphery enables the computation outside of the memory array by modifying the SA and adding combinational logic. Its primitive operation is achieved by modifying the way how the memory is read, adding reference signals to be compared in order to execute the operations. The modification is therefore performed on the SAs, and CMOS logic can be added to enables more operations in the periphery. Since this technique affects the periphery, the memory array can be of every memory technology previously shown. Moreover, the logic operations are executed by reading the memory, without changing its state. This has the consequence of reducing the stress on the memory, especially in the case of the memristors, considering that these devices still suffer from endurance problems. Therefore, moving the operations from the memristor to the SAs can improve the lifetime of the device, still providing CIM capabilities. The operations that can be performed with

the Logic in Periphery are OR, AND and XOR. Every operation performed is limited to a single memory read operation. Then the current or the voltage drop are compared with a reference signal. The XOR makes use of a NOR between the result of the OR and the AND. To perform the logic operations, two different types of SAs can be implemented, current or voltage based. The logic operations are executed by reading two memory cells at a time: in the resistance-based memories, the current going to the SA will therefore depend on the parallel between the resistances of the read cells in_1 and in_2. The resulting value of $in_1 \| in_2$ can have three values: $R_{ON}/2$, $R_{OFF}/2$ and $R_{ON} \| R_{OFF} \simeq R_{ON}$. Depending on the resistance value, the current (or voltage) sensed varies, and the logic operations can be performed by properly setting the reference current(s), or voltage(s), depending on the chosen sense amplifiers.

The MRIMA architecture [30] is based on the Logic In Periphery: it exploits re-configurable SAs to perform arithmetic and logic operations on STT-MRAM. It allows to perform:

- Fast Row Copy,
- NOT,
- Two-input memory logic - AND2/NAND2, OR2/NOR2, XOR2/XNOR2,
- Three-input memory logic - AND3/NAND3, OR3/NOR3, MAJ/MIN.

The MRIMA accelerator can be used for CNN and data encryption applications: it has an in-memory bit-wise adder and an in-memory bit-wise convolver. To perform the addition, the MRIMA authors propose a parallel in-memory adder: it exploits the 2- and 3-input operations to be faster. In fact, C_{OUT} can be calculated via the majority function, and it can be temporarily stored in a carry latch. At the same time, the Sum can be calculated via a XOR gate. The one-bit addition is therefore performed in 2 cycles, each one composed of 4 steps (S1, S2, C1 and C2). The basic idea of MRIMA is to have an independent high-performance and energy-efficient accelerator. From the programmer point of view, it can be seen as a third party accelerator. Therefore, to be integrated in the system, MRIMA needs system-level libraries, and a Virtual Machine together with an Instruction Set Architecture (ISA) to be used.

Other LIM in array and periphery that work with a similar concept are [33, 40–47, 50, 52]

2.2.3 LIM Array + Periphery, Non Stateful Logic

The intrinsic logic function that PLiM [31, 115] implements is the Resistive Majority. This is the majority voter, with one of the inputs that is negated.

$$RMAJ = MAJ(in_1, \neg in_2, in_3)$$

This operation has hybrid inputs: it is executed on the memristor $in_1 out$ that acts as input and output and the two other inputs are the two voltages V_{in_1} and V_{in_2}. The operation is performed applying these voltages to the top and bottom electrodes of the memristor. This implementation also takes into account the memory controller, a lightweight unit managing the operations performed on the memory array. Possible primary functions that can be performed are the AND2 and OR2, achieved by fixing one input voltage. The primitive operations can be found in table 2.11.

2.3 State of the art solutions and comparison

In recent years, the number of research papers dealing with memristive LIM on different abstraction levels has significantly increased. In this context, research is mainly focused on the design of LIM architectures, the development of LIM-compatible instruction sets, the methods for system integration, and development of the programming model for LIM integration in computing systems. Nevertheless, the actual status of the research is fragmented and the reproduction of the reported results, along with the choice of an implementation to be adopted, are not trivial. For instance, [22] presents a review on in-Memory Computing, focusing on the memories enabling it and on its applications, [90] offers a classification of CIM solutions, while [122] describes several adder implementations. In addition to fragmentation, the use of existing electrical models and their parameters is not always supported by physical measurements on real devices. While these comparative surveys show the characteristics of existing solutions, a thorough analysis of the implementations of Boolean functions is still missing. Therefore, in this chapter is presented a

study of the existing LIM implementations in order to perform a fair comparison in terms of resources and efficiency. More in particular, a thorough study of simple Boolean functions implemented in memory is shown, resulting in a comprehensive comparison of LIM solutions in terms of required number of memristors and number of operations.

The purpose of this preliminary study is to provide a comprehensive comparison of existing LIM solutions and understand their implementation complexity. The analysis has been performed on basic Boolean functions, in order to be as generic as possible and to provide the designer an indication of the implementation complexity and cost of each LIM solution. In addition, this study can give an indication of which LIM solution is more suitable for a target application, depending on the most used Boolean functions.

In order to achieve a fair comparison among all solutions, we mapped all the 0-input logic functions (TRUE and FALSE), 1-input logic functions (COPY, NOT), 2-input logic functions (NOR, OR, NAND, AND, XNOR, XOR, NIMPLY, IMPLY) and the Full Adder as 3-input logic function, by using the primitive operations of MAGIC (and its extensions), IMPLY (and ORNOR3) and PLiM solutions. The full adder has as inputs A, B and C_{IN} while the outputs are S and C_{OUT}.

$$C_{OUT} = ((A+B)' + (B+C)' + (C+A)')' \tag{2.7}$$

$$S = (((A' + B' + C')' + ((A+B+C)' + C_{OUT})')')' \tag{2.8}$$

Tables 2.10 and 2.11 summarise the results of this study. Table 2.11 shows the mapping of all considered Boolean functions on LIM primitive operations. For each LIM implementation, the primitive operations are written in blue. Each row contains the mapping of the Boolean function defined in the first column

For clarity, the same unique syntax used in section 2.2 is also used for all cells of table 2.11:

LIM_Boolean (used memory cells)
$i - j$) LIM_Boolean (used memory cells)

where the first line defines the name of the Boolean function implemented in a specific LIM, together with the used memory cells for inputs and outputs; the following lines

describe the algorithm used to map that function on primitive operations, underlying the number of steps required for its execution (i-j). In the case of a majority voter requiring additional registers, the algorithm contains additional read operations marked as r_i. For each function, a number of memristors are used to store the input (in_i), the output (out), and intermediate results (f_i). The intermediate results are used to solve complex mapping algorithm where several operations are executed in sequence, and they are stored in so-called functional memristors. In case of destructive operations (i.e., IMPLY), the content of one of the input memristors is overwritten by the output (noted as $in_i out$ in the table).

	Input			#memristors						#steps					
2 bit	0 0 1 1 0 1 0 1														
Gate	Output	#in	#out	MAGIC	FELIX	IMPLY	IMPLY destr.	ORNOR3	MAJ3	MAGIC	FELIX	IMPLY	IMPLY destr.	ORNOR3	MAJ3
TRUE (write 1)	1 1 1 1	0	1	1 + 0	1 + 0	1 + 0	1 + 0	1 + 0	1 + 0	1	1	1	1	1	1
FALSE (write 0)	0 0 0 0	0	1	1 + 0	1 + 0	1 + 0	1 + 0	1 + 0	1 + 0	1	1	1	1	1	1
in1 (COPY)	0 0 1 1	1	1	2 + 1	2 + 1	2 + 1	2 + 1	2 + 1	2 + 0	4	3	4	4	4	2 + 1
NOT *in1*	1 1 0 0	1	1	2 + 0	2 + 0	2 + 0	2 + 0	2 + 0	2 + 0	2	2	2	2	2	2 + 1
in1 NOR *in2*	1 0 0 0	2	1	3 + 0	3 + 0	3 + 1	3 + 0	3 + 0	3 + 1	2	2	9	5	2	6 + 4
in1 OR *in2*	0 1 1 1	2	1	3 + 1	3 + 0	3 + 1	2 + 1	3 + 1	3 + 1	4	2	7	3	4	4 + 3
in1 NAND *in2*	1 1 1 0	2	1	3 + 2	3 + 0	3 + 0	3 + 0	3 + 0	3 + 1	8	2	3	3	3	6 + 5
in1 AND *in2*	0 0 0 1	2	1	3 + 2	3 + 1	3 + 1	3 + 1	3 + 1	3 + 1	6	4	5	5	5	4 + 3
in2 IMP *in1*	1 0 1 1	2	1	3 + 1	3 + 1	3 + 1	2 + 0	2 + 0	3 + 0	6	4	5	1	1	2 + 2
in2 NIMP *in1*	0 1 0 0	2	1	3 + 1	3 + 1	3 + 1	3 + 0	3 + 0	3 + 1	4	4	7	3	3	4 + 3
in1 XOR *in2*	0 1 1 0	2	1	3 + 2	3 + 0	3 + 2	3 + 2	3 + 2	3 + 1	10	3	13	13	10	7 + 4
in1 EQUAL *in2* (XNOR)	1 0 0 1	2	1	3 + 2	3 + 1	3 + 2	3 + 2	3 + 2	3 + 1	12	5	15	15	12	9 + 5
	Input														
3 bit	0 0 0 0 1 1 1 1 0 0 1 1 0 0 1 1 0 1 0 1 0 1 0 1														
	Output														
FA (sum)	0 1 1 0 1 0 0 1	3	2	5 + 4	5 + 1	5 + 2	5 + 2	5+5	5 + 1	36	12	49+37	49+37	24	9 + 8
FA (c_out)	0 0 0 1 0 1 1 1														

Table 2.10: Number of memristors and number of operations per Boolean function.

To validate the solutions, we have developed a script that checks the correctness of each Boolean function mapped on LIM primitive operations. Table 2.10 summarises, for each Boolean function:

- its truth table (column 2),
- number of inputs and outputs of the function (columns 3 and 4),
- for each LIM solution: number of used memristors, in the form #(input and output) + #functional (columns from 5 to 10),
- for each LIM solution: number of operations needed to perform the computation (columns from 11 to 16). For the PLiM implementation the steps are indicated as the memory cycles + the reading operations.

This extensive study has the goal of analysing the most prominent LIM solutions and providing a comparison in terms of required memory resources (i.e., number of memristors) and number of operations to implement basic Boolean functions. The results obtained show large discrepancies among the LIM solutions in the number of steps required to perform the operations. For instance, the XOR requires many more steps if implemented with IMPLY logic compared to FELIX logic. This had given us some hints on which could be the lightest and fastest solution, but it was just a first exploration. In fact, it is important to remark that these results reflect the complexity of each operation but do not directly translate into an estimation of the actual execution time. This is due to the fact that, due to the physical and electrical characteristics of the memristive devices, the timing of each operation can vary significantly. This analysis gave us a first feedback on how the studied implementations work (and also gave us the configuration that we could use in the future to execute different operations). Indeed, this preliminary analysis has been followed by a more in-depth analysis, as discussed in the next chapters.

	LIM Array				*LIM Array + Periphery*
	Stateful Logic				*Non Stateful Logic (Hybrid Inputs)*
	MAGIC-based		**IMPLY-based**		**PLIM**
	MAGIC	**FELIX**	**IMPLY**	**IMPLY (non-destructive)**	**ORNOR3**	**RMAJ**

Primitives	`MAGIC_NOT(in1,out)`
	`MAGIC_NOR(in1,in2,out)`
		`FELIX_OR(in1,in2,out)`
		`FELIX_NAND(in1,in2,out)`
		`FELIX_MIN(in1,in2,in3out)`
			`IMPLY_IMPLY(in1,in2out):`		`ORNOR3(in1out,in2,in3)`	`RMAJ(in1,in2,in3out)`

COPY	`MAGIC_COPY(in1,f1,out):` 1-2) MAGIC_NOT(in1,f1) 3-4) MAGIC_NOT(f1,out)	`FELIX_COPY(in1,f1,out):` 1) Write0(f1) 2-3) FELIX_OR(in1,f1,out)	`IMPLY_COPY(in1,f1,out):` 1-2) IMPLY_NOT(in1,f1) 3-4) IMPLY_NOT(f1,out)			`RMAJ_COPY(in1,out):` 1) Write0(out) r1) Read(in1) 2) RMAJ(in1,0,out):
NOT	`MAGIC_NOT(in1,out):` 1) Write1(out) 2) MAGIC(in1,out)		`IMPLY_NOT(in,out):` 1) Write0(out) 2) IMPLY(in,out)			`RMAJ_NOT(in1,out):` 1) Write0(out) r1) Read(in1) 2) RMAJ(1,in1,out)
NOR	`MAGIC_NOR(in1,in2,out):` 1) Write1(out) 2) MAGIC(in1,in2,out)		`IMPLY_NOR(in1,in2,out):` 1-3) IMPLY_OR(in1,out,in2) 4-5) IMPLY_NOT(in2,out)	`IMPLY_NOR_ND(in1,in2,f1,out):` 1-7) IMPLY_OR_ND(in1,in2,out,f1) 8-9) IMPLY_NOT(f1,out)	`ORNOR3_NOR(in1,in2,out):` 1) Write0(out) 2) ORNOR3(out,in1,in2)	`RMAJ_NOR(in1,in2,f1,out):` 1-4,r3) RMAJ_OR(in1,in2,out,f1) 5-6,r4) RMAJ_NOT(f1,out)
OR	`MAGIC_OR(in1,in2,f1,out):` 1-2) MAGIC_NOR(in1,in2,f1) 3-4) MAGIC_NOT(f1,out)	`FELIX_OR(in1,in2,out):` 1) Write0(out) 2) FELIX(in1,in2,out)	`IMPLY_OR(in1,f1,in2out):` 1-2) IMPLY_NOT(in1,f1) 3) IMPLY_IMP(f1,in2out)	`IMPLY_OR_ND(in1,in2,f1,out):` 1-4) IMPLY_COPY(in2,f1,out) 5-6) IMPLY_NOT(in1,f1) 7) IMPLY(f1,out)	`ORNOR3_OR_ND(in1,in2,f1,out):` 1-2) ORNOR3_NOR(in1,in2,f1) 3-4) IMPLY_NOT(f1,out)	`RMAJ_OR(in1,in2,f1,out):` 1) Write1(out) 2-3,r1) RMAJ_NOT(in2,f1) r2-r3) Read(in1,f1) 4) RMAJ(in1,f1,out)
NAND	`MAGIC_NAND(in1,in2,f1,f2,out):` 1-6) MAGIC_AND(in1,in2,out,f1,f2) 7-8) MAGIC_NOT(f2,out)	`FELIX_NAND(in1,in2,out):` 1) Write1(out) 2) FELIX(in1,in2,out)	`IMPLY_NAND(in1,in2,out):` 1-2) IMPLY_NOT(in2,out) 3) IMPLY(in1,out)			`RMAJ_NAND(in1,in2,f1,out):` 1-4,r4) RMAJ_AND(in1,in2,out,f1) 5-6,r5) RMAJ_NOT(f1,out)
AND	`MAGIC_AND(in1,in2,f1,f2,out):` 1-2) MAGIC_NOT(in1,f1) 3-4) MAGIC_NOT(in2,f2) 5-6) MAGIC_NOR(f1,f2,out)	`FELIX_AND(in1,in2,f1,out):` 1-2) FELIX_NAND(in1,in2,f1) 3-4) MAGIC_NOT(f1,out)	`IMPLY_AND(in1,in2,f1,out):` 1-3) IMPLY_NAND(in1,in2,f1) 4-5) IMPLY_NOT(f1,out)			`RMAJ_AND(in1,in2,f1,out):` 1) Write0(out) 2-3,r1) RMAJ_NOT(in2,f1) r2-r3) Read(in1,f1) 4) RMAJ(in1,f1,out)
IMP	`MAGIC_IMPLY(in1,in2,f1,out):` 1-2) MAGIC_NOT(in1,out) 3-6) MAGIC_OR(out,in2,f1,out)	`FELIX_IMPLY(in1,in2,f1,out):` 1-2) MAGIC_NOT(in1,f1) 3-4) FELIX_OR(f1,in2,out)	`IMPLY_IMPLY(in1,in2out):` 1) IMPLY(in1,in2out)	`IMPLY_IMPLY_ND(in1,in2,f1,out):` 1-4) IMPLY_COPY(in2,f1,out) 5) IMPLY(in1,out)		`RMAJ_IMPLY(in1,in2,out):` 1) Write1(out) r1-r2) Read(in1,in2) 2) RMAJ(in2,in1,out)
NIMP	`MAGIC_NIMPLY(in1,in2,f1,out):` 1-2) MAGIC_NOT(in1,f1) 3-4) MAGIC_NOR(f1,in2,out)		`IMPLY_NIMPLY(in1,in2,out):` 1) IMPLY(in1,in2) 2-3) IMPLY_NOT(in2,out)	`IMPLY_NIMPLY_ND(in1,in2,f1,out):` 1-5) IMPLY_ND(in1,in2,out,f1) 6-7) IMPLY_NOT(f1,out)		`RMAJ_NIMPLY(in1,in2,f1,out):` 1-2,r2) RMAJ_IMPLY(in1,in2,f1) 3-4,r3) RMAJ_NOT(f1,out)
XOR	`MAGIC_XOR(in1,in2,f1,f2,out):` 1-2) MAGIC_NOT(in1,f1) 3-4) MAGIC_NOT(in2,out) 5-6) MAGIC_NOR(f1,out,f2) 7-8) MAGIC_NOR(in1,in2,f1) 9-10) MAGIC_NOR(f1,f2,out)	`FELIX_XOR(in1,in2,out):` 1-2) FELIX_OR(in1,in2,out) 3) FELIX_NAND(in1,in2,out)	`IMPLY_XOR(in1,in2,f1,f2,out):` 1-4) IMPLY_COPY(in1,f2,f1) 5) IMPLY(in2,f1) 6-7) IMPLY_NOT(f1,out) 8-11) IMPLY_COPY(in2,f1,f2) 12) IMPLY(in1,f2) 13) IMPLY(f2,out)		`ORNOR3_XOR(in1,in2,f1,f2,out):` 1-2) IMPLY_NOT(in1,f1) 3-4) IMPLY_NOT(in2,out) 5-6) ORNOR3_NOR(f1,out,f2) 7-8) ORNOR3_NOR(in1,in2,f1) 9-10) ORNOR3_NOR(f1,f2,out)	`RMAJ_XOR(in1,in2,f1,out):` 1-2,r1) RMAJ_COPY(in1,f1) r2) Read(in2) 3) RMAJ(0,in2,f1) 4-5,r2) RMAJ_COPY(in2,out) r3) Read(in1) 6) RMAJ(0,in1,out) r4) Read(f1) 7) RMAJ(f1,0,out)
XNOR	`MAGIC_XNOR(in1,in2,f1,f2,out):` 1-10) MAGIC_XOR(in1,in2,out,f1,f2) 11-12) MAGIC_NOT(f2,out)	`FELIX_XNOR(in1,in2,f1,out):` 1-3) FELIX_XOR(in1,in2,f1) 4-5) MAGIC_NOT(f1,out)	`IMPLY_XNOR(in1,in2,f1,f2,out):` 1-13) IMPLY_XOR(in1,in2,out,f1,f2) 14-15) IMPLY_NOT(f2,out)		`ORNOR3_XNOR(in1,in2,f1,f2,out):` 1-10) ORNOR3_XOR(in1,in2,out,f1,f2) 11-12) IMPLY_NOT(f2,out)	`RMAJ_XNOR(in1,in2,f1,out):` 1-7,r4) RMAJ_XOR(in1,in2,out,f1) 8-9,r5) RMAJ_NOT(f1,out)

Full Adder	`MAGIC_FA(in1,in2,cin,f1,f2,f3,f4,s,cout):`
	1-2) MAGIC_NOR(in1,in2,f1)
	3-4) MAGIC_NOR(in2,cin,s)
	5-6) MAGIC_NOR(f1,s,cout)
	7-8) MAGIC_NOT(cout,f1)
	9-10) MAGIC_NOR(cin,in1,s)
	11-12) MAGIC_NOR(f1,s,cout)
	13-14) MAGIC_NOR(in1,in2,f1)
	15-16) MAGIC_NOT(f1,s)
	17-18) MAGIC_NOR(s,cin,f1)
	19-20) MAGIC_NOR(f1,cout,f2)
	21-22) MAGIC_NOT(in1,f1)
	23-24) MAGIC_NOT(in2,f3)
	25-26) MAGIC_NOR(f1,f3,s)
	27-28) MAGIC_NOT(s,f1)
	29-30) MAGIC_NOR(cin,f3)
	31-32) MAGIC_NOR(f1,f3,f4)
	33-34) MAGIC_NOR(f2,f4,f1)
	35-36) MAGIC_NOT(f1,s)
		`FELIX_FA(in1,in2,cin,f1,s,cout):`
		1-3) FELIX_XOR(in1,in2,f1)
		4-6) FELIX_XOR(f1,cin,s)
		7-9) FELIX_COPY(cin,cout,f1)
		10) FELIX_MIN(in2,in2,f1)
		11-12) MAGIC_NOT(f1,cout)
			`IMPLY_FA_SUM(in1,in2,cin,f1,f2,out):`
			1-2) Write0(f2), Write1(out)
			3-11) IMPLY_OR(in1,f1,f2), IMPLY_OR(in2,f1,f2), OR(cin,f1,f2)
			12-16) IMPLY_AND(f2,out,f1,out)
			17) Write0(f2)
			18-22) IMPLY_OR(in1,f1,f2), IMPLY(in2,f2), IMPLY(cin,f2)
			23-27) IMPLY_AND(f2,out,f1,out)
			28) Write0(f2)
			29-33) IMPLY(in1,f2), IMPLY(in2,f2), IMPLY_OR(cin,f1,f2)
			34-38) IMPLY_AND(f2,out,f1,out)
			39) Write0(f2)
			40-44) IMPLY(in1,f2), IMPLY(cin,f2), IMPLY_OR(in2,f1,f2)
			45-49) IMPLY_AND(f2,out,f1,out)
			`IMPLY_FA_COUT(in1,in2,cin,f1,f2,cout):`
			1-2) Write0(f2), Write1(cout)
			3-8) IMPLY_OR(in1,f1,f2), IMPLY_OR(in2,f1,f2)
			9-13) IMPLY_AND(f2,cout,f1,cout)
			14) Write0(f2)
			15-20) IMPLY_OR(in1,f1,f2), IMPLY_OR(cin,f1,f2)
			21-25) IMPLY_AND(f2,cout,f1,cout)
			26) Write0(f2)
			27-32) IMPLY_OR(in2,f1,f2), IMPLY_OR(cin,f1,f2)
			33-37) IMPLY_AND(f2,cout,f1,cout)
					`ORNOR3_FA(in1,in2,f1,f2,f3,f4,f5,sum,cin,cout):`
					1-7) Write0(sum,cout,f1,f2,f3,f4,f5)
					8) IMPLY(in1,f1)
					9) IMPLY(in2,f2)
					10) ORNOR3(cout,f1,f2)
					11-12) Write0(f1,f2)
					13) ORNOR3(sum,in1,in2)
					14) ORNOR3(f1,sum,cout)
					15) IMPLY(cin,f2)
					16) ORNOR3(cout,f2,sum)
					17) Write0(sum)
					18) IMPLY(f2,f3)
					19) IMPLY(f1,f4)
					20) ORNOR3(f5,f3,f2)
					21-22) Write0(f1,f3)
					23) ORNOR3(f1,f2,f4)
					24) ORNOR3(sum,f5,f1)
						`RMAJ_FA(in1,in2,cin,f1,sum,cout):`
						1-2,r1) RMAJ_COPY(cin,cout)
						3-4,r2) NOT(in2,f1)
						r3,r4) Read(in1,f1)
						5)- RMAJ(in1,f1,in3out)
						6-7,r5) RMAJ_COPY(cin,sum)
						r6,r7) Read(in1,in2)
						8) RMAJ(in1,in2,sum)
						r8) Read(cin)
						9) RMAJ(in2,cin,sum)

Table 2.11 – Mapping of all considered Boolean functions on LIM primitive operations.

3 Simulation Environment

Today, there is a trend towards sharing information and enabling reproducible science. Unfortunately, when it comes to electrical simulations, the use of proprietary Computer-Aided Design (CAD) tools (which differ from provider to provider) and foundry-licensed device model libraries makes reproducible science almost impossible. This is mainly due to the fact that even having the full description of an electronic circuit, performing parametric or statistical electrical simulations requires CAD tool-specific configurations, which, if not set exactly, might lead to different results than originally reported. This can hinder the repeatability of experiments, as scientists are not able to exactly repeat experiments if they are not provided with all the files and settings of the initial research. This led us to create a framework to address these issues and allow users to create configurations for projects that can easily be shared between different teams and modified to scale up the number of components by exploiting the facility of manipulating text files (Spice netlists). This also makes our framework usable with ease in conjunction with any CAD tool and the underlying electrical simulator. For a similar reason, we created a *State Observer*, a verilogA component that allows convenient export of measurements from the circuit under study.

More in detail the simulation and analysis environment has been manually set up to allow maximum automation, by reducing the possible human errors. The main used tools have been Cadence Virtuoso to run the simulations, the Voltage ThrEshold Adaptive Memristor (VTEAM) verilogA memristor model, the verilogA State Observer and three Python3 programs handling the simulation parameters and the start of the simulations, the analysis of the results and the plot of the analysed data.

We started from the netlist of the circuits we wanted to simulate. We used Cadence Virtuoso to design, test and run the circuits, and we prepared the simulation environments we wanted to run. The resulting netlists have been different according to the simulations we ran, and will be treated in their respective chapters of this thesis.

3.1 VTEAM

In order to run memristive LIM simulations, the first approach has been to find a suitable memristor implementation to use. We started from the state of the art, noticing that we needed a voltage threshold memristor: the resistive value will start changing after a certain voltage threshold is reached. The memristor behaviour is simulated using the Voltage ThrEshold Adaptive Memristor (VTEAM) model [139], with the parameters shown in table 3.1. More in detail, its High Resistive State (HRS) is of 300 kΩ and LRS is of 1 kΩ, representing, respectively, logic 0 and logic 1.

The VTEAM model proposes to describe the behaviour of voltage-controlled memristors. This model extends the (TEAM) one [140], a general, flexible model which describes current-controlled memristors. The VTEAM model integrates the TEAM model with voltage-controlled memristors. Voltage controlled memristors allow the MAGIC and FELIX operations by enabling the full switching of the output memristor during the operations. The equations to describe a voltage-controlled time-invariant memristive device are the following:

$$\frac{dw}{dt} = f(w, v) \tag{3.1}$$

$$i(t) = G(w, v) \cdot v(t) \tag{3.2}$$

Name	Description	Value
model	Memristor model - VTEAM	4
window_type	window function - no window	0
dt [s]	numeric simulation time step (for Cadence Virtuoso)	1E-12
init_state [0:1]	The initial state of the state variable	0
Roff [Ω]	Memristor's maximum resistance	300000
Ron [Ω]	Memristor's minimum resistance	1000
D [m]	Physical width of the memristor	3E-09
w_multiplied	A normalization for the state variable (for Cadence Virtuoso)	1E+08
p_coeff	The value of p in the window functions	2
p_window_noise	A small noise to avoid boundary problems in window functions (for Cadence Virtuoso)	1E-18
x_c [m]	Normalized length for Simmons tunnel barrier	1.07E-10
a_on [m]	Upper bound of undoped region (Simmons tunnel barrier)	2E-09
a_off [m]	Lower bound of undoped region (Simmons tunnel barrier)	1.2E-09
K_on [m/s]	kon in Voltage ThrEshold Adaptive Memristor (TEAM)	-216.2
K_off [m/s]	koff in TEAM	0.091
Alpha_on	Nonlinearity power coefficient for TEAM	4
Alpha_off	Nonlinearity power coefficient for TEAM	4
v_on [V]	Threshold voltage in VTEAM	-1.5
v_off [V]	Threshold voltage in VTEAM	0.3
x_on [m]	Lower bound of undoped region (TEAM)	0
x_off [m]	Upper bound of undoped region (TEAM)	3E-09

Table 3.1: VTEAM model parameters.

Where w is an internal state variable, v is the voltage across the memristor, i is the current passing thought it, G is the conductance and t is the time. The equation (3.1) is developed as following for the VTEAM model:

$$\frac{dw(t)}{dt} = \begin{cases} k_{off} \cdot \left(\frac{v(t)}{v_{off}}\right)^{\alpha_{off}} \cdot f_{off}(w) & 0 < v_{off} < v \\ 0 & v_{on} < v < v_{off} \\ k_{on} \cdot \left(\frac{v(t)}{v_{on}-1}\right)^{\alpha_{off}} \cdot f_{on}(w) & v < v_{on} < 0 \end{cases} \tag{3.3}$$

Where k_{on} (negative value), k_{off} (negative value), α_{on} and α_{off} are constants; v_{on} and v_{off} are threshold voltages. The functions $f_{on}(w)$ and $f_{off}(w)$ correlate the derivative of the state variable with the state variable w. These are window functions that constrain the state variable within $[w_{ON},\ w_{OFF}]$.

The current–voltage relationship is the following:

$$i(t) = \left[R_{ON} + \frac{R_{OFF} - R_{ON}}{w_{OFF} - w_{ON}} \cdot (w - w_{ON}) \right]^{-1} \cdot v(t) \tag{3.4}$$

where R_{ON} and R_{OFF} are the memristor's minimum (LRS) and maximum resistance (HRS) when the state is, respectively, w_{ON} and w_{OFF}.

3.2 LIM Instruction Set

Once we had the memristor model to simulate, we moved on to the circuit netlist. The first goal was to simulate the memristor model, to then simulate a basic memory operation, e.g., the MAGIC NOR. The most basic netlist for a complete MAGIC NOR operation has to contain at least three memristors; for each memristor we put a voltage generator able to provide the correct voltage to the BLs to perform the SET operation, and another one to supply the control voltage enabling the MAGIC NOR operation, and finally some switches able to properly direct the current to therefore have the NOR gate configuration, but as well to enable the SET operation. Finally, with a slightly increased complexity, we could enable the RESET operation as well.

The second step to have the netlist ready to be simulated has been to provide the correct control sequence to control the switches and to enable the operations. As we mentioned before, the operation must be run with a voltage (described by the equations presented in chapter 2) for a certain amount of time. This meant being able to have the correct timings of the wanted operations and the needed voltage values.

Once the netlist setup is complete, the step 0) for any operation is to have the input value(s) on the input memristor(s), as shown in section 2.2.1. Then, as in the steps shown in the previous chapter, we need to SET the output memristor, then connect it to the ground, and, having the NOR gate configuration (as shown in table 2.1), we have to apply the control voltage V_0) on the two input memristors.

The controls managing the operations are complicated to be manually handled, considering that for each memristor we would need two voltage generators, one to control the switch (that can leave the memristor floating, connected to ground or to a voltage generator), and the other one handling the voltage applied to the memristor (in case this one

is connected to it). And on top of that, we needed an additional voltage generator handling the WL switch (connecting the WL to ground or leaving it floating) and the other one for a possible voltage input. Moreover, using the visual interface of Cadence Virtuoso (running on a server and being tunnelled through Secure Shell Protocol (SSH) or by using Virtual Network Computing (VNC)), the speed and reliability of the connection was making the setup of the basic complicated and it would have made any change to the setup complicated as well.

To manage this complexity in a more standard way, easier to handle, and more scalable (and to reduce human errors), we developed an Instruction Set of the LIM operations we targeted: this is a Python3 class that generates text files containing the needed voltage inputs that can be fed to the Cadence Virtuoso simulation. Furthermore, we had two *generations* of voltage generators: the first voltage generators we implemented were voltage generators that could take any voltage value from the text input, and they would provide the given voltage for a chosen amount of time. Afterwards we changed the implementation by putting a voltage generator for each voltage input we could have in the circuit; this allowed the circuit to better represent a real implementation and therefore be able to better study the consumption and the electromagnetic traces of the circuit. After several versions, implementations and tests, we reached a final version of the basic netlist. A basic netlist that can perform operations that require up to 3 memristors (such as the MAGIC NOR section 2.2.1) is shown in figure 3.1. As can be seen, there are 3 memristors (in_1, in_2, and out: the first two are the input memristors, and the last one is the output memristor. The switches we use are ideal switches with an open switch resistance of 1 TΩ and a closed switch resistance of 1 Ω. Each switch is driven by a voltage generator that provides a $V_{switch,\ off}$ or a $V_{switch,\ on}$ depending if we want the switch to be open or closed.

The Instruction Set can support the following operations:

- LD,
- NOP,
- MAGIC NOT,
- MAGIC NOR,

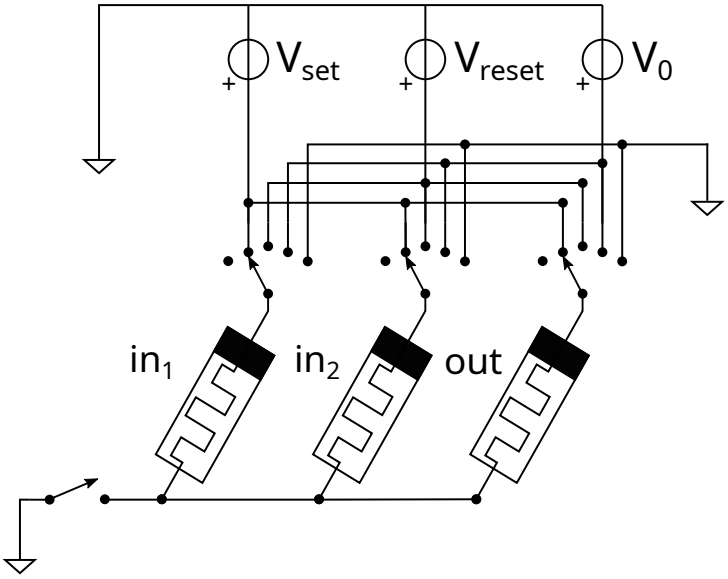


Figure 3.1: Basic netlist for operations with up to 3 memristors.

- FELIX NAND,
- FELIX OR,
- FELIX XOR,
- IMPLY.

The Instruction Set can be used in two ways: it can be used directly within a Python3 code or by using a pseudo-code approach, similar to writing an assembly code. In fact, the pseudo-code can be useful when we want to fast develop and test one or more operations one after the other. To do more in-depth simulation and analysis, the Python approach is more methodical and useful, i.e., we used it to perform several parametric sweeps.

3.2.1 Code examples

In the following are presented a few code examples showing how the LIM Instruction Set works. All the operations work by setting the various switches by allowing the wanted circuit. Every time that there is a new operation (or just a new step, even just a NOP) a transition time is inserted to avoid an abrupt voltage change.

The first example is the most needed operation: the LD operation (source code 3.1).

```
def f_ld(self, args, t_duration=t_write):
    self.current_time += self.t_transition  # allow a transtion time to avoid changing the voltage
    ↪ abruptly and not to have a rectangular waveform, that would not be realistic
    new_list_set = [False] * self.n_BLs  # SET voltage controller
    new_list_reset = [False] * self.n_BLs  # RESET voltage controller

    # Enable or disable SET or RESET for each memristor in the circuit
    for i in range(int(len(args) / 2)):
        new_list_set[args[0::2][i] - 1] = args[1::2][i] == 1
        new_list_reset[args[0::2][i] - 1] = args[1::2][i] == 0

    # Enable or disable the switches
    self.switches_control(
        t_duration,  # operation duration
        new_list_set,  # SET voltage
        new_list_reset,  # RESET voltage
        [False] * self.n_BLs,  # control voltage
        [False] * self.n_BLs,  # control_2 voltage
        [False] * self.n_BLs,  # control_3 voltage
        [False] * self.n_BLs,  # gnd
        True)  # WL switch enabled

    self.current_time += t_duration
```

Source code 3.1: LD operation snippet.

The second operation presented is the MAGIC NOT. Operations are always divided into two functions: the first handles the output memristor value, and then handles the actual application of the control voltage, as can be seen in source code 3.2.

```
def f_magic_not(self, operation_args):
    self.f_ld([operation_args[1], 1])
    self.f_not(operation_args[0], operation_args[1])

def f_not(self, source_mem_1, dest_mem):
    self.current_time += self.t_transition

    self.switches_control(
        self.t_not,  # operation duration
        [False] * self.n_BLs,  # SET voltage
        [False] * self.n_BLs,  # RESET voltage
```

```
            [col + 1 == int(source_mem_1) for col in range(self.n_BLs)],  # control voltage
            [False] * self.n_BLs,  # control_2 voltage
            [False] * self.n_BLs,  # control_3 voltage
            [col + 1 == int(dest_mem) for col in range(self.n_BLs)],  # gnd
            False)  # WL switch enabled

        self.current_time += self.t_not
```

Source code 3.2: MAGIC NOT snippet.

3.2.2 Supported operations

In this part we will show how the gates are enabled inside the circuit to execute the LIM operations.

LD

The load operation is used to perform a SET or a RESET operation. As these operations are used in the following operations, they are directly shown afterwards.

NOP

The NOP operation consists in leaving the memristive circuit floating.

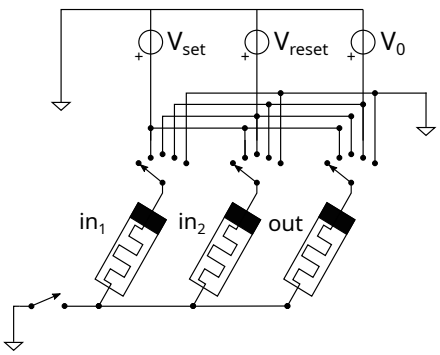
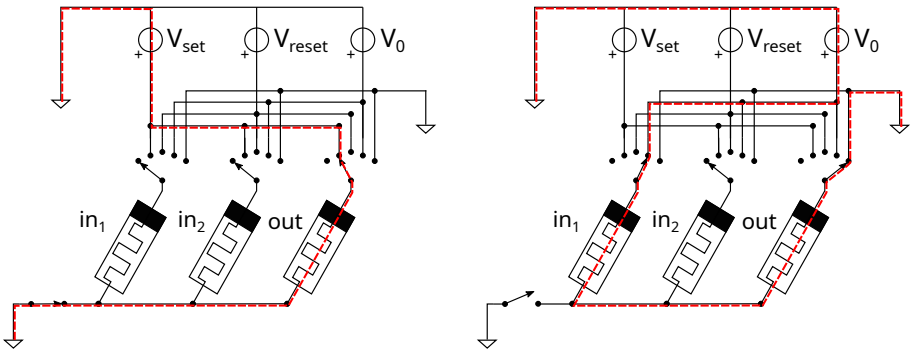


Figure 3.2: NOP operation on a 3-memristor circuit.

MAGIC NOT



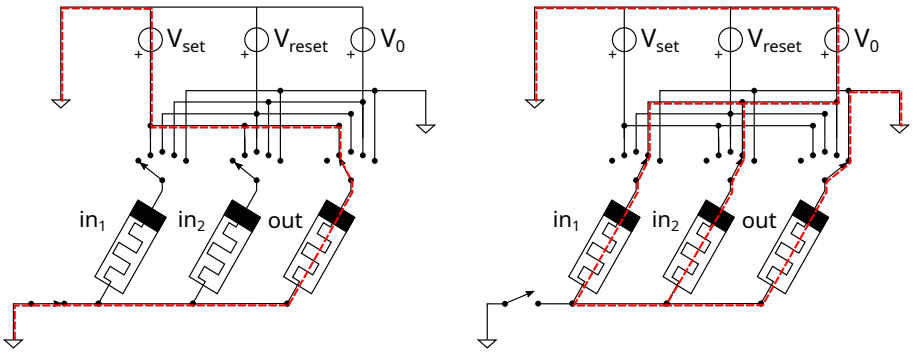
(a) Step 1: SET the output memristor.

(b) Step 2: Apply the control voltage V_0.

Figure 3.3: MAGIC NOT operation on a 3-memristor circuit.

MAGIC NOR and FELIX NAND

The MAGIC NOR and FELIX NAND operations have identical functioning, except for the applied voltage and the duration of the operation.

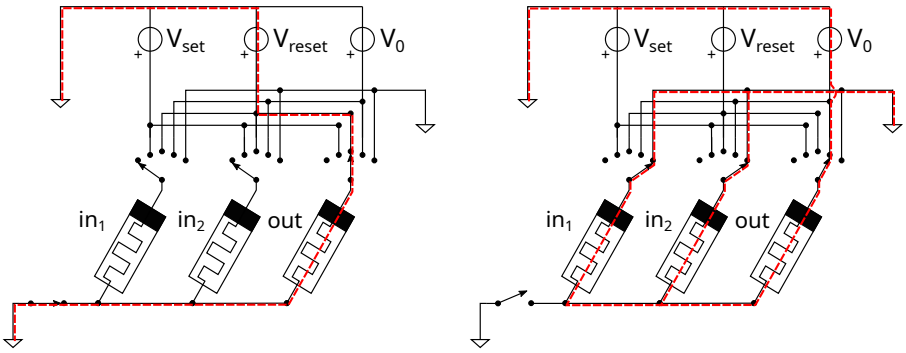


(a) Step 1: SET the output memristor.

(b) Step 2: Apply the control voltage V_0.

Figure 3.4: MAGIC NOR and FELIX NAND operations on a 3-memristor circuit.

FELIX OR

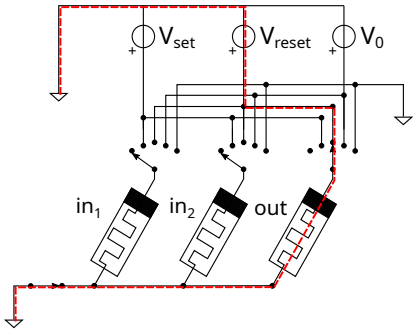


(a) Step 1: RESET the output memristor.

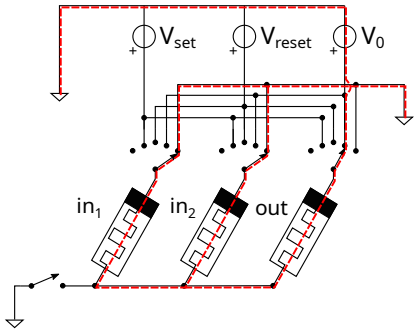
(b) Step 2: Apply the control voltage V_0.

Figure 3.5: FELIX OR operation on a 3-memristor circuit.

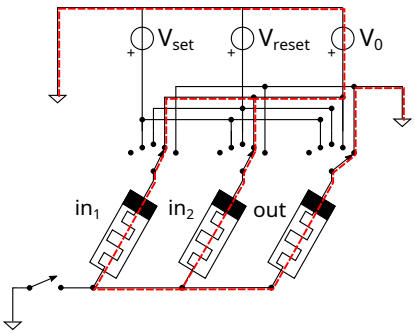
FELIX XOR



(a) Step 1: RESET the output memristor.



(b) Step 2: Apply the control voltage $V_{0,OR}$.



(c) Step 3: Apply the control voltage $V_{0,NAND}$.

Figure 3.6: FELIX XOR operation on a 3-memristor circuit.

IMPLY

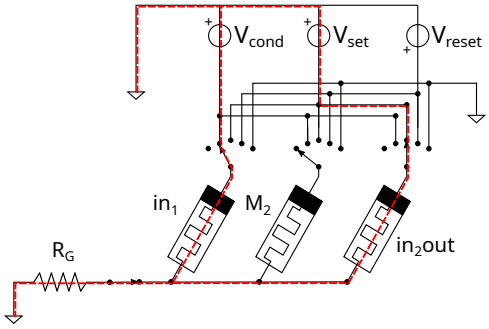


Figure 3.7: IMPLY operation on a 3-memristor circuit. V_{SET} and V_{COND} are in the opposite direction to the previous cases because the IMPLY gates has the memristors in the opposite direction to the other solutions.

3.3 State Observer

The state observer [141] is designed for Spice- and VerilogA-based simulators. It is a sub-circuit that is placed in the schematic and is connected to the components in need of measurements.

For transient simulations, the state observer will write the state measured every time step to a Comma-Separated Values (CSV) file, so that it can be easily loaded into any conventional software to analyse and extract information. For this reason, the simulation parameter `STEP_SIZE` found in the simulation environment is particularly important: it allows running the state analyser with the desired granularity. Moreover, as shown in source code 3.3, the state observer can be modified to save the wanted data only in certain cases, to allow a lighter and more application-specific log.

The idea of being able to externally save results in a user-defined file comes also useful in case we want to modify the parameters for the next simulations according to the results just obtained. As an example, we could increase or decrease the step size of parameters depending if we are close or not to the expected working behaviour.

Most of the commercially available software allows exporting simulation results into external files. This process is nevertheless carried out most of the time through a graphical user interface, which slows down the simulation iteration cycle, or the supported file types

are not the desired ones.

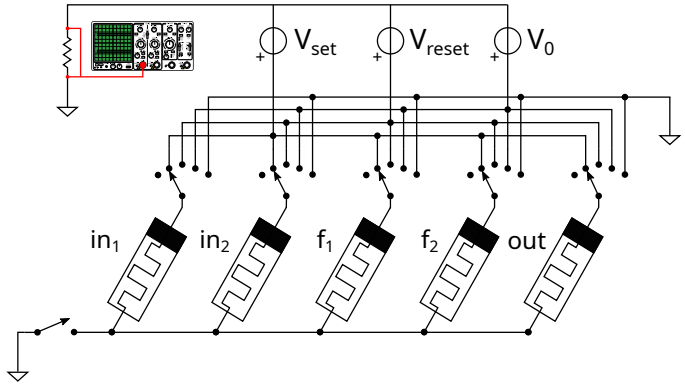


Figure 3.8: State Observer (on the right top corner), represented by an oscilloscope connected to a 5-memristor netlist.

In the practical case of our simulations and in the particular example of source code 3.3, we used it to save the state of two memristors in the execution of LIM operations. The goal of the simulation was to sweep over the control voltage(s) and the duration of the LIM operations to find the best working settings [142]. Hence, for this code, we concatenated the automatic generation of parameters with the state observer. We automatically generated the inputs and parameters to run the simulations and saved the simulation times at which each LIM operation is concluded. We also saved the input value(s), the expected output value and selected parameters, including the control voltage(s) and the duration of the operation. During the operation, the resulting file is read by the State Observer and is used to choose the simulation times at which it will save the desired states. Finally, the resulting CSV generated by the state observer will have all the fields coming from the builder and the read values:

- Provided by the builder (our main program)
  - Input value(s),
  - Expected output value,
  - Duration of the operation,
  - Control voltage,

  - Timestamp for save action execution;

- Added by the state observer

  - Read value(s).

This allows to have all the information about the executed operation in the same CSV and it supports directly performing data analysis from the collected data to assess the correct behaviour of the studied operations.

Since the state observer is defined in a VerilogA file, it can also be integrated with the builder in order to automatically generate the correct number of signals depending on the user-defined configuration. This simplifies the sweep of the input parameters, as the schematic itself will be rendered according to the configuration and the values generated for each iteration.

```
1  `include "constants.vams"
2  `include "disciplines.vams"
3
4  // define meter units for w parameter
5  nature distance
6      access = Metr;
7      units = "m";
8      abstol = 0.01n;
9  endnature
10
11 discipline Distance
12     potential distance;
13 enddiscipline
14
15 // define resistance units for res_value parameter
16 nature resistance
17     access = Ohm;
18     units = "ohm";
19     abstol = 0.01;
20 endnature
21
22 discipline Resistance
23     potential resistance;
24 enddiscipline
25
26 // State Observer Module
```

```
27  module state_observer(w_pos_1, w_pos_2, r_1, r_2);
28      // inputs
29      input w_pos_1, w_pos_2, r_1, r_2;
30      // types definition
31      Distance w_pos_1, w_pos_2;
32      Resistance r_1, r_2;
33
34      // internal variables
35      // time at which to save
36      real save_time;
37      // output file descriptor
38      integer fp_tt;
39      // input file descriptor (having the times at which
40      // to save and the expected results)
41      integer fp_tt_time;
42      //
43      string save_data;
44
45      // Analog behavior
46      analog begin
47          // initial Step
48          @(initial_step) begin
49              fp_tt_time=$fopen(expected_output,"r");
50              fp_tt=$fopen(output_file","a");
51
52              $fscanf(fp_tt_time, "%s", save_data);
53              $fscanf(fp_tt_time, "%g,%s", save_time, save_data);
54          end
55
56          // every time the state observer is executed
57          if($abstime > save_time && $abstime < save_time + 1.1 * STEP_SIZE) begin
58              $fwrite(fp_tt,"%.10g,%s,%.10g,%g,%g,%g,%g\n", save_time, save_data, $abstime, Ohm(r_1),
    ↪  Ohm(r_2), Metr(w_pos_1), Metr(w_pos_2));
59              if ($feof(fp_tt_time)) begin
60                  $finish(0);
61              end
62              else begin
63                  $fscanf(fp_tt_time, "%g,%s", save_time, save_data);
64              end
65          end
66
67          // final Step
68          @(final_step) begin
69              $fclose(fp_tt_time);
70              $fclose(fp_tt);
```

```
71            end
72        end
73    endmodule
```

Source code 3.3: Verilog-A source code to measure the resistance and threshold of 2 memristors.

3.4 Parametric Sweep using Cadence Virtuoso

Finally, having the entire set-up and the LIM Instruction Set ready with the possibility to be used in Python scripts, we prepared a program to perform parametric sweeps to assess the behaviour of the LIM implementations with the memristive devices we have chosen. Therefore, we designed a tool that is capable of sweeping various voltage ranges and duration of the voltage input. The idea was to check all the input combinations with respect to the expected output. In addition to this, by using the State Observer (section 3.3) and providing to it the expected result at a certain time, we have been able to obtain all the information needed to assess the proper functioning of the simulated operations. By doing so, we have been able to have a framework allowing us to run and test the different operations to then study the results.

3.5 Results extraction and plots

Having the results in a CSV file, we wanted to automate the extraction of the results, with some plots showing in a graphical way the effectiveness of the operations.

Since we could run different types of simulation and we had run bulky simulations having more than one operation in a single output file, we had to first treat and clean the data we wanted to work on, and then analyse it. Moreover, we wanted more than just one plot to fit them all, henceforth, we created three different programs able to split and take the desired lines from the CSVs.

The resulting set of Python scripts allows to analyse the results of the Cadence simulations run on the server. It takes as input one or more CSV files in the format given by our state observer.

There are three main files, to be used in order (or, if the state of the files is already

clean, the first ones may not be used).

1. Split CSV by operation type,
2. Split CSV by the values of the input memristors,
3. Analyse and plot from CSV.

An additional script, *Convert to PowerPoint*, can be used to automatically convert the output of the final script to a PowerPoint presentation to provide an easier interface than simple images and better present the results.

Split CSV by operation type This script allows to split the file(s) according to the operation type (e.g., NOR, OR, NAND, XOR, IMPLY, etc...). It does not take into account the different inputs (i.e., if they are at the nominal value or at a non-ideal value): before continuing the analysis it is necessary to split again the output files or take into account in the analysis code.

Split CSV by the values of the input memristors This code allows to split the files containing just an operation type. The output is a file for each operation and for each input couple. It generates temporary files to complete the operation.

Analyse and plot from CSV This script allows to analyse a CSV file that contains the results of the Cadence simulation. The CSV file must have:

- only one operation type,
- only one input couple (e.g., 1 kΩ-300 kΩ or 1.025 kΩ-299.5 kΩ).

The resulting plots are many, but the most significant and interesting ones are the following:

- scatter plot with boundaries
  - a scatter plot showing the points where the operation had worked with the strictest threshold,

  - several boundaries with different colours showing the *areas* where the operation had for each threshold,
  - plot legend explaining the colours (optional);

- scatter plot with a colour scale
  - a scatter plot showing the points where the operation had worked with selected thresholds, each one being shown with a different colour,
  - the points that were not included in the loosest threshold are represented in white: according to the chosen background colour we can show them or not;

- histogram showing the distribution of the points shown in the scatter plot explained above; the x axis indicate the ΔR, also called *diff*: the distance between the expected value (i.e., the correct result of the operation) and the read value.

To consider an operation as working, the results must be the same as the expected ones, for all the input combinations. Moreover, we wanted to know if some operation had changed the value of the input memristors, because it could have been a problem concatenating operations. To have a deeper understanding of how the operations behave, we have plotted the aforementioned plots, but with a finer-grain approach: we could select each input couple separately, or all together, taking therefore the worst-case scenario among them. Furthermore, we could select ΔR (i.e., the distance between the expected value and the read value, called *diff*) on the input memristors only, on the output memristor only, or on both. Figure 3.9 provides a schema of it.

The script has many, easy to modify options that allow to plot different features of the analysis.

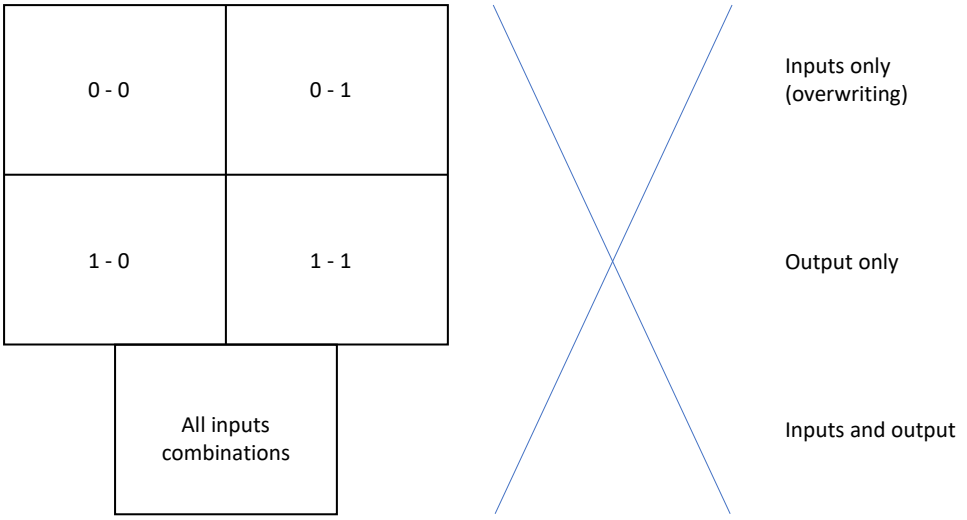


Figure 3.9: Schema showing the possible input choices for the plots. On the left side there are the possible inputs couples that can be chosen: 0-0, 0-1, 1-0 and 1-1. On the right side are shown the ΔR (i.e., the distance between the expected value and the read value) on the inputs only (showing therefore if and how the inputs changed after the operation), the output only (showing the efficacy of the operation) and finally both (showing the worst case scenario of all).

3.6 Final framework and capabilities

Finally, the complete framework we obtained is able to do the following:

- Read pseudo assembly code and generate the input for the netlist,
- Run Cadence simulation,
- Save Output,
- Save Expected Output (save_state):
  - Command given,
  - Input(s) given,
  - Expected output;
- Generate a “truth table” input for the given function that, for all the possible input combinations:

1. Loads the inputs,
2. Performs the given functions on the inputs,
3. Saves the state,
4. Executes a `NOP`;

- Additional script that enables to perform the same operations several times exploring the variation of the duration and the voltage of the input:
  1. Instantiate the class,
  2. Loop functions, voltages and function times.
- Take the results and plot with user-defined thresholds and colours.

4

Electrical Simulation of Boolean Operations with Ideal Memristive Elements

This chapter contains an in depth study of different LIM solutions based on electrical simulations and assuming ideal memristors. Only the LIM solutions considered best suited for use within a cryptocore were studied. Indeed some of the LIM solutions do not preserve the content of the input memristors after executing the operation (i.e., input-destructive operations); while others, like MAGIC and FELIX promise non-input destructive operations, as classical computation paradigms, allowing therefore the re-use of data for several operations. This chapter is dedicated to the second class of LIM solutions as they are better suited for our target application. While their ability preserve input data has been shown in the literature for ideal operating condition, this work is showing that in reality their behaviour is far from ideal when the operation condition vary. We analysed the electrical behaviour of some significant LIM implementations (MAGIC NOT, MAGIC NOR, FELIX OR and FELIX NAND) under various operating conditions to see how they performed

under several aspects, e.g., precision and speed. Our results show that it is not trivial to guarantee non-input destructive operations (e.g., in the case of FELIX NAND) and there is a real difficulty in concatenating several operations due to non-ideal intermediate results. Among the studied implementations, MAGIC NOR and FELIX NAND functions are functionally complete, i.e., each one of them can be used to implement all logic functions in one or more steps. Therefore, these functions are the most interesting among the ones studied. The contributions presented in this chapter are:

- The in-depth analysis of the operating conditions of MAGIC NOT, MAGIC NOR, FELIX OR, FELIX NAND, FELIX XOR and IMPLY with the identification of the ranges in which the operations are performed correctly, highlighting the values of control signals which lead to input-destructive operations;
- An introduction of the possible XOR implementations using the solutions under analysis;
- The demonstration that while the operation is performed correctly, the resistance of the output memristor does not always reach its ideal value (either low or high).

4.1 Motivation

The LIM operations introduced in section 2.2.1 have the control voltage range that is described by means of equations giving a working range. We evaluated such values and they are represented in figure 4.2. While these equations give a range of theoretical values for the control voltage amplitude, the duration of the control voltage strongly depends on the dynamics of the memristor, but it is not trivial to derive its analytical expression. Furthermore, memristors exhibit an adjustable resistive range extending from LRS to HRS. Although it is straightforward to employ their digital properties for assessing logic 1 and 0, leveraging their analog capabilities becomes imperative for LIM operations. This analog utilisation substantially reduces the margin of error, enhancing the precision of such processes. Therefore, to identify combinations of control voltage amplitude and duration for which the operation is performed successfully, we have resorted to electrical simulations. When performing the electrical simulations, we have identified 4 scenarios:

1. the operation has been performed correctly without any effect on the input;
2. the operation has been performed correctly but the information on the input was lost;
3. the operation has been performed correctly but the resistive state of the output memristor did not reach the nominal HRS or LRS, as shown in figure 4.1;
4. the operation has failed.

The third scenario has led us to the assumption that concatenating operations in which the output of the first operation (which did not reach the nominal HRS or LRS) is used as input for the next operation could lead to a wrong result. We proved this assumption right as explained later in this chapter and in chapter 5.

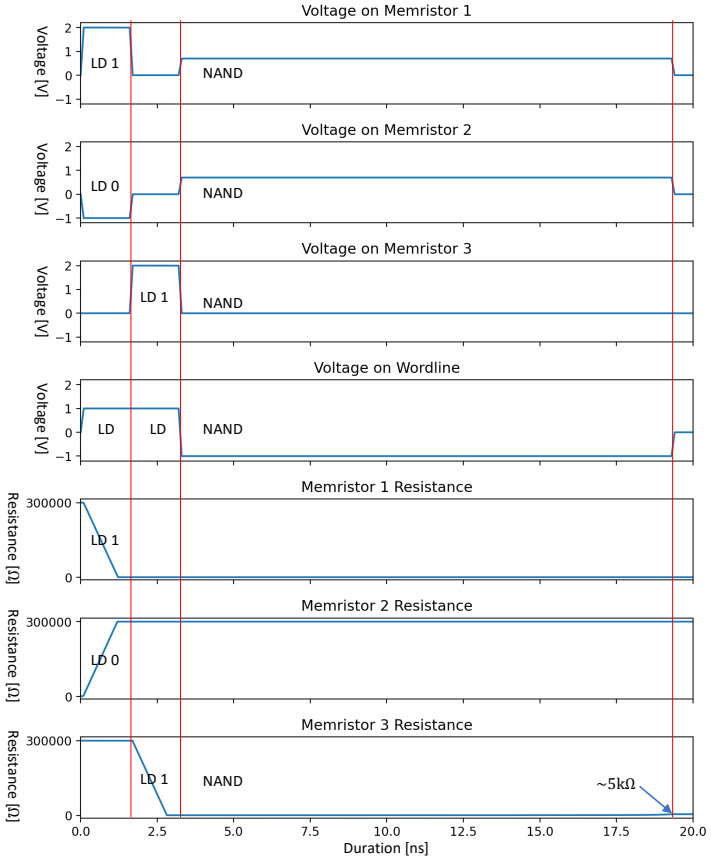


Figure 4.1: Waveform of a FELIX NAND operation, where the final resistive state does not reach the nominal LRS. LD is the load operation on a memristor (SET if LD 1, RESET if LD 0). At the end of the NAND operation, the final resistive value of the output memristor (memristor 3) is not 1 kΩ as expected, but 5 kΩ. The vertical red lines indicate the start of the next operation.

4.2 Simulation environment and preliminary analysis

The basic simulation environment is explained in chapter 3. Theoretical values for V_0 (calculated with equation (2.2) range between 0.60 V and 1.50 V for the NOR operation, and between 0.40 V and 0.45 V for the NAND operation (calculated with equation (2.3)), and 0.5 V and 2.5 V for the OR operation. To perform the in-depth analysis, the input voltage has been swept from 0.20 V to 2 V with a step of 0.1 V. Based on published results about the duration, we considered all durations ranging from 0.25 ns to 20 ns with a step of 0.25 ns. Actual ranges depend on the operation. In the case of the IMPLY operation, the load resistance R_G has been analysed in the range between 500 Ω and 60 kΩ, with a 500 Ω step.

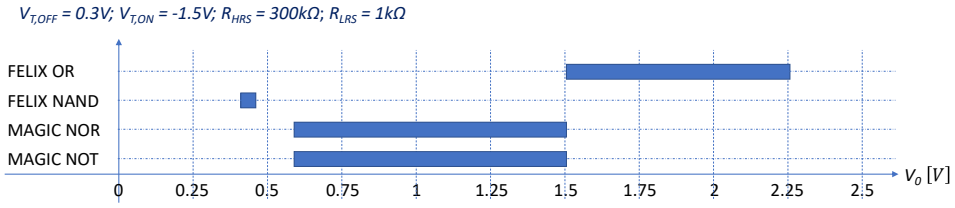


Figure 4.2: Theoretical ranges of the control voltages V_0 that allow a correct operation for MAGIC NOT and NOR, and FELIX NAND and OR operations.

For each control voltage setup (i.e., couple of amplitudes and duration), we have simulated the execution of MAGIC NOT, MAGIC NOR, FELIX NAND and FELIX OR functions applying all input combinations (i.e., 0,1 for MAGIC NOT and 00, 01, 10, 11 for the other operations) and the final states of the three memristors (the two inputs and the output) have been stored and analysed. We performed a preliminary analysis (with much larger sweep steps) on the MAGIC NOR operation, the results of which are shown in figure 4.3. For each combination of voltage amplitude and duration, we have plotted the outcome of the operation using the following colour scheme:

1. *Dark green*, when the operation has been performed correctly for all (2 or 4) input combinations, and the input memristor did not lose their state (non-input-destructive operation);

2. *Light green*, when the operation has been performed correctly for all (2 or 4) input

combinations but at least one of the input memristor lost its state for at least one input combination (input-destructive operation);

3. *White*, the output is wrong for at least one input combination.

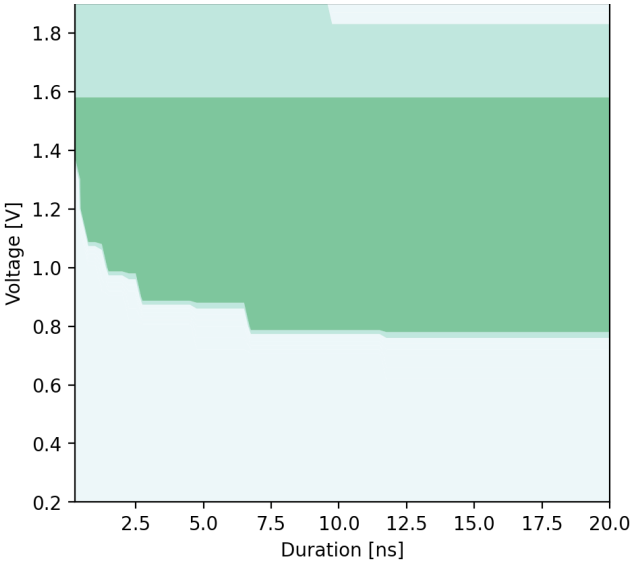


Figure 4.3: Results of the preliminary simulation of the NOR Operation.
In dark green is represented where the operation performed correctly, in light green where the operation performed correctly for all 4 input combinations but at least one of the input memristor lost its state for at least one input combination (input-destructive operation) and in white the operation did not work correctly.

Results in figure 4.3 are in agreement with the theoretical analysis. Indeed, we observed correct operations when the voltage amplitude ranges between 0.7 V and 1.55 V. The slight difference is due to the parameters of the memristive model that we used. After the MAGIC NOR, we performed a similar analysis on the FELIX NAND operation, where no combinations of amplitude and duration generated the correct results. This can be due to the fact that the theoretical voltage range for correct operation is very narrow and to a non-optimal setup of the memristive model. Another possibility could be that the step we had used for the initial sweep (0.05 V) was too big to find a working couple of voltage and duration.

4.3 Introduction of *non-ideal values* and thresholds

In the previous section, we have used a coarse-grained approach to the simulation and we have considered as correct results only the cases where the output memristor reaches exactly the nominal values of HRS or the LRS. This explains why all the simulated NAND operations were considered flawed and why the range of control voltages for correct NOR operations is narrower than the theoretical one. Nevertheless, on a closer analysis, we have observed that combinations of amplitude and duration exist for which the output memristor reaches resistive values close to the nominal HRS or LRS (close enough so that they can be read as correct logic values). We have defined these states that cannot reach the nominal value but can reach close enough to it as **non-ideal HRS** and **non-ideal LRS**. Due to this observation, we updated the conditions for an operation to be considered correct, i.e., an operation is correct if the resistive value of the updated memristor is between the *non-ideal* HRS and nominal HRS when a logic 0 is expected and between nominal LRS and the *non-ideal* LRS when a logic 1 is expected. Figures 4.4 and 4.5 show the colour-coded space of control voltages for the FELIX NAND operation, under two different groups of non-ideal values: in figure 4.4 non-ideal LRS=1.3 kΩ and non-ideal HRS=290 kΩ, while in figure 4.5 non-ideal LRS=50 kΩ and non-ideal HRS=150 kΩ. It is worth noticing that the higher the distance between nominal and non-ideal state, the larger is the control voltage range where the LIM operation is considered correct. This knowledge can lessen the restrictions on the voltage regulators (to generate the control voltages).

From this we have noticed that there were operations able to change the value of the output memristor, but we didn't know if it was sufficient to have results stable enough. This point will be discussed in chapter 5.

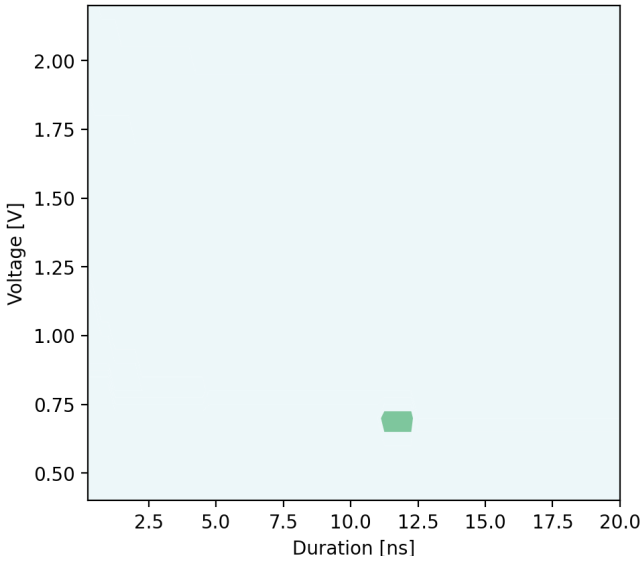


Figure 4.4: Results of the preliminary simulation of the NAND operation with non-ideal LRS=1.3 kΩ and non-ideal HRS=290 kΩ for reading the output value. The colour code is explained in section 4.2.

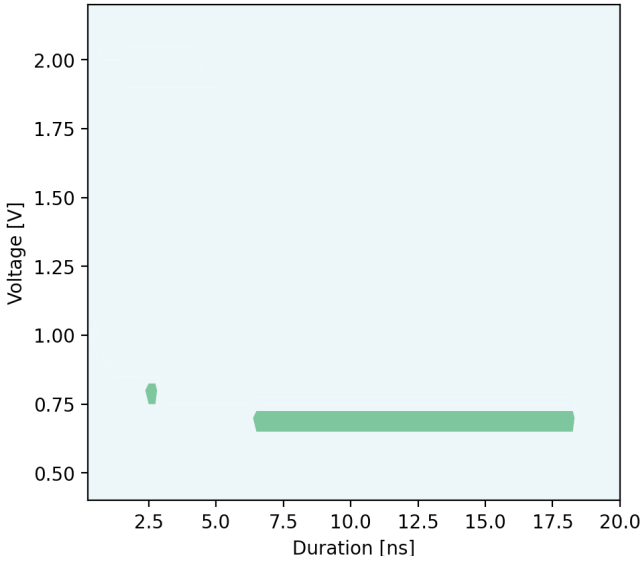


Figure 4.5: Results of the preliminary simulation of the NAND operation with non-ideal LRS=50 kΩ and non-ideal HRS=150 kΩ for reading the output value. The colour code is explained in section 4.2.

4.4 Concatenation of operations

One of the major advantages of the Stateful Logic in Array paradigm is that operations could be concatenated (i.e., using the output of one operation as input for the next one) or that one or more inputs of an operation could be used for more operations. This can first of all bring a significant speed up of the operations execution. As we introduced in table 2.11, most of the operations that we could do need more than just one step of a primitive operation. For example the AND operation would need 6 steps being implemented by MAGIC, 4 steps with FELIX, 5 with IMPLY and 4 (plus 3 reads) with RMAJ. As we explained in chapter 2, we already know the range of voltage values where the operations could work, but we don't know the time the various operations need to reach the nominal value to be considered correct. At the end of this chapter (section 4.5) we show the study we did to assess when the operations are working, and as a consequence we have been able to calculate the duration for each operation and to discuss even further about their usability.

MAGIC-based AND (in_1, in_2, f_1, f_2, out) – 6 steps, 5 memristors	
1 – 2)	MAGIC NOT(in_1, f_1)
3 – 4)	MAGIC NOT(in_2, f_2)
5 – 6)	MAGIC NOR (f_1, f_2, out)

Table 4.1: MAGIC-based AND basic steps. in_1 and in_2 represent the input memristors, f_1 and f_2 represent functional memristors, i.e., the memristors used to store intermediate results of the operation, while out represents the output memristor. `MAGIC NOT` and `MAGIC NOR` represent the whole operations, i.e., the SET of the output memristor (the last on between the parenthesis) and the application of the control voltages $V_{0,\ MAGIC\ NOT}$ and $V_{0,\ MAGIC\ NOR}$, respectively to the BLs of the input memristors.

All encryption algorithms are based on the XOR operation. It is fundamental in algorithms such as Data Encryption Standard (DES) [143], Advanced Encryption Standard (AES) [144, 145], and PRESENT [146], where the plaintext is XORed with the secret key. We have chosen an XOR gate as target circuit for our analysis, starting from the assumption that if this operation is not secure, the whole algorithm is not secure either. Therefore, having prepared the basic operations, we built XOR operations with selected implemen-

FELIX-based AND (in_1, in_2, f_1, out) – 4 steps, 4 memristors	
1 – 2)	FELIX NAND (in_1, in_2, f_1)
3 – 4)	MAGIC NOT(f_1, out)

Table 4.2: FELIX-based AND basic steps. in_1 and in_2 represent the input memristors, f_1 represents a functional memristor, i.e., a memristor used to store intermediate results of the operation, while out represents the output memristor. `MAGIC NOT` and `FELIX NAND` represent the whole operations, i.e., the SET of the output memristor (the last on between the parenthesis) and the application of the control voltages $V_{0,\,MAGIC\ NOT}$ and $V_{0,\,FELIX\ NAND}$, respectively, to the BLs of the input memristors.

IMPLY-based AND (in_1, in_2, f_1, out) – 5 steps, 4 memristors	
1 – 2)	IMPLY NOT (in_1, f_1)
3)	IMPLY (in_1, f_1)
4 – 5)	IMPLY NOT (f_1, out)

Table 4.3: IMPLY-based AND basic steps. in_1 and in_2 represent the input memristors, f_1 represents a functional memristor, i.e., a memristor used to store intermediate results of the operation, while out represents the output memristor. `IMPLY` represents the application of the control voltage V_0 to the BLs of the input memristors. `IMPLY NOT` is already a composite operation.

tations. The analysis we showed in section 2.3 indicated that the FELIX solution, on the paper, would require a much lower number of memristors and steps to achieve the XOR operation and the full adder than the MAGIC solutions. Therefore, we had to test it and assess it using our simulations.

The XOR operations use from three to five memristors (in_1, in_2, f_1, f_2 and out): the first two are the input memristors, the third and the fourth are functional memristors, i.e., memristors used to store intermediate values of the operation, and finally the last one is the output memristor. FELIX XOR does not need functional memristors.

RMAJ-based AND (in_1, in_2, f_1, out) – 4 + 3R steps, 4 memristors	
1)	Write 0 (out)
2 – 3, $r1$)	RMAJ NOT (in_2, f_1)
$r2$ – $r3$)	Read (in_1, f_1)
4)	RMAJ (in_1, f_1, out)

Table 4.4: RMAJ-based AND basic steps. in_1 and in_2 represent the input memristors, f_1 represents a functional memristor, i.e., a memristor used to store intermediate results of the operation, while out represents the output memristor. `Write 0` represents the RESET operation (that allows to have the output memristor at HRS). `RMAJ` represents the application of the control voltage V_0 to the BLs of the input memristors. `RMAJ NOT` is already a composite operation.

4.4.1 MAGIC-based XOR

In order to implement XOR operation with MAGIC, it is necessary to concatenate multiple NOR and NOT operations as shown in table 4.5.

MAGIC-based XOR (in_1, in_2, f_1, f_2, out) – 10 steps, 5 memristors	
1 – 2)	MAGIC NOT (in_1, f_1)
3 – 4)	MAGIC NOT (in_2, out)
5 – 6)	MAGIC NOR (f_1, out, f_2)
7 – 8)	MAGIC NOR (in_1, in_2, f_1)
9 – 10)	MAGIC NOR (f_1, f_2, out)

Table 4.5: MAGIC-based XOR steps. in_1 and in_2 represent the input memristors, f_1 and f_2 represent functional memristors, i.e., the memristors used to store intermediate results of the operation, while out represents the output memristor. `MAGIC NOT` and `MAGIC NOR` represent, respectively, the whole MAGIC NOT and MAGIC NOR operations, i.e., the SET of the output memristor (the last in parentheses) and the application of the control voltages $V_{0,\ MAGIC\ NOT}$ and $V_{0,\ MAGIC\ NOR}$, respectively, to the BLs of the input memristors.

Inputs	Step 1 f_1 = NOT(in_1)	Step 2 out = NOT(in_2)	Step 3 f_2 = NOR(out,f_1)	Step 4 f_1 = NOR(in_1,in_2)	Step 5 out = NOR(f_1,f_2)
00	1	1	0	1	0
01	1	0	0	0	1
10	0	1	0	0	1
11	0	0	1	0	0

Table 4.6: MAGIC-based XOR steps and values. Under the steps are the resulting values of each operation.

4.4.2 FELIX-based XOR

The FELIX XOR solution is made of a FELIX OR operation followed by a FELIX NAND operation, both executed on the same memristors, reusing the input memristors and without initialising the output memristor. In this case, if the FELIX OR performed similarly to the FELIX NAND, we would have a case of a concatenated operation having both non-ideal inputs and output, and it could cause issues.

FELIX-based XOR (in_1, in_2, out) – 3 steps, 3 memristors	
1 – 2)	FELIX OR (in_1, in_2, out)
3)	FELIX NAND (in_1, in_2, out)

Table 4.7: FELIX-based XOR steps. in_1 and in_2 represent the input memristors, while out represents the output memristor. `FELIX OR` represents the whole operation, i.e. the RESET of the output memristor (the last in between the parentheses) and the application of the control voltages $V_{0,\ FELIX\ OR}$ to the BLs of the input memristors. `FELIX NAND` is, in this case, just the application of the control voltage $V_{0,\ FELIX\ NAND}$ to the BLs of the input memristors. Applying the control voltage $V_{0,\ FELIX\ NAND}$ on the output without initialising it is what allows to use the result of the FELIX OR operation and to make the XOR operation possible.

Inputs	Step 1 out = 0	Step 2 out = OR(in_1, in_2)	Step 3 out = NAND_V_0(in_1, in_2)
00	0	0	0
01	0	1	1
10	0	1	1
11	0	1	0

Table 4.8: FELIX-based XOR steps and values. In this case the first operation (i.e., the FELIX OR operation) is split into its steps: 1) `Write 0` and 2) the application of the control voltage V_0. The FELIX NAND operation is only partially executed: the way to have the correct result is by running a FELIX NAND after a FELIX OR *without* preparing the output memristor, in order to exploit the result of the previous operation.

4.4.3 IMPLY-based XOR

The IMPLY XOR solution is made by several steps, as shown in tables 4.9 and 4.10. What makes this operation so long is the fact that IMPLY uses as input *and* output the same memristor: in the case of an XOR, where reusing the inputs can speed up the operation, the IMPLY operation is forced to copy the value of the inputs to use it, therefore increasing the duration of the operation.

IMPLY-based XOR (in_1, in_2, f_1, f_2, out) – 13 steps, 5 memristors	
1 – 4)	IMPLY COPY (in_1, f_2, f_1)
5)	IMPLY (in_2, f_1)
6 – 7)	IMPLY NOT (f_1, out)
8 – 11)	IMPLY COPY (in_2, f_1, f_2)
12)	IMPLY (in_1, f_2)
13)	IMPLY (f_2, out)

Table 4.9: IMPLY-based XOR steps. in_1 and in_2 represent the input memristors, f_1 and f_2 represent functional memristors, i.e., memristors used to store intermediate results of the operation, while out represents the output memristor. `IMPLY` represents the application of the control voltage V_0 to the BLs of the input memristors. `IMPLY COPY` and `IMPLY NOT` are composite operations. To see the full steps unfolded, see table 4.10.

We investigated the XOR-based IMPLY operation and determined it could to be too lengthy to be worth implementing. Therefore, we decided to let the performance of the IMPLY operation dictate the choice.

IMPLY-based XOR (in_1, in_2, f_1, f_2, out) – 13 steps, 5 memristors					
1-4)	IMPLY COPY(in_1,f_2,f_1)	1-2)	IMPLY NOT(in_1,f_2)	1)	Write 0(f_2)
				2)	IMPLY(in_1,f_2)
		3-4)	IMPLY NOT(f_2,f_1)	3)	Write 0(f_1)
				4)	IMPLY(f_2,f_1)
5)	IMPLY(in_1,f_1)			5)	IMPLY(in_1,f_1)
6-7)	IMPLY NOT(f_1,out)			6)	Write 0(out)
				7)	IMPLY(f_1,out)
8-11)	IMPLY COPY(in_2,f_1,f_2)	8-9)	IMPLY NOT(in_2,f_1)	8)	Write 0(f_1)
				9)	IMPLY(in_2,f_1)
		10-11)	IMPLY NOT(f_1,f_2)	10)	Write 0(f_2)
				11)	IMPLY(f_1,f_2)
12)	IMPLY(in_2,f_2)			12)	IMPLY(in_2,f_2)
13)	IMPLY(f_2,out)			13)	IMPLY(f_2,out)

Table 4.10: IMPLY-based XOR steps. in_1 and in_2 represent the input memristors, f_1 and f_2 represent functional memristors, i.e., memristors used to store intermediate results of the operation, while *out* represents the output memristor. `IMPLY` represents the application of the control voltage V_0 to the bitlines of the input memristors. `IMPLY COPY` and `IMPLY NOT` are composite operations, and are therefore immediately unfolded and described in the table.

4.5 Parametric Sweep

After having analysed the results of the coarse-grained simulation section 4.2, we noticed that a finer-grained analysis could have given us more insight in cases where it seemed there was no working solutions, e.g., the FELIX NAND operation. Therefore, we used the simulation environment of chapter 3 and for each control voltage setup (i.e., couple of amplitude and duration) and for each load resistance (for the IMPLY operation) we have simulated the execution of the LIM functions applying all input combinations (i.e. 0, 1 or 00, 01, 10, 11) saving the final state of the used memristors. Among the data extracted from the analysis of the results, there are: 1) the input correctness, i.e., the values of the inputs remained the ones that have been written before executing the operation, 2) the output correctness, i.e., the output state is the expected one, 3) the difference from the expected value for each memristive value, calculated doing:

$$diff = \left| out_{expected} - out_{read} \right| \tag{4.1}$$

We calculated the maximum *diff* (that we have previously mentioned as ΔR as well) for each set of amplitude and duration of the control voltage (and also load resistance, for IMPLY): this allows us to take the worst case scenario for all the input combinations. These cumulative data allowed us to pinpoint the settings that are the most suitable for each operation. Using these data, we plotted graphs indicating for which settings an operation was working and, if so, how close the read memristive values were to the nominal expected values. To do so, five different colours (see figure 4.6) have been used in the following graphs to indicate five thresholds, based on the diff explained in equation (4.1):

- Purple: the read values are the expected nominal resistive values (with a tolerance of 50 Ω);
- Dark blue: diff is less than 5 kΩ;
- Light blue: diff is less than 10 kΩ;
- Green: diff is less than 50 kΩ;
- Yellow: diff is less than 149.5 kΩ.

The blank part in the scatter plots is where the operation did not work: diff is greater than $(HRS - LRS)/2$, thus not allowing a clear distinction between the two states.

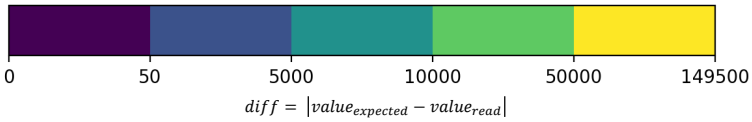
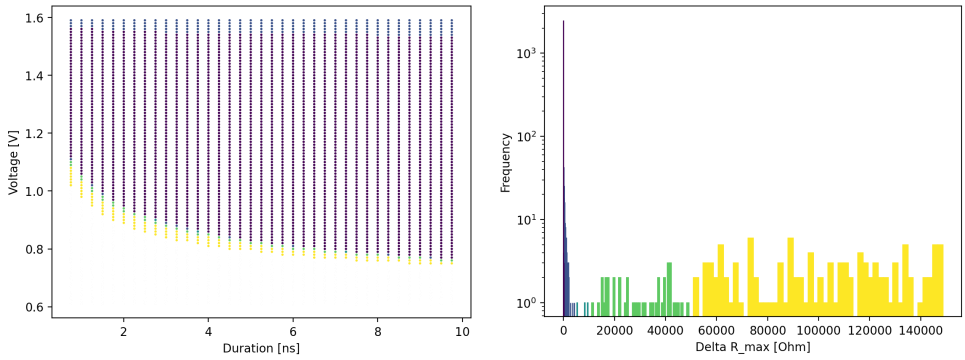


Figure 4.6: Key for the plots.

These operations we simulated use two or three memristors: in_1, (in_2), and out or, in the case of IMPLY, in_1 and $in_2 out$. in_1 and in_2 are the input memristors, $in_2 out$ is the input/output memristor used by the IMPLY operation, and out is the output memristor.

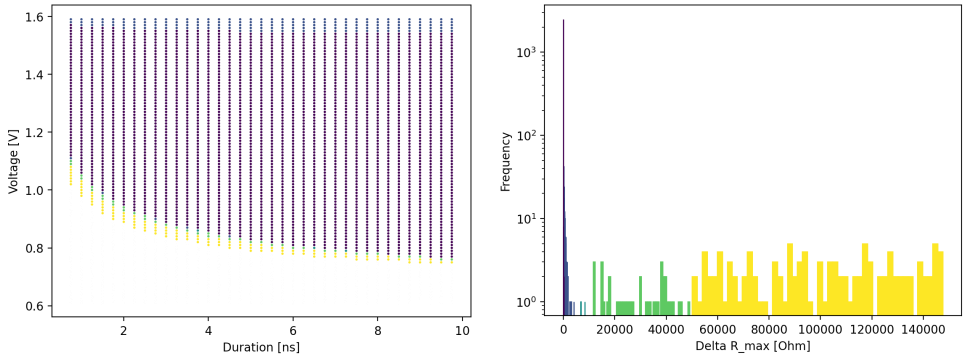
4.5.1 MAGIC NOT and NOR

The behaviour of the MAGIC NOT (figure 4.7) and MAGIC NOR (figure 4.8) operations is coherent with the theoretical analysis. Indeed, for the NOR operation we observed correct operations when the voltage amplitude ranges between 0.7 V and 1.55 V. The slight difference is due to the parameters of the memristive model we used. The two operations have very similar behaviour because the main difference between them is the number of input memristors: the operation is the same in terms of where the voltage is applied and the initial value of the output memristor. In fact, they behave in a similar manner and they have very similar working areas.



(a) (b)

Figure 4.7: Results of the simulation of the MAGIC NOT operation. (a) Plot showing the couples of control voltage V_0 and duration of the pulse (coloured) that provide working operations. (b) Histogram showing the frequency and the values of diff (the max distance from the expected value in the two memristors). This plot is in *logarithmic scale*.
The colour code is explained in figure 4.6.

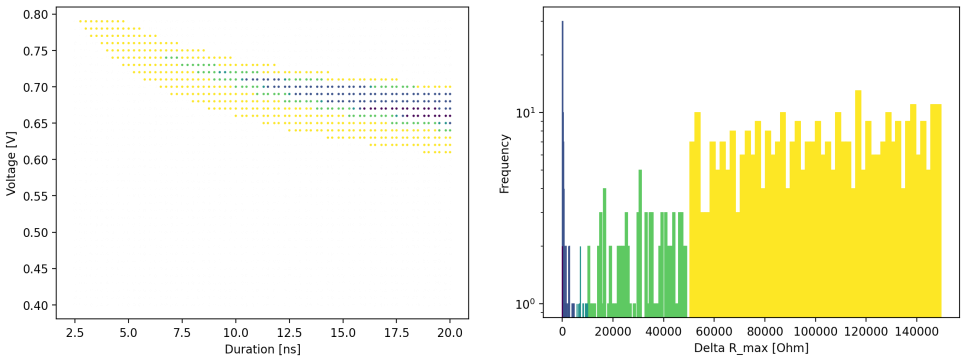


(a) (b)

Figure 4.8: Results of the simulation of the MAGIC NOR operation. (a) Plot showing the couples of control voltage V_0 and duration of the pulse (coloured) that provide working operations. (b) Histogram showing the frequency and the values of diff (the max distance from the expected value in the two memristors). This plot is in *logarithmic scale*.
The colour code is explained in figure 4.6.

4.5.2 FELIX NAND

The behaviour of the FELIX NAND (figure 4.9) operation did not behave as well as the MAGIC counterpart. Indeed, no combination of amplitude and duration has generated a result within the nominal expected values. This can be due to the fact that the theoretical voltage range for correct operation is very narrow, and to a non-optimal setup of the memristive model. Since the best obtained result is around 20 Ω from the expected value, (it has been obtained with a control voltage of 0.66 V and a pulse duration of 18 ns), we chose to allow a tolerance up to 50 Ω for the strictest threshold. The plot therefore shows a small area where we have a correct operation. The NAND operation, as explained in chapter 2, is set up as the MAGIC NOR operation but with a lower operating voltage, not allowing the output memristor to change its state from 1 to 0, for the 01 and 10 inputs couple cases. This makes the control voltage boundaries very strict compared to the MAGIC NOR limits, and, indeed, this can be observed comparing figure 4.8 and figure 4.9. From the same pictures it can also be observed that the working values of the NAND operation are just below the ones of the NOR operation, as expected. Similarly to the MAGIC NOR operation, there is a slight difference with the theoretical values for the control voltage that we calculated, due to the parameters of the memristive model we used.



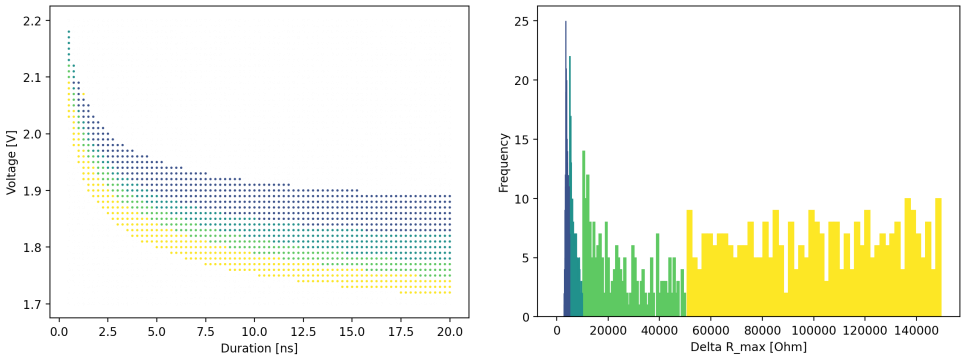
(a) (b)

Figure 4.9: Results of the simulation of the FELIX NAND operation. (a) Plot showing the couples of control voltage V_0 and duration of the pulse (coloured) that provide working operations. (b) Histogram showing the frequency and the values of diff (the max distance from the expected value in the two memristors). This plot is in *logarithmic scale*.

The colour code is explained in figure 4.6.

4.5.3 FELIX OR

Our simulations of the FELIX OR (figure 4.10) operation have shown that there is no control combination for which the operation generates a result having the nominal resistive state for inputs and outputs. Moreover, the FELIX OR easily changes the resistive value of the input: the control voltage is high enough that, during the operation, when the output memristor gets close to the nominal LRS, there is enough current flowing through the input memristors that is able to modify the resistive value of the inputs. The best results that we obtained resulted having the diff value for the output memristor around 2.7 kΩ, with a change in the inputs of 334 Ω, and it has been obtained with a control voltage of 2.03 V and a duration of 1.75 ns.



(a) (b)

Figure 4.10: Results of the simulation of the FELIX OR operation. (a) Plot showing the couples of control voltage V_0 and duration of the pulse (coloured) that provide working operations. (b) Histogram showing the frequency and the values of diff (the max distance from the expected value in the two memristors). This plot is in *linear scale*.

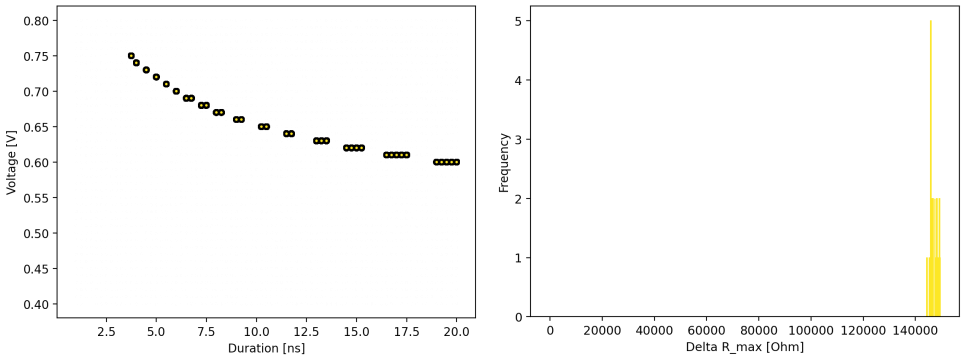
The colour code is explained in figure 4.6.

4.5.4 FELIX XOR

Considering that the FELIX OR operation is the first step of the FELIX XOR two-cycle operation and that the next step (FELIX NAND) does not initialise the output memristor, we had serious doubts about finding a wide working area, or even finding a working area at all, even before actually simulating the operation. Moreover, we had seen that the FELIX NAND operation already has a very limited set of control voltages and pulse durations that allow an operation whose results are close to the ideal ones. Taking into account these initial considerations, we were not expecting a good and ideal behaviour of the XOR operation. In fact, the FELIX XOR operation behaviour (as seen in figure 4.11), is far away from the ideal one: the only threshold that highlights some results is the least strict, and the diff values, as can be seen from the histogram (b) in the figure, are all above 140 kΩ. This behaviour is due to the FELIX OR results being too far from the nominal ones to allow the proper functioning of the subsequent FELIX NAND operation. This operation would be really difficult to implement and it would be impossible to enable the concatenation of operations after this one.

The best result obtained for the nominal inputs is around 144 kΩ from the expected value (for the output memristor; the diff value of the input memristor is negligible) and it has been obtained with a control voltage V_{OR} of 1.94 V and a pulse duration of 3.75 ns for the FELIX OR operation, and a control voltage V_{NAND} of 0.61 V and a pulse duration of 17 ns for the FELIX NAND operation.

As can be noted, the FELIX XOR operation works at its best with voltage values and durations of the operations that are different from those used for the FELIX NAND and FELIX OR operations. This implies that the voltage regulator will need to take care of these additional voltages, and it will not be possible to treat the XOR operation as just a concatenation of the FELIX NAND and OR, but it would be a whole new operation that, on top of it, does not show results interesting enough. Therefore, the FELIX XOR operation would be very difficult to implement and use in a concatenated manner: the very strict input voltage constraints and the final resistive values of the output, which would surely need a refresh, would not make it interesting enough for our application.



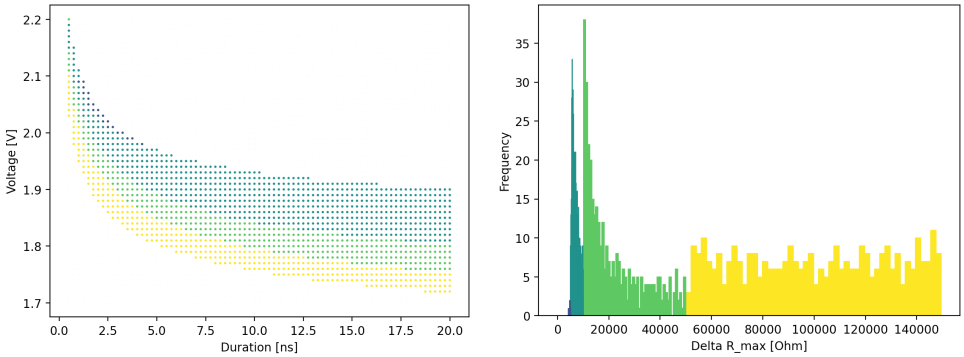
(a) (b)

Figure 4.11: Results of the simulation of the FELIX XOR operation. In this case, only the plot for the FELIX NAND operation is shown (being it the second step of the FELIX XOR operation): the best couple of voltage and duration of the FELIX OR operation that brought to the case shown here has already been calculated. (a) Plot showing the control voltage V_0 and the duration of the pulse couples (coloured) that grant working operations. In this particular case, having the resulting diff only between 140 kΩ and 149.5 kΩ, the yellow points have been contoured with a black circle, to enable to properly see the plots. (b) Histogram showing the frequency and the diff values (the maximum distance from the expected value in the two memristors). This plot is in *linear scale*. The colour code is explained in figure 4.6.

Being the FELIX XOR an operation made of the consecutive execution of two operations, the plot shows only the voltage and duration of the second operation, i.e., the FELIX NAND operation. A preliminary study has been done on all the control voltages and duration of the pulse for the FELIX OR operation run beforehand, and the chosen one (that was giving us the best results is with V_0=1.94 V and the duration of the pulse is 3.75 ns.

4.5.5 IMPLY

The final state of the IMPLY operation (figure 4.12) does not reach the nominal values, but it is working having the value diff < 5 kΩ. The executed simulations showed that the lower the resistance and the higher the absolute values of V_{SET} and V_{COND}, the lower the value of diff will be. The parameters of the IMPLY simulation have been the load resistance R_G and the two control voltages V_{SET} and V_{COND}. We ran a coarse-grained simulation to find the most suitable R_G and V_{SET}. Finally, the chosen settings are the load resistance R_G of 500 Ω, the voltage V_{SET} of −2 V and the voltage V_{COND} between −1.4 V and −1.275 V. The best result obtained in the simulation with nominal values is with the output memristor around 625 Ω from the expected value (it has been obtained with the control voltages V_{COND} of −1.35 V and V_{SET} of −2 V, a pulse duration of 20 ns, and a load resistance of 500 Ω).



(a) (b)

Figure 4.12: Results of the simulation of the IMPLY operation. (a) Plot showing the couples of control voltage V_0 and duration of the pulse (coloured) that provide working operations. (b) Histogram showing the frequency and the values of diff (the max distance from the expected value in the two memristors). This plot is in *logarithmic scale*.

The colour code is explained in figure 4.6.

Having the IMPLY operation two different control voltages and load resistance, to properly show the graphs we performed several coarse grain simulations and, once we found the most promising load resistance (500 Ω) and V_{SET} (−2 V), we plotted the graphs showing the behaviour of V_{COND}.

4.6 Conclusion

In this chapter we have analysed the electrical behaviour of some significant LIM implementations under various operating conditions, with ideal nominal resistive values. Our results showed that:

- While operations are performed correctly from a logic point of view, the resistance of the output memristor does not always reach its ideal value;
- The higher the distance between nominal and non-ideal state still considered an accepted value, the larger the control voltage range where the LIM operation is considered correct. This knowledge can lessen the restrictions on the voltage regulators (to generate the control voltages);
- Certain operations (e.g., FELIX XOR and IMPLY) may not exhibit a level of reliability that instills confidence. This potential lack of reliability might raise concerns about the effectiveness and consistency of these operations and its usage in multi-operations (consecutive among them without rewriting the input and output values). It could be worth further exploring and evaluating these processes to determine whether enhancements are needed to ensure a more robust and dependable outcome.

5

Electrical Analysis of Boolean Operations with Non-Ideal Memristive Elements

This chapter contains an in depth study of the MAGIC and FELIX LIM solutions based on electrical simulations and assuming non-ideal memristors. Indeed, in the previous chapter, we have shown that, under certain conditions, there are operations for which the output memristor does not reach the ideal resistance value. This observation has raised the following question: what happens when the output of one operation needs to be used as input for an other (i.e., when operations are concatenated). Indeed, LIM solutions such as MAGIC and FELIX have been shown to perform complex logic by concatenating basic operations without the need of storing intermediate results outside the memory array. In this chapter, we show how a non-ideal output of one operation can impact the correctness of the following operation. Indeed, to achieve a seamless cascade of operations as the LIM promises to grant, wherein the outputs of preceding processes serve as inputs for successive ones, it becomes imperative to ascertain the viability of each operation under

non-ideal conditions. Non-ideal conditions that are represented by non-ideal values, i.e., resistive values in one (or more) memristor used to perform the operation that are only close to the nominal one but it does not reach it. In fact, the FELIX NAND operation showed us that we can have both an output value that doesn't reach the expected value (we will refer to it as **non-ideal output**) and, on top of it, there are cases where the input values could be slightly degraded (we will refer to them as **non-ideal inputs**).
Moreover, the FELIX XOR solution is made of a FELIX OR operation followed by a FELIX NAND operation, both executed on the same memristors, reusing the input memristors and the output memristor (without reinitialising it inside the FELIX NAND operation). Before having the results of the simulation of the FELIX XOR (presented in section 4.5), we had supposed that if the FELIX OR had performed similarly to the FELIX NAND, we would have a case of a concatenated operation risking to have both non-ideal inputs and output. And we did not know how the operation could have behaved. However, the FELIX XOR is a particular case because its output memristor, being not initialised, behaves as a third input for the operation (a non-ideal input, as we have demonstrated). In conclusion, we found that it is important to study how the input values can affect the operations and, as a consequence, what are the acceptable resistive values to consider an operation as correct. And to consider an operation correct, we need to distinguish the resistive values thresholds for which the LIM operations under study can be concatenated.

The contributions presented in this chapter are:

- The in-depth analysis of the operating conditions of MAGIC NOT, MAGIC NOR, FELIX OR, FELIX NAND, FELIX XOR and IMPLY with the identification of the ranges in which the operations are performed correctly, highlighting the values of control signals which lead to input-destructive operations, by having these operation fed with non-ideal input values;

- The demonstration that concatenating operations (i.e., using the output of one operation as input of the next operation) starting from non-ideal resistive values can lead to wrong results.

5.1 Simulation environment and preliminary analysis

After the operation is executed, the resistive values of the inputs can be partially overwritten (and therefore be further from the ideal value, or even not correct anymore), and the output value could not reach the wanted value, or reach a non-ideal one.

We analysed the effect of a non-ideal state as input in both NOR and NAND operations. We repeated the preliminary simulations done to study the operations but considering non-ideal states in the input memristors. We have considered two cases: (i) non-ideal states have been chosen from the output resistive values obtained during the NAND operation shown in figure 4.4; (ii) the non-ideal LRS has been chosen from the output resistive values obtained during the NAND operation shown in figure 4.5, while the HRS has been kept in the nominal resistive value. In the first case, non-ideal HRS is equal to 297.5 kΩ and non-ideal LRS is equal to 1225 Ω. In the second case, non-ideal LRS is equal to 5 kΩ.

For the first case, simulations have been performed with non-ideal states in both input memristors and have shown that the operations still work. The NOR and NAND operations can be seen in figures 5.2 and 5.3, respectively. While for the NOR operation there are no remarkable differences, in the case of the NAND operation the working conditions are different from the case where both inputs are initialised with the nominal values. More in particular, the dark green regions in figures 4.5 and 5.3 only partially overlap. This knowledge hardens the restrictions on the voltage regulators (to generate the control voltages), since it reduces the range of voltages for which the operations are correctly executed when considering concatenation.

For the second case, we considered the inputs initialised to the non-ideal LRS of 5 kΩ and nominal HRS. This case is illustrated in the waveform of figure 5.1. In this case, the NOR operation did not work for any combination of voltage amplitude and duration, as seen in figure 5.4. However, the NAND operation worked, with a larger range of voltages, as shown in figure 5.5. Nevertheless, there are no common values of control signals for which the NAND operation is performed correctly when considering nominal and non-ideal states. As a consequence, the operation cannot work.

In this preliminary analysis we have studied the electrical behaviour of the most significant LIM implementations among the ones we selected (MAGIC NOR and FELIX NAND)

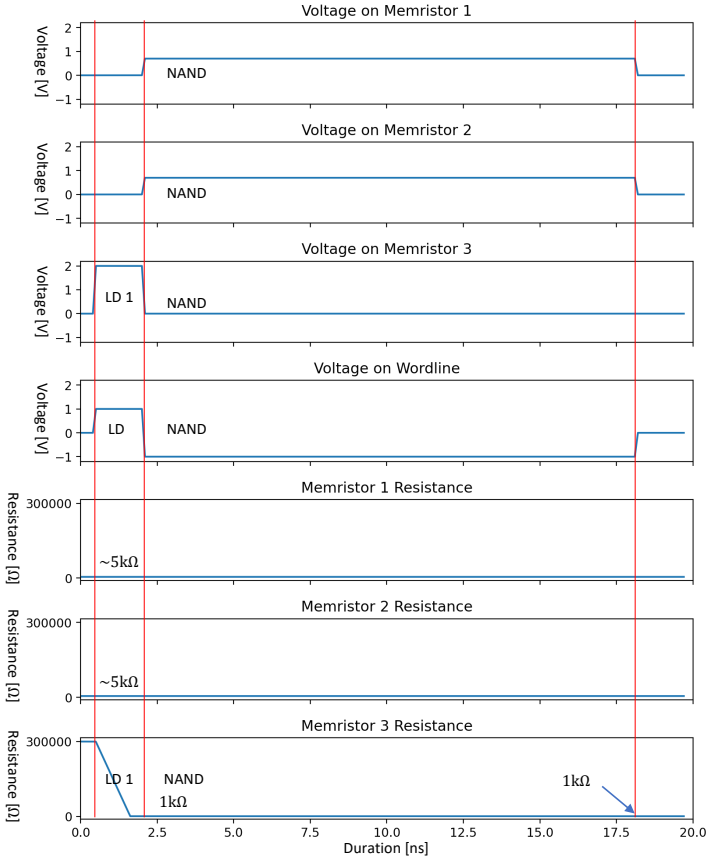


Figure 5.1: Waveform of a FELIX NAND operation having the input memristors with non-ideal LRS=5 kΩ. The vertical red lines indicate the start of the next operation. As it can be seen, instead of the expected logic 1 (300 kΩ), it is obtained as a result a logic 0 (1 kΩ): the memristor cannot change its resistive value due to the too high resistive value of the inputs. The NAND operation is therefore failed.

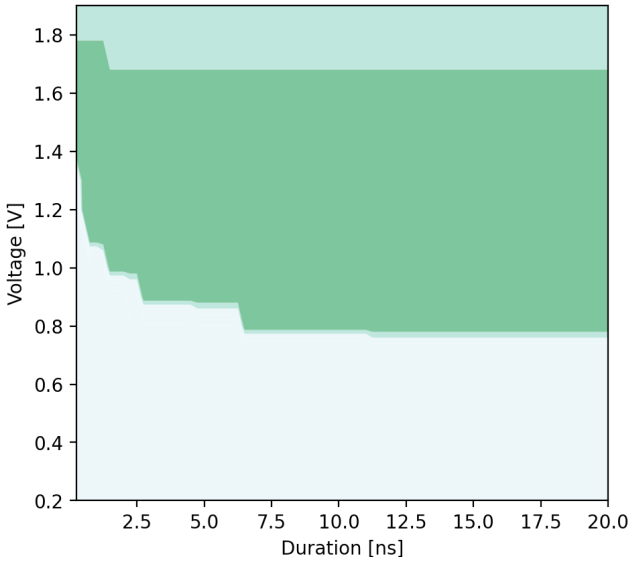


Figure 5.2: Results of the preliminary simulation of the NOR operation with non-ideal state inputs (1225 Ω, 297.5 kΩ) and with non-ideal LRS=1.3 kΩ and non-ideal HRS=290 kΩ for reading the output value. The colour code is explained in section 4.2.

under various operating conditions. Our results showed that concatenating operations (i.e., using the output of one operation as input of the next operation) starting from non-ideal resistive values can lead to wrong results. To overcome this problem, refresh cycles should be added at the end of every logic operation to restore the resistive state to its nominal value (either HRS or LRS) to guarantee the correct result of complex calculations.

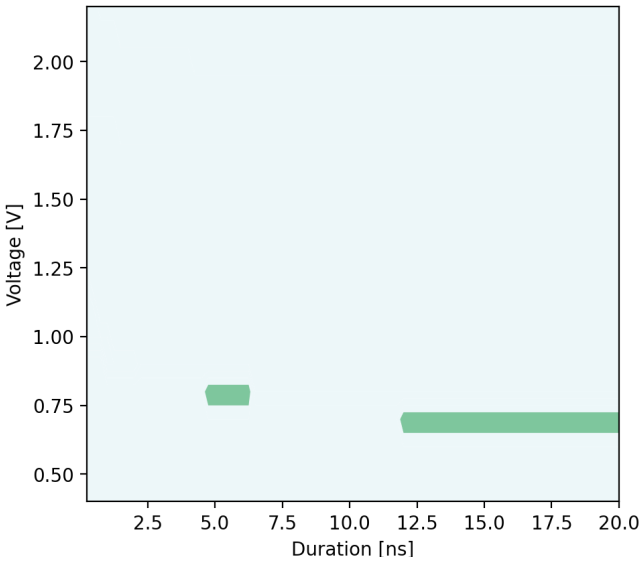


Figure 5.3: Results of the preliminary simulation of the NAND operation with non-ideal state inputs (1225 Ω, 297.5 kΩ) and with non-ideal LRS=1.3 kΩ and non-ideal HRS=290 kΩ for reading the output value. The colour code is explained in section 4.2.

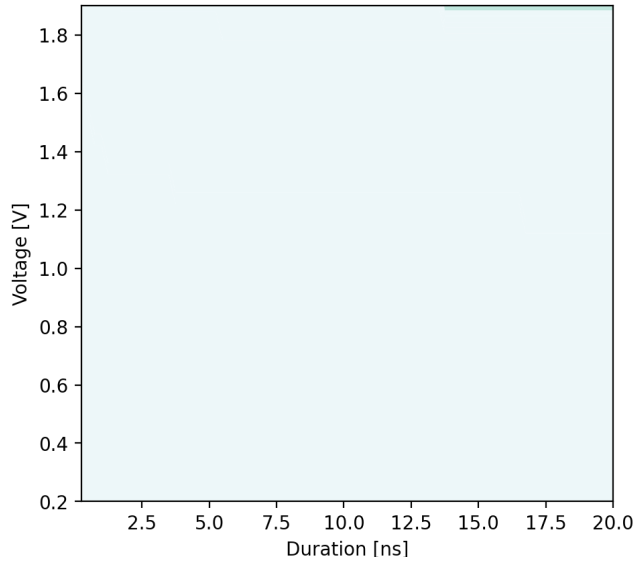


Figure 5.4: Results of the preliminary simulation of the NOR operation with non-ideal state inputs (5 kΩ, 300 kΩ) and with non-ideal LRS=50 kΩ and non-ideal HRS=150 kΩ for reading the output value. The colour code is explained in section 4.2.

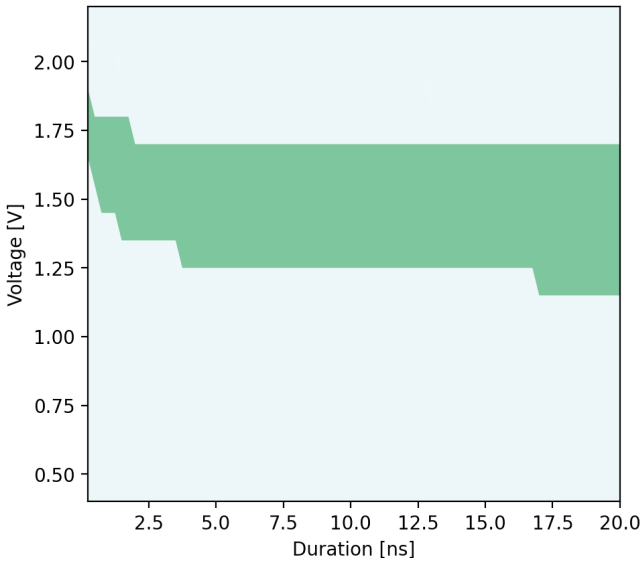


Figure 5.5: Results of the preliminary simulation of the NAND operation with non-ideal state inputs (5 kΩ, 300 kΩ) and with non-ideal LRS=10 kΩ and non-ideal HRS=290 kΩ. for reading the output value. The colour code is explained in section 4.2.

5.2 Parametric Sweep

As for the study of the operations with ideal memristive elements, after analysing the findings of the coarse-grained simulation with non-ideal inputs, we decided that a finer-grained analysis could have provided us with additional insights, even more in circumstances where there appeared to be no working solutions. As a result, we once again used the simulation environment of chapter 3 and simulated the execution of the LIM functions using all input combinations (i.e., 00, 01, 10, 11) saving the final state of the used memristors for each control voltage setup (i.e., couple of amplitude and duration) and for each load resistance (for the IMPLY operation).

We use several thresholds to show how far the final state of the memristors is from the expected one, starting from a minimum distance equal to the half of the HRS - LRS value. We analysed the effect of a non-ideal state as input in all the operations studied. To achieve it, we performed the simulations with the same settings for the nominal input values and on sets of non-ideal inputs. We have considered as inputs three non-ideal cases taken from the final state of the input memristors that we obtained from the simulation of NAND operations:

a) HRS = 299.5 kΩ, LRS = 1025 Ω,

b) HRS = 297.5 kΩ, LRS = 1225 Ω,

c) HRS = 295 kΩ, LRS = 1500 Ω.

These values have been used for the input memristors (in_1 for MAGIC NOT, in_1 and in_2 for MAGIC NOR, FELIX NAND, OR and XOR and in_1 and $in_2 out$ for IMPLY).

For each control voltage setup (i.e., couple of amplitude and duration) and for each load resistance (for the IMPLY operation), we have simulated the execution of the LIM functions applying all input combinations (i.e., 00, 01, 10, 11) and saving the final state of the used memristors. Among the data extracted from the analysis of the results there are 1) the inputs correctness, i.e., the inputs remained the ones that have been written before executing the operation, 2) the output correctness, i.e., the output state is the expected one,

3) the difference from the expected value for each memristive value, calculated doing:

$$diff = \left| out_{expected} - out_{read} \right| \tag{5.1}$$

We calculated the maximum diff for each set of amplitude and duration of the control voltage (and also load resistance, for IMPLY): this allows us to take the worst case scenario for all the input combinations. These cumulative data allowed us to pinpoint the settings that were the most suitable for each operation. Using these data, we plotted graphs indicating for which settings an operation was working, and, if so, how close the read memristive values were to the nominal expected values. To do so, five different colours (see figure 5.6) have been used in the following graphs to indicate five thresholds, based on the diff explained in equation (5.1):

- Purple: the read values are the expected nominal resistive values (with a tolerance of 50 Ω),
- Dark blue: diff is less than 5 kΩ,
- Light blue: diff is less than 10 kΩ,
- Green: diff is less than 50 kΩ,
- Yellow: diff is less than 149.5 kΩ.

The blank part in the scatter plots is where the operation didn't work: diff is greater than $(HRS - LRS)/2$, thus not allowing a clear distinction between the two states.

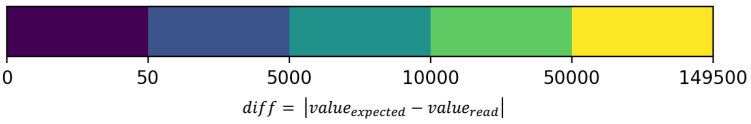


Figure 5.6: Key for the plots.

These operations we simulated use two or three memristors: in_1, (in_2), and out or, in the case of IMPLY, in_1 and $in_2 out$. in_1 and in_2 are the input memristors, $in_2 out$ is the input/output memristor used by the IMPLY operation, and out is the output memristor.

5.2.1 MAGIC NOT and NOR

In the cases with non-ideal inputs, we can observe a minor and gradual reduction of the working area made of the couples of control voltage and spike duration for both the operations, that gets smaller together with the increase of the non-ideality of the inputs. More in detail, there is a slight increase of the voltage needed to have the operation working. In fact, the higher resistive value of the LRS increases the lower voltage bound, while the lower resistive value of the HRS decreases the upper bound (this is the case where in_1 = 0 for NOT and $in_1 = in_2 = 0$ for NOR), because it allows to change the resistive value of the inputs. Concluding, the two operations could be used in an implementation that takes advantage of consecutive operations. The vast working area and the ability to reach the expected values even with non-ideal inputs, makes of them robust choices for LIM.

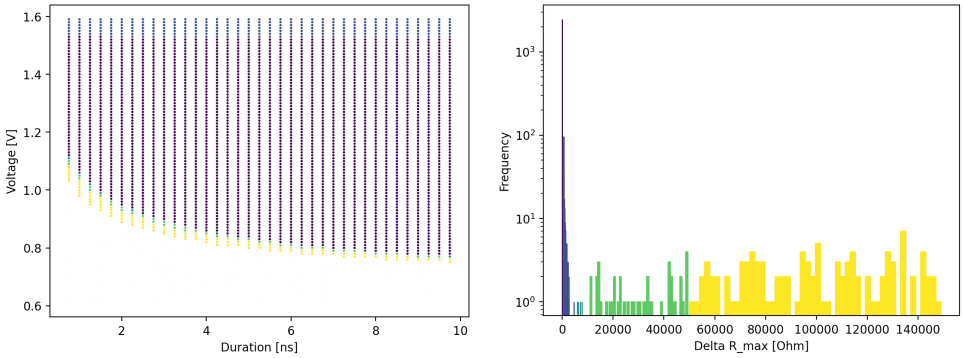
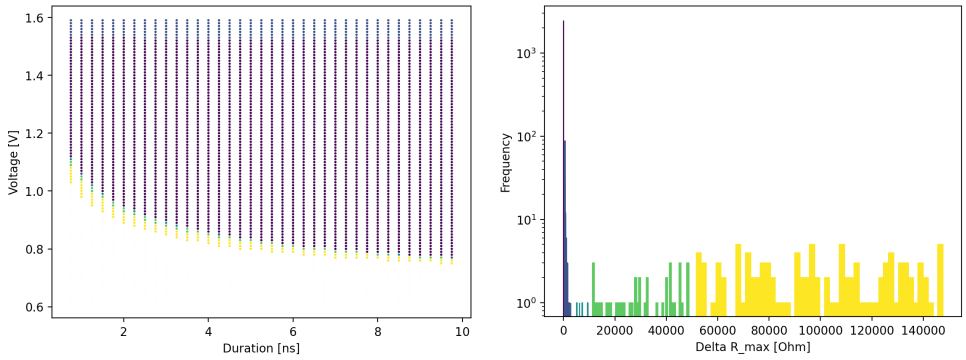
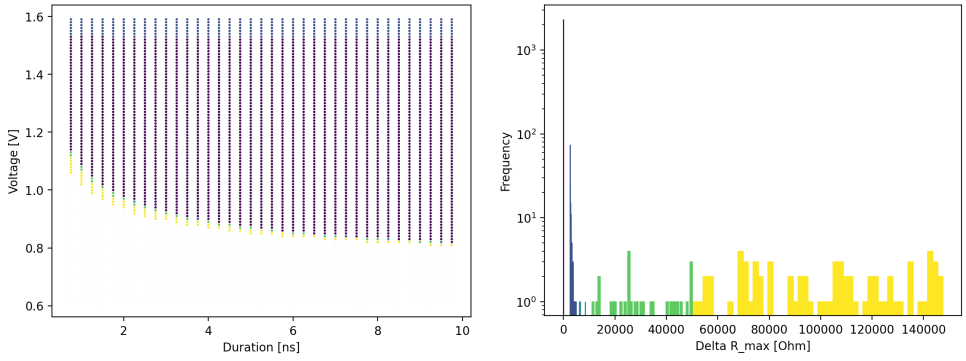


Figure 5.7: Results of the simulation of the MAGIC NOT operation with non-ideal input values (HRS = 299.5 kΩ, LRS = 1025 Ω). (a) Plot showing the couples of control voltage V_0 and duration of the pulse (coloured) that provide working operations. (b) Histogram showing the frequency and the diff values (the maximum distance from the expected value in the two memristors). This plot is in *logarithmic scale*.
The colour code is explained in figure 4.6.



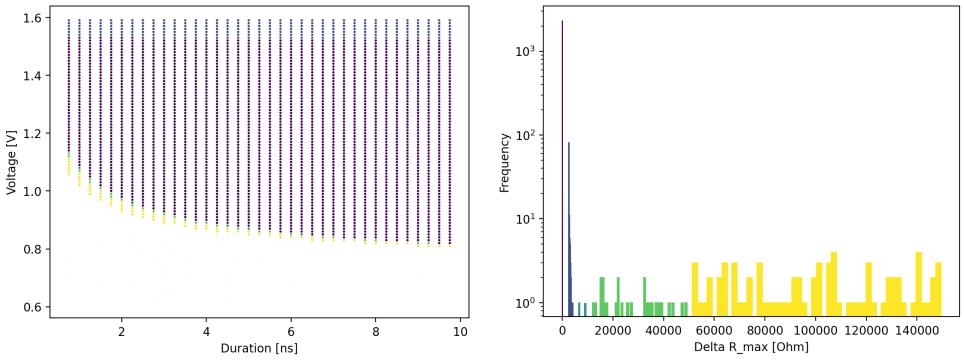
(a) (b)

Figure 5.8: Results of the simulation of the MAGIC NOR operation with non-ideal input values (HRS = 299.5 kΩ, LRS = 1025 Ω). (a) Plot showing the couples of control voltage V_0 and duration of the pulse (coloured) that provide working operations. (b) Histogram showing the frequency and the values of diff (the max distance from the expected value in the two memristors). This plot is in *logarithmic scale*.
The colour code is explained in figure 4.6.



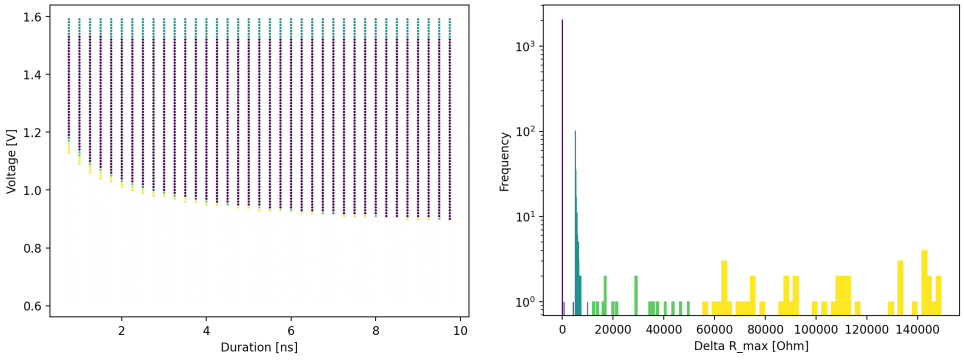
(a) (b)

Figure 5.9: Results of the simulation of the MAGIC NOT operation with non-ideal input values (HRS = 299.5 kΩ, LRS = 1025 Ω). (a) Plot showing the couples of control voltage V_0 and duration of the pulse (coloured) that provide working operations. (b) Histogram showing the frequency and the diff values (the maximum distance from the expected value in the two memristors). This plot is in *logarithmic scale*.
The colour code is explained in figure 4.6.



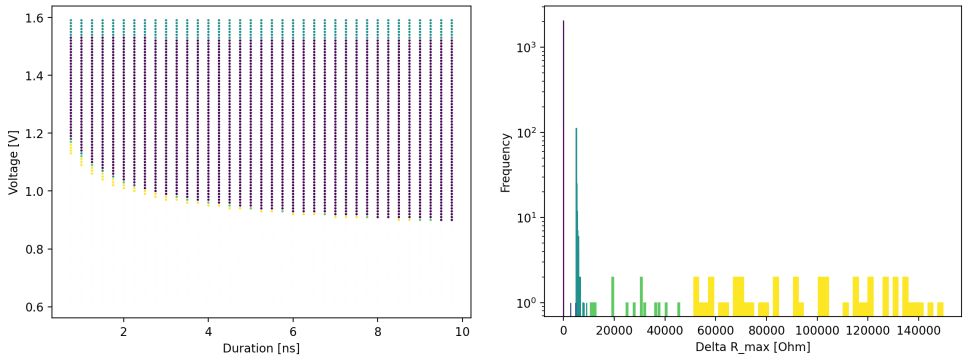
(a) (b)

Figure 5.10: Results of the simulation of the MAGIC NOR operation with non-ideal input values (HRS = 299.5 kΩ, LRS = 1025 Ω). (a) Plot showing the couples of control voltage V_0 and duration of the pulse (coloured) that provide working operations. (b) Histogram showing the frequency and the values of diff (the max distance from the expected value in the two memristors). This plot is in *logarithmic scale*.
The colour code is explained in figure 4.6.



(a) (b)

Figure 5.11: Results of the simulation of the MAGIC NOT operation with non-ideal input values (HRS = 299.5 kΩ, LRS = 1025 Ω). (a) Plot showing the couples of control voltage V_0 and duration of the pulse (coloured) that provide working operations. (b) Histogram showing the frequency and the diff values (the maximum distance from the expected value in the two memristors). This plot is in *logarithmic scale*.
The colour code is explained in figure 4.6.



(a) (b)

Figure 5.12: Results of the simulation of the MAGIC NOR operation with non-ideal input values (HRS = 299.5 kΩ, LRS = 1025 Ω). (a) Plot showing the couples of control voltage V_0 and duration of the pulse (coloured) that provide working operations. (b) Histogram showing the frequency and the values of diff (the max distance from the expected value in the two memristors). This plot is in *logarithmic scale*.
The colour code is explained in figure 4.6.

5.2.2 FELIX NAND

With lesser ideal input values, the needed voltage slightly increases (for the same reasons of the MAGIC NOR and NOT operations), but, on the other hand, the upper bound increases, allowing a greater overall working area. However, the overlap of the areas with ideal results of each input case figures 4.9 and 5.13 to 5.15 (plots a) is null. In fact, the overlap of the input cases figures 4.9, 5.13 and 5.14 gives only 6 purple points that are common in the three plots. This hardens the restrictions on the voltage regulators (to generate the control voltages allowing a correct operation), since it reduces even more the range of voltages for which the operations are correctly executed when considering concatenation. By accepting non-ideal result values, there is a wider area, but it would compromise the concatenation of the operation. The FELIX NAND operation, assuming that the control voltage could be precise enough to enable the operation, could be used with consecutive operations under strict input constraints.

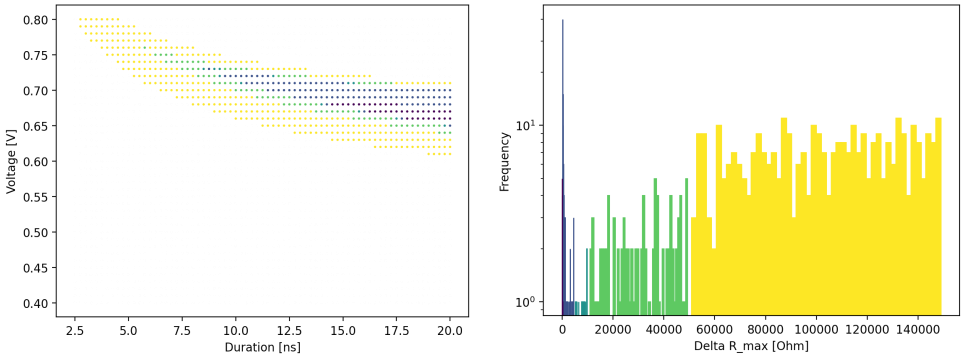
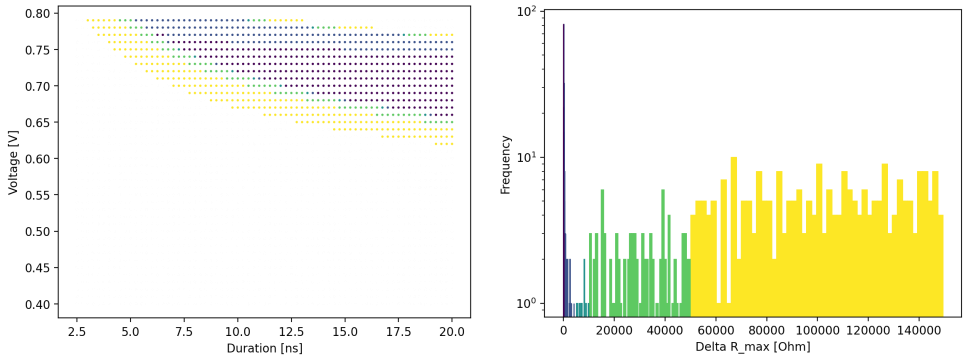


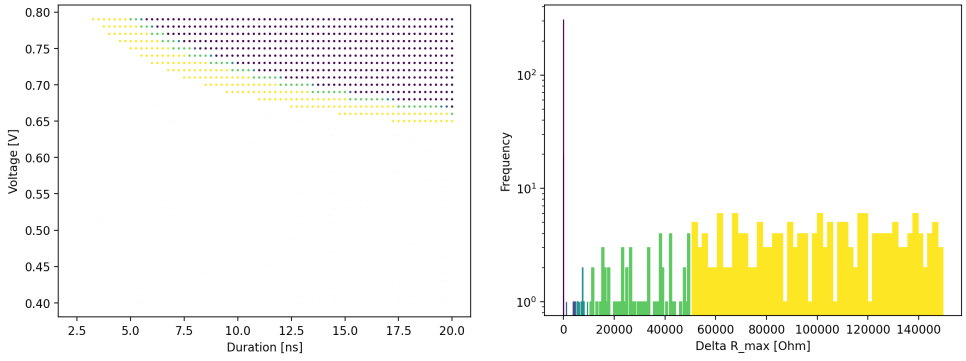
Figure 5.13: Results of the simulation of the FELIX NAND operation with non-ideal input values (HRS = 299.5 kΩ, LRS = 1025 Ω). (a) Plot showing the couples of control voltage V_0 and duration of the pulse (coloured) that provide working operations. (b) Histogram showing the frequency and the values of diff (the max distance from the expected value in the two memristors). This plot is in *logarithmic scale*.
The colour code is explained in figure 4.6.



(a) (b)

Figure 5.14: Results of the simulation of the FELIX NAND operation. (a) Plot showing the couples of control voltage V_0 and duration of the pulse (coloured) that provide working operations. (b) Histogram showing the frequency and the values of diff (the max distance from the expected value in the two memristors). This plot is in *logarithmic scale*.

The colour code is explained in figure 4.6.



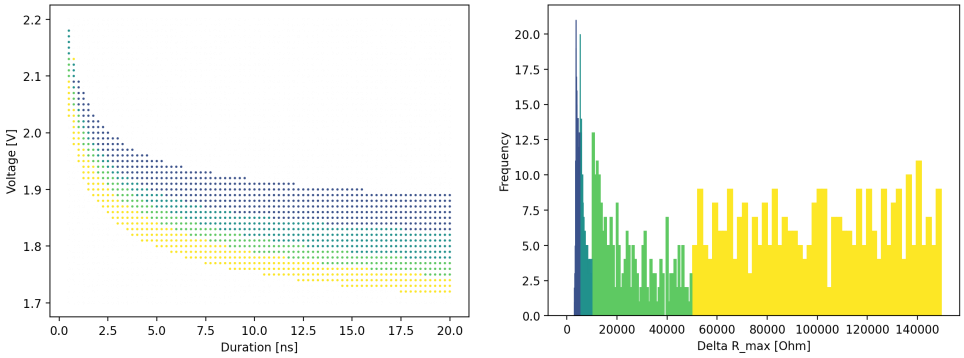
(a) (b)

Figure 5.15: Results of the simulation of the FELIX NAND operation. (a) Plot showing the couples of control voltage V_0 and duration of the pulse (coloured) that provide working operations. (b) Histogram showing the frequency and the values of diff (the max distance from the expected value in the two memristors). This plot is in *logarithmic scale*.

The colour code is explained in figure 4.6.

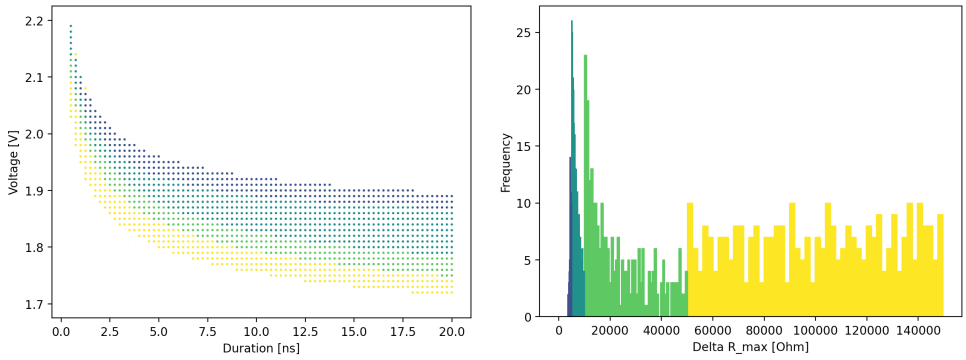
5.2.3 FELIX OR

The behaviour of the operation for non-ideal input cases worsens with respect to the case with nominal inputs: the working area from the combination of the voltage and the operation duration gets smaller and the values reached are less close to the nominal values. As it can be seen in the histograms in figures 4.10 and 5.16 to 5.18 (cases b), the best diff values reached are lower and lower, having only few results in the case with diff < 5 kΩ. While the FELIX OR is resistant enough to correctly perform the operation with the non-ideal inputs we used, the result would not be enough close to the nominal value to be used as input for other operations. To allow a proper concatenation with the FELIX OR, we would need a refresh of at least the output memristor, being it the one showing the highest drift from the nominal expected value.



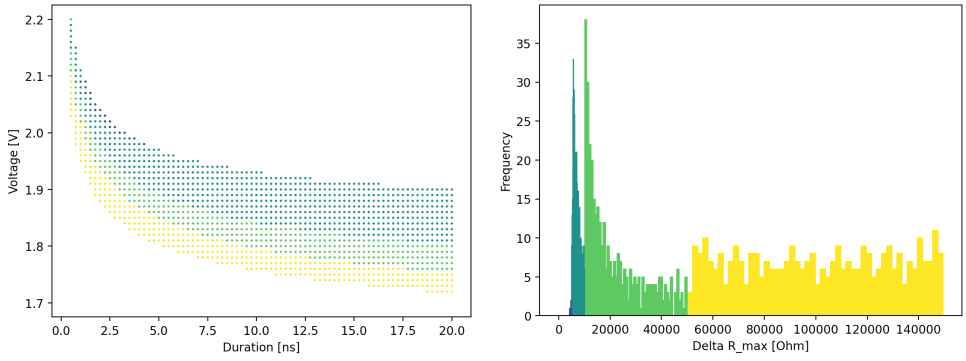
(a) (b)

Figure 5.16: Results of the simulation of the FELIX OR operation with non-ideal input values (HRS = 299.5 kΩ, LRS = 1025 Ω). (a) Plot showing the couples of control voltage V_0 and duration of the pulse (coloured) that provide working operations. (b) Histogram showing the frequency and the values of diff (the max distance from the expected value in the two memristors). This plot is in *linear scale*.
The colour code is explained in figure 4.6.



(a) (b)

Figure 5.17: Results of the simulation of the FELIX OR operation with non-ideal input values (HRS = 297.5 kΩ, LRS = 1225 Ω). (a) Plot showing the couples of control voltage V_0 and duration of the pulse (coloured) that provide working operations. (b) Histogram showing the frequency and the values of diff (the max distance from the expected value in the two memristors). This plot is in *linear scale*.
The colour code is explained in figure 4.6.

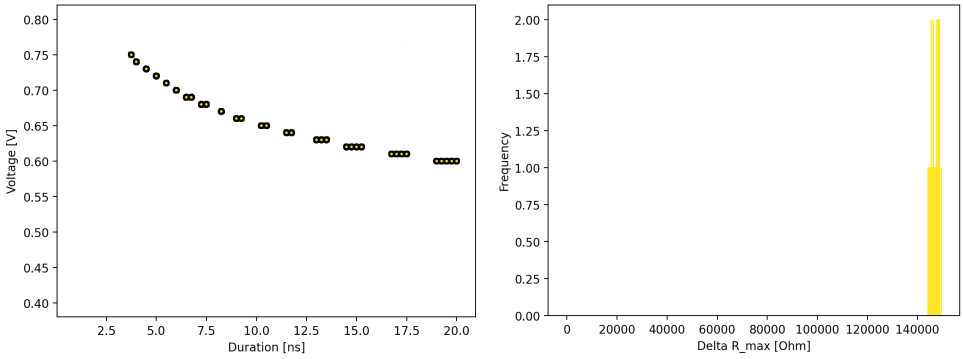


(a) (b)

Figure 5.18: Results of the simulation of the FELIX OR operation with non-ideal input values (HRS = 295 kΩ, LRS = 1500 Ω). (a) Plot showing the couples of control voltage V_0 and duration of the pulse (coloured) that provide working operations. (b) Histogram showing the frequency and the values of diff (the max distance from the expected value in the two memristors). This plot is in *linear scale*.
The colour code is explained in figure 4.6.

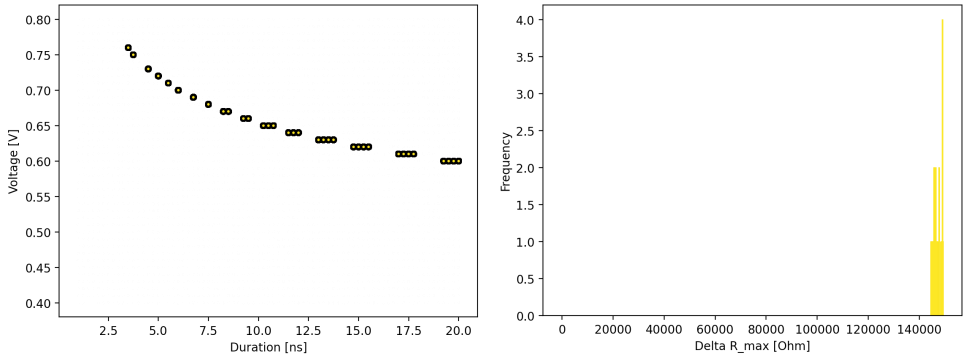
5.2.4 FELIX XOR

Starting from the considerations on the FELIX XOR operations we treated in the last chapter, this is an operation that we do not consider actually usable with the use constraints that we have put. We performed anyway the simulations to see how it could have behaved with non ideal cases. The results are similar to the ones with ideal nominal input values. The diff is slightly higher, but there are common values between the 4 input cases, showing us that it can tolerate less ideal cases. But the diff value is too high, and therefore, as assessed before, the FELIX XOR operation would be very difficult to implement and to use in a concatenated manner: the very strict input voltage constraints and the final resistive values of the output, which would surely need a refresh represent a huge obstacle.



(a) (b)

Figure 5.19: Results of the simulation of the FELIX XOR operation with non-ideal input values (HRS = 299.5 kΩ, LRS = 1025 Ω). In this case, only the plot for the FELIX NAND operation is shown (being it the second step of the FELIX XOR operation): the best couple of voltage and duration of the FELIX OR operation that brought to the case shown here has already been calculated. (a) Plot showing the control voltage V_0 and the duration of the pulse couples (coloured) that grant working operations. In this particular case, having the resulting diff only between 140 kΩ and 149.5 kΩ, the yellow points have been contoured with a black circle, to enable to properly see the plots. (b) Histogram showing the frequency and the diff values (the maximum distance from the expected value in the two memristors). This plot is in *linear scale*.
The colour code is explained in figure 4.6.
Being the FELIX XOR an operation made of the consecutive execution of two operations, the plot shows only the voltage and duration of the second operation, i.e., the FELIX NAND operation. A preliminary study has been done on all the control voltages and duration of the pulse for the FELIX OR operation run beforehand, and the chosen one (that was giving us the best results is with V_0=1.94 V and the duration of the pulse is 3.75 ns.

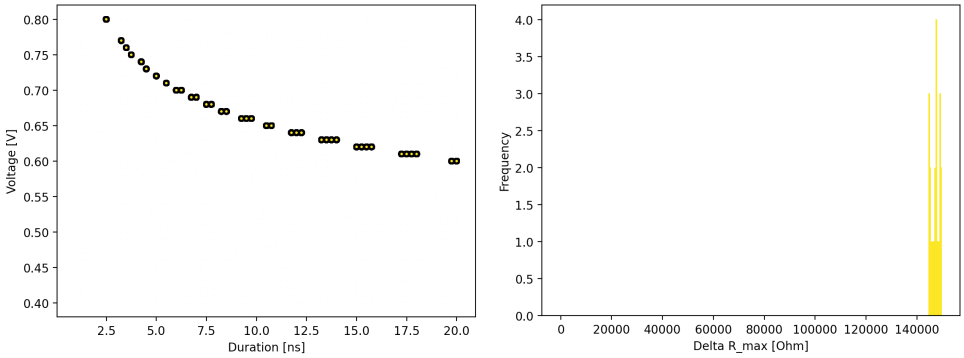


(a) (b)

Figure 5.20: Results of the simulation of the FELIX XOR operation with non-ideal input values (HRS = 297.5 kΩ, LRS = 1225 Ω). In this case, only the plot for the FELIX NAND operation is shown (being it the second step of the FELIX XOR operation): the best couple of voltage and duration of the FELIX OR operation that brought to the case shown here has already been calculated. (a) Plot showing the control voltage V_0 and the duration of the pulse couples (coloured) that grant working operations. In this particular case, having the resulting diff only between 140 kΩ and 149.5 kΩ, the yellow points have been contoured with a black circle, to enable to properly see the plots. (b) Histogram showing the frequency and the diff values (the maximum distance from the expected value in the two memristors). This plot is in *linear scale*.

The colour code is explained in figure 4.6.

Being the FELIX XOR an operation made of the consecutive execution of two operations, the plot shows only the voltage and duration of the second operation, i.e., the FELIX NAND operation. A preliminary study has been done on all the control voltages and duration of the pulse for the FELIX OR operation run beforehand, and the chosen one (that was giving us the best results is with V_0=1.94 V and the duration of the pulse is 3.75 ns.



(a) (b)

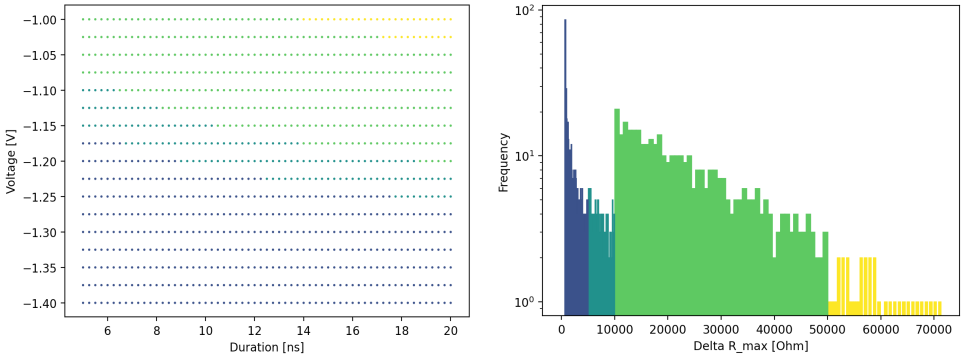
Figure 5.21: Results of the simulation of the FELIX XOR operation with non-ideal input values (HRS = 295 kΩ, LRS = 1500 Ω). In this case, only the plot for the FELIX NAND operation is shown (being it the second step of the FELIX XOR operation): the best couple of voltage and duration of the FELIX OR operation that brought to the case shown here has already been calculated. (a) Plot showing the control voltage V_0 and the duration of the pulse couples (coloured) that grant working operations. In this particular case, having the resulting diff only between 140 kΩ and 149.5 kΩ, the yellow points have been contoured with a black circle, to enable to properly see the plots. (b) Histogram showing the frequency and the diff values (the maximum distance from the expected value in the two memristors). This plot is in *linear scale*.

The colour code is explained in figure 4.6.

Being the FELIX XOR an operation made of the consecutive execution of two operations, the plot shows only the voltage and duration of the second operation, i.e., the FELIX NAND operation. A preliminary study has been done on all the control voltages and duration of the pulse for the FELIX OR operation run beforehand, and the chosen one (that was giving us the best results is with V_0=1.94 V and the duration of the pulse is 3.75 ns.

5.2.5 IMPLY

The behaviour of the IMPLY operation under non-ideal input condition worsens with respect to the one we found with nominal input values: as it can be seen by the histograms in figures 4.12 and 5.22 to 5.24, the further the input values are from the nominal ones, the further is the final state of the memristors from the expected one. There is a common area between the four input cases, having the diff value in the worst case between 10 kΩ and 50 kΩ. The behaviour of the IMPLY operation is not close to what we are looking for in terms of functioning and reliability, therefore we chose it would not be able to be used for our application and we excluded it.

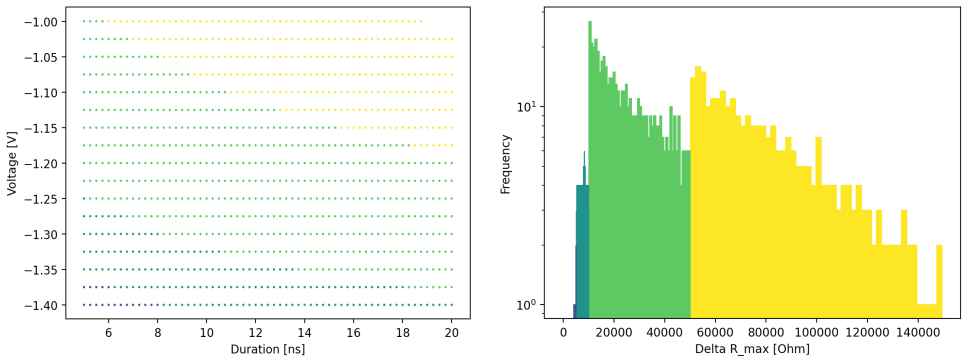


(a) (b)

Figure 5.22: Results of the simulation of the IMPLY operation with non-ideal input values (HRS = 299.5 kΩ, LRS = 1025 Ω). (a) Plot showing the couples of control voltage V_0 and duration of the pulse (coloured) that provide working operations. (b) Histogram showing the frequency and the values of diff (the max distance from the expected value in the two memristors). This plot is in *logarithmic scale*.

The colour code is explained in figure 4.6.

Having the IMPLY operation two different control voltages and load resistance, to properly show the graphs we performed several coarse grain simulations and, once we found the most promising load resistance (500 Ω) and V_{SET} (−2 V), we plotted the graphs showing the behaviour of V_{COND}.

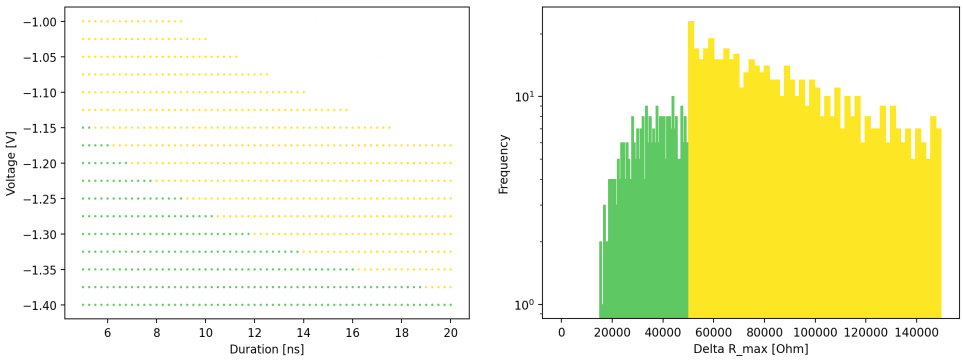


(a) (b)

Figure 5.23: Results of the simulation of the IMPLY operation with non-ideal input values (HRS = 297.5 kΩ, LRS = 1225 Ω). (a) Plot showing the couples of control voltage V_0 and duration of the pulse (coloured) that provide working operations. (b) Histogram showing the frequency and the values of diff (the max distance from the expected value in the two memristors). This plot is in *logarithmic scale*.

The colour code is explained in figure 4.6.

Having the IMPLY operation two different control voltages and load resistance, to properly show the graphs we performed several coarse grain simulations and, once we found the most promising load resistance (500 Ω) and V_{SET} (−2 V), we plotted the graphs showing the behaviour of V_{COND}.



(a) (b)

Figure 5.24: Results of the simulation of the IMPLY operation with non-ideal input values (HRS = 295 kΩ, LRS = 1500 Ω). (a) Plot showing the couples of control voltage V_0 and duration of the pulse (coloured) that provide working operations. (b) Histogram showing the frequency and the values of diff (the max distance from the expected value in the two memristors). This plot is in *logarithmic scale*.

The colour code is explained in figure 4.6.

Having the IMPLY operation two different control voltages and load resistance, to properly show the graphs we performed several coarse grain simulations and, once we found the most promising load resistance (500 Ω) and V_{SET} (−2 V), we plotted the graphs showing the behaviour of V_{COND}.

5.3 Discussion and Conclusion

In the previous and this chapter, we have simulated and analysed the electrical behaviour of selected LIM solutions (MAGIC NOT, MAGIC NOR, FELIX NAND, FELIX OR, FELIX XOR and IMPLY), under both ideal and non-ideal input conditions. We have shown that there are several differences between them.

5.3.1 Comparison among XOR implementations

This section aims to discuss the main aspects of the implementations of the XOR operation using the different LIM solutions. The XOR operation is the main step to perform the most common security operations, such as encryption and decryption operations. We focus therefore to compare the different implementation-based XOR operations according to several considerations described below.

Area We will consider the area occupation as the number of memristors needed to perform the XOR operation. As can be seen in tables 4.5, 4.7 and 4.9, the solution that takes the least number of memristors is the FELIX XOR, since it only requires the two input and the output memristors. The others require two additional functional memristor to enable intermediate operations that allow the operation to reach the result.

Speed Regarding the speed, in table 5.1 are presented the minimum durations of the selected operations, with the strictest threshold (diff < 50 Ω) to consider them as correct. As can be seen, the FELIX NAND and IMPLY operations take more than 15 ns, while the MAGIC NOT and NOR operations are able to work correctly in 0.5 ns. Therefore, the MAGIC-based XOR operation, even if it has more steps than the FELIX XOR, is the fastest.

Robustness of the operation We consider an operation as robust if it does not have a non-negligible effect on the inputs and it allows to reach a nominal value in the output. According to the simulations performed, the MAGIC-based XOR is again the most robust XOR operation.

Operation	Voltage(s) [V]	t (control voltage pulse) [ns]	t (chosen) [ns]	t (full operation) [ns]	diff [Ω]	Chosen voltage(s) [V]
Write	2.3 / -1.5	0.25+	0.25	0.25	0	2.3 / -1.5
IMPLY	1.35 / 2	0.25+	20	20.25	627	1.35 / 2
MAGIC NOT	1.36 ÷ 1.54	0.5+	0.25	0.5	0	1.36 ÷ 1.54
MAGIC NOR	1.2 ÷ 1.53	0.25+	0.25	0.5	0	1.2 ÷ 1.53
FELIX NAND	0.66, 0.67	16 ÷ 19	16	16.25	38	0.67
FELIX OR	2.03	1.75 ÷ 2.5	1.75	2	2753	2.03
FELIX XOR	1.94 / 0.61	3.75 / 17	20.75	21	144.5k	-
MAGIC XOR	-		-	2.5	0	
IMPLY XOR	-		-	162	-	

Table 5.1: Table showing the operation time, the possible input ranges and the chosen inputs for the analysed operations.

Operation range Among the operations that we analysed, the ones with the widest operating range are the MAGIC NOT and NOR operations. This wide range allows to be realistically implemented from the voltage regulators and the architectural aspects.

5.3.2 Conclusion

The main conclusions of this chapter are:

- The resistive value of the input memristor has an impact on the ranges in which the operations are performed correctly;
- As a consequence from the last point, concatenating operations (i.e., using the output of the previous operation as input of the new operation) starting from non-ideal values can lead to wrong results. To overcome this issue, refresh cycles should be added at the end of every operation that does not allow to reach nominal values in the output memristor. This would allow to have again the nominal values (HRS or LRS) and allow the correct execution of the successive operations;
- Complex operations such as the XOR operation can be more interesting when executed taking into account its robustness and speed instead of a reduced area and a lower number of steps. The MAGIC-based XOR operation results faster (even if composed of 10 steps) and more robust (it allows to have nominal results avoiding a change in the input memristors) than the other alternatives we explored.

The main lesson learnt is that some operations are not able to guarantee correct functioning when run consecutively. As a way to mitigate this problem, we would need to read and rewrite (refresh) the memristor's values. This could be done when needed, by developing a current sensor that checks the correct value of the memristor and, in case it is needed, refreshes the value. Other operations (like the NAND operation) are more resilient to the non-ideal values, and they could avoid this problem. Another option could be to have the *compiler* handling the issue: we are able to know which are the operations that are delicate, and scheduling changes, together with aimed refreshes could be a better solution.

Variability is another important issue that has to be taken into account: it could allow similar non-ideal values as we showed in this chapter, changing the behaviour of the operations. More studies are needed on variability and maturity of technology.

6

Security Aspects of LIM Solutions

Besides the need for power efficiency and computation speed, the need for security has also become increasingly important. This has led to the development of hardware components and Intellectual Properties (IPs) for cryptography, but it has also created new types of threats and hardware attacks, such as: side-channel attacks, which exploit information leaked through a device's physical characteristics, such as power consumption or electromagnetic emissions; fault injection attacks, which aim to introduce faults into a device's hardware in order to disrupt its normal operation or extract sensitive information. In classical architectures, data and cryptographic keys are stored in the main memory and transferred to the processor to execute the cryptographic functions. Therefore, confidential information transits unencrypted via the communication buses, being susceptible to information leakage. On the contrary, with CIM, secure operations can be performed without resorting to data transfer, therefore mitigating the risk of data leakage and avoiding exposure to attacks. Among the many CIM solutions, considering the previous work introduced in the previous chapter of this thesis, we focus on LIM based on MAGIC [27] which is able to perform any logic operation within the memory array.

In this chapter we show that:

- MAGIC-based operations used to perform the MAGIC-based XOR operation have a power consumption profile which is data-dependent, thus enabling side-channel attacks;

- Memristive memory arrays are very sensitive to variations of electrical operation conditions, and thus prone to fault attacks.

With respect to the state of the art, side-channel analysis has already been used to reverse-engineer the functional structure of IPs implemented with MAGIC in a memristive array [147], whereas [148] performs side-channel and differential power analyses on another type of LIM implementation, i.e., Complementary Resistive Switching (CRS). Moreover, [149] takes advantage of side-channel attacks on STT-MRAM-based cache, being it an interesting application on emerging memories.

6

6.1 MAGIC-based XOR

There are different LIM solutions described in the literature, such as MAGIC [27], FELIX [110], and IMPLY [28], which implement some basic logic functions: NOT and NOR for MAGIC; NAND and OR for FELIX and implication for IMPLY. The IMPLY solution has the disadvantage of being input-destructive (i.e., the inputs are not preserved after the operation is executed), making the reuse of data difficult. The FELIX is not a robust solution, as demonstrated in the previous chapters. In contrast, MAGIC solution is robust and non-input destructive, therefore suitable for the implementation of complex logic. Moreover, MAGIC (i.e. NOT and NOR operations) is not sensitive to dynamic and static memristive variability (cycle-to-cycle and device-to-device), while it is sensitive to variations in control signals, as shown in chapter 5. The MAGIC NOR operation requires three memristors: two in parallel as input values (in_1 and in_2) and the third, in series, for the output (out). The logic operations are carried out by first setting the output memristor to logic 1 (LRS), then providing a voltage V_0 to the resistive structure. The output memristor will switch to logic 0 (HRS) if its voltage drop is large enough, which depends on the input values. This switch occurs when at least one of the input memristors is at logic 1, thus performing a NOR operation. The MAGIC NOT operates in the same way, but only with one input

memristor. The analysis we performed in this chapter is based on simulation, with the simulation environment explained in chapter 3. Parameters, duration and control voltage values for the MAGIC NOT and NOR have been selected according to the previous results: R_{OFF} (HRS, logic 0) = 300 kΩ; R_{ON} (LRS, logic 1) = 1 kΩ; a cycle time of 0.25 ns; a control voltage V_0 of 1.4 V.

The goal of this study is the analysis of the sensitiveness to side-channel and fault injections on MAGIC-based LIM implementations in the context of secure applications. The most significant logic operation for cryptography is the XOR operation (commonly used in all encryption algorithms, including DES [143], AES [145], and PRESENT [146]), which secures the plaintext by combining it with the secret key. Therefore, we have chosen to focus our analysis on the XOR gate, as we believe that if this operation is not secure, the overall algorithm is not secure either. In order to implement XOR with MAGIC, it is necessary to concatenate multiple NOR and NOT operations by obtaining a sequence of five steps (as fully described in section 4.4.1): 1) MAGIC NOT (in_1, f_1), 2) MAGIC NOT(in_2, out), 3) MAGIC NOR(f_1, out, f_2), 4) MAGIC NOR(in_1, in_2, f_1) and 5) MAGIC NOR(f_1, f_2, out). These steps involve the use of five memristors: in_1 and in_2 as input memristors, f_1 and f_2 as functional memristors used to store temporary results, and out as the output memristor where the final result of the operation is written. Figure 6.1 shows the schematic we implemented to perform our simulations. V_{set} and V_{reset} are used to initialise memristors to the desired logic value, and V_0 is used to perform the NOR and NOT logic operations.

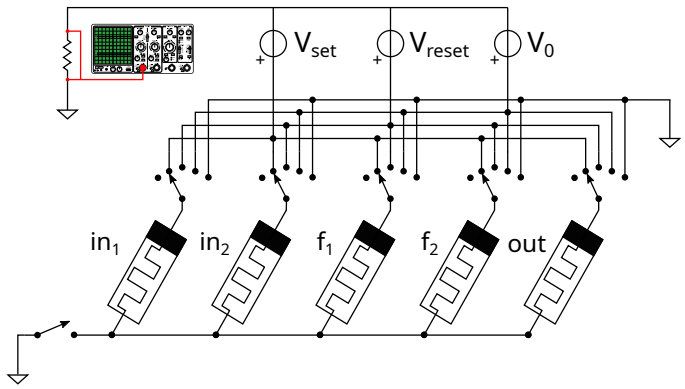


Figure 6.1: MAGIC-based XOR netlist.

Inputs	Step 1 f_1 = NOT(in_1)	Step 2 out = NOT(in_2)	Step 3 f_2 = NOR(out,f_1)	Step 4 f_1 = NOR(in_1,in_2)	Step 5 out = NOR(f_1,f_2)
00	1	1	0	1	0
01	1	0	0	0	1
10	0	1	0	0	1
11	0	0	1	0	0

Table 6.1: MAGIC-based XOR steps and values. Under the steps are the resulting values of each operation.

6.2 Side Channel Attacks on MAGIC-based XOR

Side-Channel Attacks exploit the fact that secure devices leak physical information during data processing. This physical leakage (e.g., power dissipation, electromagnetic emanation, timing information) can be measured externally and used for compromising confidential data, such as the secret key of a cryptographic system. Side-channel attacks such as SPA and DPA have become popular since, without proper countermeasures, they require the knowledge of the algorithm, a model correlating the physical measurements and the processed data, but not the physical implementation of the target device. On classical CMOS-based circuits, DPA exploits the fact that transitions (from 0 to 1 or from 1 to 0) of the logic gates require energy (which can be measured via an oscilloscope). On the other side, without transitions, the gate's transistors only have static power consumption. Therefore, by measuring the current consumed by the circuit, it is possible to create a correlation with internal circuit's transitions. The most common information leakage models are the hamming distance and hamming weight. The Hamming distance model assumes that the power consumption of a device is correlated with the number of bits that change between two input states. The Hamming weight model assumes that the power consumption of a device is correlated with the number of bits set to 1 in the input data. On the contrary, in resistive-based circuits, we observe large variations in currents consumed by the circuit, with or without state transitions. In this thesis, we investigate how side-channel analysis can be performed based on this principle.

6.2.1 Current consumption of MAGIC-based XOR

In order to create a proper information leakage model for the memristive-based LIM operations, we simulated the MAGIC-based XOR operation while measuring the corresponding current profile, for all input combinations, as shown in figure 6.2. It should be noticed that the XOR is obtained by concatenating the 5 operations (as shown in table 6.1) and each operation is performed in two steps, i.e., the SET of the output memristor, followed by the actual NOR (or NOT) operation involving the inputs. Some particularities of the current behaviour should be noticed:

- A current spike is observed every time the output memristor changes its state from the preset value (i.e., switches from 1 to 0). The positions of the current spikes reflect the truth tables in table 6.1. In addition, the spike amplitude is at least 3 orders of magnitude larger than what would be observed in a classical CMOS gate.

- The energy consumed during the SET operation depends on the initial state of the memristor. Indeed, if the initial state is LRS, the SET operation does not change the state of the memristor, but has high energy consumption. If the initial state is HRS, the SET operation changes the state of the memristor but its energy consumption its very low before the switch happens. In operations 1, 2 and 3, the initial states of memristors f_1, f_2, and out are assumed unknown (and in any case not related to the input values – for simplicity, in the simulation we assumed their initial state to HRS), while in operations 4 and 5, the initial states of memristors f_1 and out depend on the input values. Indeed, before the SET of f_1 in operation 4, the state of f_1 is the result of operation 1, i.e., $not(in_1)$, while before the SET of out in operation 5, the state of out is the result of operation 2, i.e., $not(in_2)$. For the case $0 \oplus 0$, the energy consumption during the last two operations is therefore very high, while for the case $1 \oplus 1$ is very low in comparison. For the cases $0 \oplus 1$ and $1 \oplus 0$, the energy has an intermediate value.

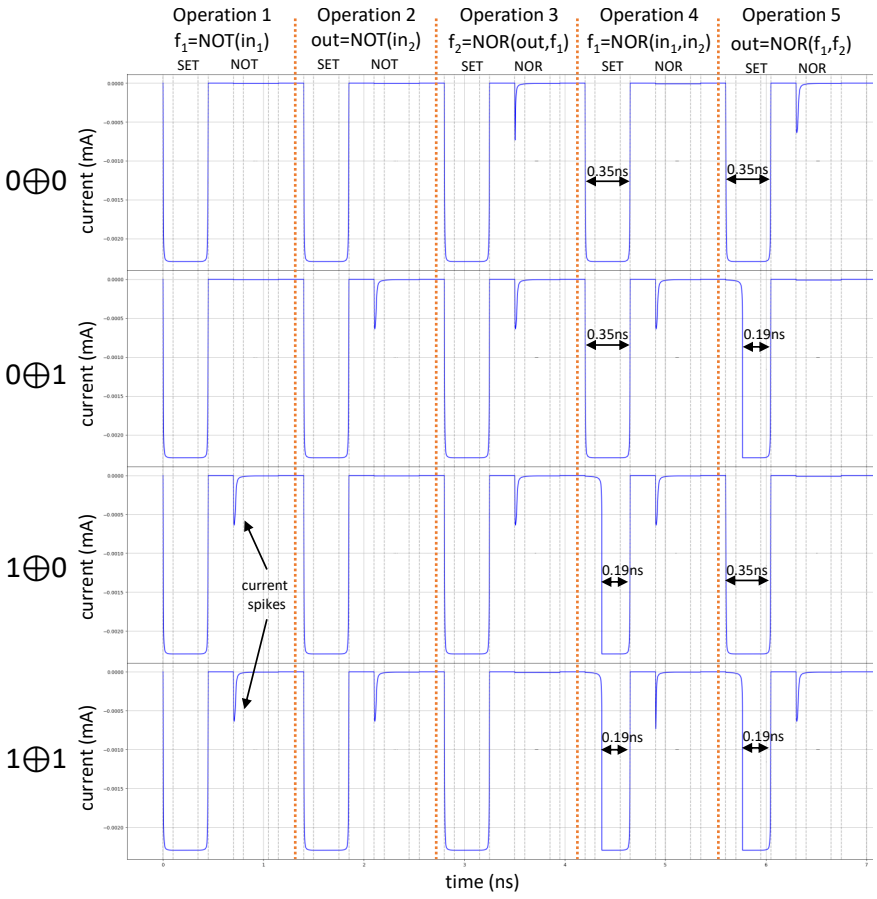


Figure 6.2: MAGIC-based XOR current curves.

6.2.2 DPA on MAGIC-based XOR

In the execution of the DPA, the supply current measurements of a large number of encryption steps are divided over two sets by means of a selection function based on the information leakage model (which is data-dependent) and a guess on the secret key. The difference between the averages of the two sets will approach zero for a wrong key guess, but has noticeable peaks if the correct secret key has been predicted.

In order to prove that the MAGIC-based XOR operation can be attacked via DPA, we have created a circuit able to perform eight 2-bit XOR operations at the same time. We have fixed one of the two inputs (to emulate the presence of a secret key) and we have

applied exhaustively all possible input combinations (i.e., 256). Based on observation of figure 6.2, we have created our selection function in such a way that 0⊕0 operations belong to the first set (the one contributing to the energy consumption), 1⊕1 belong to the second set, while 0⊕1 and 1⊕0 are ignored. We used the tool in [150] to perform the DPA. The result of the attack is shown in figure 6.3, where each line represents the DPA result for each key guess. The line of maximum amplitude corresponds to the correct key guess, thus showing the success of the attack. These results have been obtained by including process variability (5% standard deviation of the resistive values) and noise on the control signals (white noise with a maximum of 10% variation) in the simulation. However, the results are not affected at all by these effects, since the large difference between R_{ON} and R_{OFF} overtakes them.

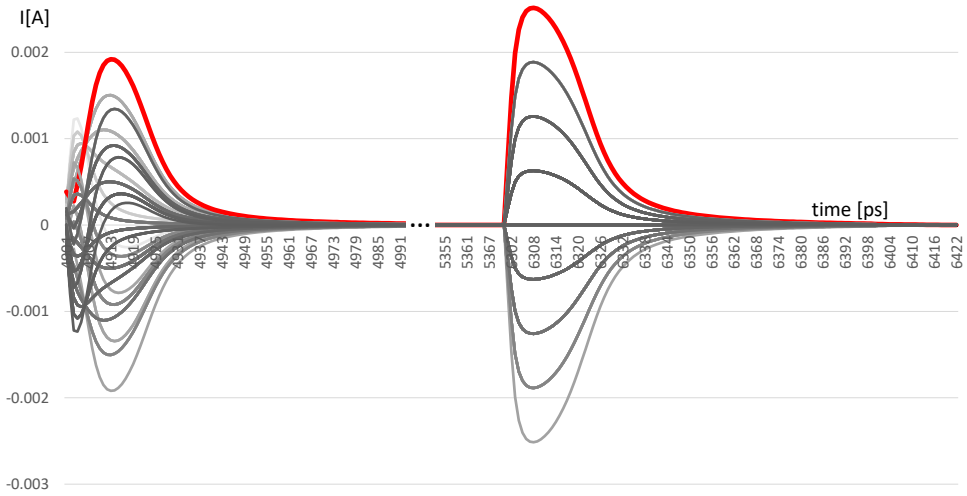


Figure 6.3: DPA result on eight 2-bit XOR operations. The red line corresponds to the correct key guess.

This finding is noteworthy as we are dealing with a system that uses resistors with two vastly different resistance values, and the currents involved in the computation are on the order of mA when the resistance is low or on the order of µA when the resistance is high. Therefore, there is a significant correlation with the processed data, which can be exploited by side-channel analysis. Moreover, this outcome can be extended to any type of CIM where the values of resistances (or currents) differ substantially between the two logical states.

6.3 Fault Analysis on MAGIC-based XOR

Fault attacks [68] are a class of attacks that exploit weaknesses in a system by introducing controlled faults or errors in its operation. The goal of a fault attack is to cause the system to behave in an unintended way, reveal sensitive information, or break its security measures. Fault attacks can be performed by manipulating the physical environment in which the system operates, such as temperature, voltage, electromagnetic radiation, or clock signals. DFAs and SEAs are especially potent against cryptographic systems. A DFA exploits the differences in the behaviour of a system when operating normally versus under faulty conditions to obtain secret information. On the other hand, a SEA induces transient faults to cause a single-bit information leak, depending on whether the targeted algorithm produces an error or not. Since the most significant logic operation for cryptography is the XOR, we show the effects of voltage manipulation on the behaviour of the MAGIC-based XOR, to demonstrate the feasibility of DFA and SEA on MAGIC-based cryptography. As shown in table 6.1, the MAGIC-based XOR operation is a concatenation of several NOR (and NOT) operations. If one of these is not performed correctly, the result of the XOR is not correct either.

6.3.1 Fault Analysis of MAGIC-based XOR

In this study, we assume an attacker is able to manipulate the voltage of the system to change the behaviour of NOR and NOT operations, thus affecting the result of the XOR. Under these conditions, three scenarios are possible:

1. The attacker manipulates only the SET voltage (V_{SET});
2. The attacker manipulates only the control voltage V_0;
3. The attacker manipulates the main power supply, thus affecting both the SET voltage (V_{SET}) and the control voltage V_0 in one or multiple cycles.

To understand the effect of this attack on the full XOR gate, we have first analysed its effect on the basic NOR/NOT operations. The following effects have been observed:

1. If V_{SET} is below the threshold voltage of the SET operation, the memristor cannot be initialised at logic 1, which is the first step in performing any NOR/NOT operation. This will not affect the correctness of the operation if the memristor is already at logic 1. However, if the memristor is at logic 0, the correctness of the operation depends on the input values, as shown in table 6.2, column "Low V_{SET}". The 'X' value is shown when the result of the operation depends on the initial state of the memristor, which might be unknown.

2. If V_0 is low, the voltage drop on the output memristor might not be enough to change its state from logic 1 to logic 0, when at least one of the inputs is at logic 1 (as per MAGIC-operation principle described in section 2.2.1). This situation is illustrated in table 6.2, in the column "Low V_0".

3. If both V_{SET} and V_0 are low, the two effects described before are combined, and the output memristor is not able to change its initial state, as shown in table 6.2, column "Low V_{SET} and V_0".

	Inputs	Initial output state	Expected	Low V_{SET}	Low V_0	Low V_{SET} and V_0
NOT	0	0	1	X	1	X
		1				
	1	0	0	0	1	X
		1				
NOR	00	0	1	X	1	X
		1				
	01	0	0	0	1	X
		1				
	10	0	0	0	1	X
		1				
	11	0	0	0	1	X
		1				

Table 6.2: MAGIC NOT and NOR behaviour for control voltages affected by external perturbations.

MAGIC XOR is built as the concatenation of 5 NOT/NOR steps. We have considered

several Attack Scenarios (ASs), based on the instant and duration of the perturbation: (AS1) only the V_{SET} in one of the 5 steps; (AS2) only the V_0 in one of the 5 steps; (AS3) both V_{SET} and V_0 in one or multiple consecutive steps. These attacks can be realised with expensive means (AS1 and AS2, which require very precise time resolution, such as EM injection) or with more affordable means for the attack AS3 (voltage or power glitching). Table 6.3 shows the expected outputs of the XOR operation when affected by the aforementioned attacks.

	AS1	AS2	AS3 (Low V_{SET} and V_0)				
	Low V_{SET}	Low V_0	1 step	2 steps	3 steps	4 steps	5 steps
Step 1 (S1)	0X10	0111	0X1X				
Step 2 (S2)	01X0	0111	01XX	0XXX (S1+S2)			
Step 3 (S3)	011X	0000	0XXX	0XXX (S2+S3)	0XXX (S1+S2+S3)		
Step 4 (S4)	0110	0000	0010	00XX (S3+S4)	00XX (S2+S3+S4)	XXXX (S1+S2+S3+S4)	
Step 5 (S5)	0010	1111	1010	1010 (S4+S5)	1010 (S3+S4+S5)	XXXX (S2+S3+S4+S5)	XXXX (all steps)

Table 6.3: Results of the attack on the XOR gate for the three proposed attack scenarios. The four bits in each cell represent the result of the XOR for the four input combinations 00, 01, 10, 11. The expected value is "0110''.

In table 6.3, cells are filled with different colours, based on the exploitability of the attack:

- Green cell: the attack has no effect on the result of the operation;
- Yellow cells: the effect of the attack depends on the initial state of the memristors f_1, f_2 and out before the execution of the XOR operation. This state might be unknown to the attacker, and therefore the exploitation of this attack is not guaranteed;
- Red cells: the effect of the attack can be predicted and exploited, no matter the initial states of the memristors.

Based on the results in table 6.3, we can conclude that an attack targeting a perturbation of V_0 in one cycle, will always succeed in altering the output of the XOR computation. If the attack is performed with lower resolution (i.e., targeting both V_{SET} and V_0 over one or multiple cycles), the probability of a successful attack is reduced.

6.4 Conclusion

CIM holds great potential for tackling the obstacles encountered by conventional computing architectures, including the memory wall and the energy expenditure of data transfer. Moreover, this approach could enhance security measures, as limited data movement decreases the likelihood of information leakage through communication buses. In this chapter, we have presented the results of side-channel and fault analyses on the MAGIC solution, which is representative of in-array memristive-based logic computation. We have shown that the MAGIC-based XOR operations are sensitive to both side-channel analysis and fault attacks. More in particular, we have demonstrated that the significant correlation with processed data resulting from the utilisation of resistors with vastly different resistance, highlights the potential for side-channel analysis. We have also shown that perturbation of voltage sources is an efficient means of inflicting fault attacks. To conclude, even under the assumption supported by CIM that there is no data movement, it is still crucial to implement countermeasures to safeguard sensitive data and ensure the integrity of the computation. Regarding countermeasures, the classical ones used for CMOS-based architectures, such as hiding [151] or masking [152], could represent a first approach to reduce information leakage: the key idea is to mask the connection between the leakage of the device and the operations being processed. However, [148] indicates that hiding could be not as effective as for CMOS, and suggests to carefully assess such countermeasures for memristive applications, showing the need for new methods to protect memristive LIM implementation from Side Channel Attacks.

7

Concluding Remarks and Future Directions

In the world of microelectronics, there has been a longstanding quest to merge logic and memory, aiming to move away from the conventional von Neumann architecture with its ever-increasing limitations, such as the power and memory wall. This endeavour seeks to go beyond the constraints and bottlenecks inherent in traditional designs and achieve a significant technological advancement.

The chapters of this thesis have shown the versatility of NVRAMs: resistive switching properties, computing capabilities, and the potential to redefine the boundaries of logic and memory integration. We have navigated through the possible LIM implementations, analysing and pondering the pros and cons of each one with a focus on security implementations.

The objective has been to explore the synthesis of Boolean operations within the LIM framework and to embark on a security study that looks for and analyses the vulnerabilities of these novel architectures. The thesis work commenced with the mission of devising a method capable of synthesising a repertoire of Boolean operations, from the most basic

2-bit operations to the Full Adder, all while being efficient in terms of occupied area and operational cycles. In parallel, we designed a Simulation and Analysis environment capable of handling parallel simulations, paving the way for parametric sweep. These foundational contributions set the stage for the chapters that followed.

Firstly, we have created a framework that has allowed a comprehensive analysis of the existing LIM solutions and a subsequent comparative study in terms of memory resource requirements and number of operations needed to implement fundamental Boolean functions. The results showed a varied outlook, where the number of steps required for operations varied significantly among solutions. Yet, this was but a preliminary glance since the execution time of these operations remained deeply dependent on the electrical characteristics of the memristor model and on the actual time it needs to change its logic value instead of relying just on the number of steps. This chapter began our exploration and simulation, giving us a first idea of the potential of memristive-based LIM technologies.

To comprehend the electrical behaviour of the solutions we targeted, we enriched our toolbox with a framework enabling us to read pseudo-assembly code, generate netlist inputs (test benchs), run Cadence simulations, and harvest the simulation data. It is a toolkit that gave us a way to explore and understand the behaviour of LIM operations under different conditions.

The first application for the toolkit has been to delve into the electrical analysis of fundamental Boolean operations under ideal operating conditions. It enabled us to uncover the first vulnerabilities of some solutions, as we had to deal with the definition of correct operation: in some cases the resistance of output memristors often deviated from their ideal (expected) values. As the value of the memristors is treated as a logic value, it can have a high threshold to distinguish the logic 1 to the logic 0 for the conversion from analog to digital. But this can create many issues once we want to perform computations, and we use a non-ideal value as input for another operation. This was the first sign of the vulnerabilities underlying memristive-based LIM, and it gave us insights into voltage regulation, reliability, and the potential need for enhancements.

As a consequence, we focused on operations executed under non-ideal conditions, to delve deeper into what we had experienced in the previous simulations. We discovered

that the resistive value of the input memristors played a central role in determining the ranges within which operation was executed correctly. The concatenation of operations, particularly when initiated from non-ideal values, introduced the spectre of incorrect results. This highlighted the importance of exploring the robustness and efficiency of complex memristive operations, introducing the possibility of refresh cycles in case it was impossible to obtain operations stable enough. Moreover, we were able to perform a comparison on XOR implementations, being the XOR operation the core operation in security-based primitives (crypto-cores). This, added with the information on the behaviour of the single Boolean operations, allowed us to choose the most interesting implementation: the MAGIC-based XOR.

Having chosen the key operation to analyse, we explored the security-related issues. Side channel and fault analyses have been our main tools to highlight the vulnerabilities of the MAGIC-based XOR, showing an immature technology with many issues. In fact, the large resistive difference between LRS and HRS results in high voltage and current needed to make the operations work and makes it an easy target for side channel attacks. The lack in robustness allows to easily manipulate the operations, also due to the immaturity of the technology. This therefore permits to easily perform fault attacks.

This PhD journey leaves many open questions about the technology, the security issues and the possible countermeasures that could be put in place. Possible points to explore are:

- LIM Synthesis Methods that could not only improve the behaviour of the LIM solutions (e.g., by optimising the number of memristors and the execution time): there could be the option to mask the ongoing operations by possible side channel analyses.

- Real Time reactivity: implementing real-time adaptation mechanisms that can ensure reliability and correctness during consecutive operations (e.g., by inserting refresh cycles to make operations behave correctly).

- Variability and Maturity: these memristive technologies are still at a low stage of maturity. A higher technology maturity shall be vital to enhance the predictability

and robustness of LIM systems. The variability, that has been only marginally considered in this thesis, represents a very important variable in the correct behaviour of the LIM operations, and it needs to be further studied.

- Countermeasures to the attacks: the security challenges shown in the last chapter of the thesis showed the need of countermeasures tailored to memristive LIM architectures, safeguarding data and computation.

Concluding, the memristive-based LIM represents a very interesting and innovating paradigm, but at the same time it is too immature to be considered as a basis to perform secure operations and it will need more technological advancements and the implementation of security countermeasures.

List of Publications

- **P. Inglese**, E. I. Vatajelu, and G. Di Natale, *"Side Channel and Fault Analyses on Memristor-Based Logic In-Memory,"* in IEEE Design & Test, Oct. 2023.
- S. V. Gutiérrez, **P. Inglese**, G. Di Natale, and E.-I. Vatajelu, *"Open Automation Framework for Complex Parametric Electrical Simulations,"* in 2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), May 2023.
- **P. Inglese**, E. I. Vatajelu, and G. Di Natale, *"Memristive Logic-in-Memory Implementations: A Comparison,"* in SMACD / PRIME 2021; International Conference on SMACD and 16th Conference on PRIME, Jul. 2021.
- **P. Inglese**, E. I. Vatajelu, and G. Di Natale, *"On the Limitations of Concatenating Boolean Operations in Memristive-Based Logic-In-Memory Solutions,"* in 2021 16th International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS), Jun. 2021.

Participation in Local and International Activities

International Activities

- PhD forum presentation at IEEE ETS 2022.
- PhD forum presentation at IEEE ETS 2021.
- Aid in the organisation of the IEEE ETS 2021 and the associated Spring School TSS 2021.
- Aid in the organisation of two editions of the Global Test Community Quiz, that have been held in the 2021 editions of IEEE ETS and IEEE ITC.
- Peer-reviewer and TPC member for the 2022 DAC Conference (San Francisco, USA).
- Participation with a published article at the 16th International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS), 2021.
- Participation with a published article at SMACD / PRIME 2021: International Conference on SMACD and 16th Conference on PRIME.

Local Activities

- TIMA Scientific day - Hardware Security, 2023.
- TIMA PhD day, 2022.
- Junior Scientist & Industry annual meeting, 2022.
- TIMA Scientific day - Emerging Technologies, 2021.
- TIMA & RFIC PhD day, 2021.
- First scientific day of the PEM department - Doctoral schools EEATS, I-MAP^2 and Physics, 2021.

Bibliography

References

[1] IRDS™ 2022: Executive Summary - IEEE IRDS™. Technical report, 2022.

[2] Ihsen Alouani, Wael M. Elsharkasy, Ahmed M. Eltawil, Fadi J. Kurdahi, and Smail Niar. AS8-static random access memory (SRAM): asymmetric SRAM architecture for soft error hardening enhancement. *IET Circuits, Devices & Systems*, 11(1):89–94, 2017.

[3] Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, Said Hamdioui, and Koen Bertels. A Mapping Methodology of Boolean Logic Circuits on Memristor Crossbar. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(2):311–323, February 2018.

[4] Gordon E Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):4, 1965.

[5] GE Moore. Progress in digital electronics. In *Technical Digest of the Int'l Electron Devices Meeting*. IEEE Press, 1975.

[6] R.H. Dennard, F.H. Gaensslen, Hwa-Nien Yu, V.L. Rideout, E. Bassous, and A.R. LeBlanc. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, October 1974.

[7] Hadi Esmaeilzadeh, Emily Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, pages 365–376, June 2011.

[8] J. Neumann. *First draft of a report on the EDVAC*. University of Pennsylvania, Philadelphia, 1945.

[9] John Backus. Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs. *Communications of the ACM*, 21(8):613–641, August 1978.

[10] Nihal Kularatna. Chapter 5 - Digital Signal Processors. In Nihal Kularatna, editor, *Modern Component Families and Circuit Block Design*, pages 197–239. Newnes, Woburn, January 2000.

[11] Wm. A. Wulf and Sally A. McKee. Hitting the memory wall: implications of the obvious. *ACM SIGARCH Computer Architecture News*, 23(1):20–24, March 1995.

[12] Sally A. McKee. Reflections on the memory wall. In *Proceedings of the 1st conference on Computing frontiers*, CF '04, page 162, New York, NY, USA, April 2004. Association for Computing Machinery.

[13] D J Kuck and D H Lawrie. *The use and performance of memory hierarchies: a survey*. Report (University of Illinois at Urbana-Champaign. Dept. of Computer Science)no. 363. Dept. of Computer Science, University of Illinois, Urbana, 1969.

[14] Stephen W. Keckler, William J. Dally, Brucek Khailany, Michael Garland, and David Glasco. GPUs and the Future of Parallel Computing. *IEEE Micro*, 31(5):7–17, September 2011.

[15] Pradip Bose. Power Wall. In David Padua, editor, *Encyclopedia of Parallel Computing*, pages 1593–1608. Springer US, Boston, MA, 2011.

[16] Furqan Zahoor, Tun Zainal Azni Zulkifli, and Farooq Ahmad Khanday. Resistive Random Access Memory (RRAM): an Overview of Materials, Switching Mechanism, Performance, Multilevel Cell (mlc) Storage, Modeling, and Applications. *Nanoscale Research Letters*, 15(1):90, April 2020.

[17] Geoffrey W. Burr, Matthew J. Breitwisch, Michele Franceschini, Davide Garetto, Kailash Gopalakrishnan, Bryan Jackson, Bülent Kurdi, Chung Lam, Luis A. Las-

tras, Alvaro Padilla, Bipin Rajendran, Simone Raoux, and Rohit S. Shenoy. Phase change memory technology. *Journal of Vacuum Science & Technology B*, 28(2):223–262, March 2010.

[18] Dmytro Apalkov, Alexey Khvalkovskiy, Steven Watts, Vladimir Nikitin, Xueti Tang, Daniel Lottis, Kiseok Moon, Xiao Luo, Eugene Chen, Adrian Ong, Alexander Driskill-Smith, and Mohamad Krounbi. Spin-transfer torque magnetic random access memory (STT-MRAM). *ACM Journal on Emerging Technologies in Computing Systems*, 9(2):13:1–13:35, May 2013.

[19] Asif Islam Khan, Ali Keshavarzi, and Suman Datta. The future of ferroelectric field-effect transistor technology. *Nature Electronics*, 3(10):588–597, October 2020.

[20] R. Yung and N.C. Wilhelm. Caching processor general registers. In *Proceedings of ICCD '95 International Conference on Computer Design. VLSI in Computers and Processors*, pages 307–312, October 1995.

[21] Geoffrey Burr, Bulent Kurdi, Campbell Scott, Chung Lam, Kailash Gopalakrishnan, and Rohit Shenoy. Overview of candidate device technologies for Storage-Class Memory. *IBM Journal of Research and Development*, 52:449–464, July 2008.

[22] Abu Sebastian, Manuel Le Gallo, Riduan Khaddam-Aljameh, and Evangelos Eleftheriou. Memory devices and applications for in-memory computing. *Nature Nanotechnology*, pages 1–16, March 2020.

[23] Yong Huang, Zihan Shen, Ye Wu, Xiaoqiu Wang, Shufang Zhang, Xiaoqin Shi, and Haibo Zeng. Amorphous ZnO based resistive random access memory. *RSC Advances*, 6(22):17867–17872, February 2016.

[24] Matthias Wuttig and Noboru Yamada. Phase-change materials for rewriteable data storage. *Nature Materials*, 6(11):824–832, November 2007.

[25] Geoffrey W. Burr, Alvaro Padilla, Michele Franceschini, Bryan Jackson, Diego G. Dupouy, Charles T. Rettner, Kailash Gopalakrishnan, Rohit Shenoy, and John Karidis. The inner workings of phase change memory: Lessons from prototype PCM devices. In *2010 IEEE Globecom Workshops*, pages 1890–1894, December 2010.

[26] Guido De Sandre, Luca Bettini, Alessandro Pirola, Lionel Marmonier, Marco Pasotti, Massimo Borghi, Paolo Mattavelli, Paola Zuliani, Luca Scotti, Gianfranco Mastracchio, Ferdinando Bedeschi, Roberto Gastaldi, and Roberto Bez. A 90nm 4Mb embedded phase-change memory with 1.2V 12ns read access time and 1MB/s write throughput. In *2010 IEEE International Solid-State Circuits Conference - (ISSCC)*, pages 268–269, February 2010.

[27] Shahar Kvatinsky, Dmitry Belousov, Slavik Liman, Guy Satat, Nimrod Wald, Eby G. Friedman, Avinoam Kolodny, and Uri C. Weiser. MAGIC—Memristor-Aided Logic. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 61(11):895–899, November 2014.

[28] Shahar Kvatinsky, Guy Satat, Nimrod Wald, Eby G. Friedman, Avinoam Kolodny, and Uri C. Weiser. Memristor-Based Material Implication (IMPLY) Logic: Design Principles and Methodologies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(10):2054–2066, October 2014.

[29] Kyung Min Kim and R. Stanley Williams. A Family of Stateful Memristor Gates for Complete Cascading Logic. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(11):4348–4355, November 2019.

[30] Shaahin Angizi, Zhezhi He, Amro Awad, and Deliang Fan. MRIMA: An MRAM-based In-Memory Accelerator. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2019.

[31] Pierre-Emmanuel Gaillardon, Luca Amarú, Anne Siemon, Eike Linn, Rainer Waser, Anupam Chattopadhyay, and Giovanni De Micheli. The Programmable Logic-in-Memory (PLiM) Computer. In *Proceedings of the 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 427–432. Research Publishing Services, 2016.

[32] Lei Xie, H.A. Du Nguyen, Jintao Yu, Ali Kaichouhi, Mottaqiallah Taouil, Mohammad AlFailakawi, and Said Hamdioui. Scouting Logic: A Novel Memristor-Based Logic

Design for Resistive Computing. In *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 176–181, July 2017.

[33] Shuangchen Li, Cong Xu, Qiaosha Zou, Jishen Zhao, Yu Lu, and Yuan Xie. Pinatubo: a processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *Proceedings of the 53rd Annual Design Automation Conference on - DAC '16*, pages 1–6, Austin, Texas, 2016. ACM Press.

[34] Hussein Assaf, Yvon Savaria, and Mohamad Sawan. Vector Matrix Multiplication Using Crossbar Arrays: A Comparative Analysis. In *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 609–612, December 2018.

[35] Ming-Hung Wu, Ming-Chun Hong, Chih-Cheng Chang, Paritosh Sahu, Jeng-Hua Wei, Heng-Yuan Lee, Shyh-Shyuan Shcu, and Tuo-Hung Hou. Extremely Compact Integrate-and-Fire STT-MRAM Neuron: A Pathway toward All-Spin Artificial Deep Neural Network. In *2019 Symposium on VLSI Technology*, pages T34–T35, June 2019.

[36] Adrien F. Vincent, Jérôme Larroque, Nicolas Locatelli, Nesrine Ben Romdhane, Olivier Bichler, Christian Gamrat, Wei Sheng Zhao, Jacques-Olivier Klein, Sylvie Galdin-Retailleau, and Damien Querlioz. Spin-Transfer Torque Magnetic Memory as a Stochastic Memristive Synapse for Neuromorphic Systems. *IEEE Transactions on Biomedical Circuits and Systems*, 9(2):166–174, April 2015.

[37] Deliang Fan, Yong Shim, Anand Raghunathan, and Kaushik Roy. STT-SNN: A Spin-Transfer-Torque Based Soft-Limiting Non-Linear Neuron for Low-Power Artificial Neural Networks. *IEEE Transactions on Nanotechnology*, 14(6):1013–1023, November 2015.

[38] Bonan Yan, Mengyun Liu, Yiran Chen, Krishnendu Chakrabarty, and Hai Li. On Designing Efficient and Reliable Nonvolatile Memory-Based Computing-In-Memory Accelerators. In *2019 IEEE International Electron Devices Meeting (IEDM)*, pages 14.5.1–14.5.4, December 2019.

[39] Gagandeep Singh, Lorenzo Chelini, Stefano Corda, Ahsan Javed Awan, Sander Stuijk, Roel Jordans, Henk Corporaal, and Albert-Jan Boonstra. Near-memory computing: Past, present, and future. *Microprocessors and Microsystems*, 71:102868, November 2019.

[40] Kaya Can Akyel, Henri-Pierre Charles, Julien Mottin, Bastien Giraud, Gregory Suraci, Sebastien Thuries, and Jean-Philippe Noel. DRC2 : Dynamically Reconfigurable Computing Circuit based on memory architecture. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8, San Diego, CA, USA, October 2016. IEEE.

[41] Shaizeen Aga, Supreet Jeloka, Arun Subramaniyan, Satish Narayanasamy, David Blaauw, and Reetuparna Das. Compute Caches. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 481–492, Austin, TX, February 2017. IEEE.

[42] Yiqun Zhang, Li Xu, Kaiyuan Yang, Qing Dong, Supreet Jeloka, David Blaauw, and Dennis Sylvester. Recryptor: A reconfigurable in-memory cryptographic Cortex-M0 processor for IoT. In *2017 Symposium on VLSI Circuits*, pages C264–C265, Kyoto, Japan, June 2017. IEEE.

[43] Vivek Seshadri, Todd C. Mowry, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A. Kozuch, Onur Mutlu, and Phillip B. Gibbons. Ambit: in-memory accelerator for bulk bitwise operations using commodity DRAM technology. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-50 '17*, pages 273–287, Cambridge, Massachusetts, 2017. ACM Press.

[44] Shuangchen Li, Dimin Niu, Krishna T. Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. DRISA: a DRAM-based Reconfigurable In-Situ Accelerator. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-50 '17*, pages 288–301, Cambridge, Massachusetts, 2017. ACM Press.

[45] Shubham Jain, Ashish Ranjan, Kaushik Roy, and Anand Raghunathan. Computing in Memory With Spin-Transfer Torque Magnetic RAM. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(3):470–483, March 2018.

[46] Liuyang Zhang, Erya Deng, Hao Cai, You Wang, Lionel Torres, Aida Todri-Sanial, and Youguang Zhang. A high-reliability and low-power computing-in-memory implementation within STT-MRAM. *Microelectronics Journal*, 81:69–75, November 2018.

[47] Farhana Parveen, Zhezhi He, Shaahin Angizi, and Deliang Fan. HielM: Highly flexible in-memory computing using STT MRAM. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 361–366, January 2018.

[48] Abbas Rahimi, Amirali Ghofrani, Miguel Angel Lastras-Montano, Kwang-Ting Cheng, Luca Benini, and Rajesh K. Gupta. Energy-efficient GPGPU architectures via collaborative compilation and memristive memory-based computing. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2014.

[49] Leonid Yavits, Shahar Kvatinsky, Amir Morad, and Ran Ginosar. Resistive Associative Processor. *IEEE Computer Architecture Letters*, 14(2):148–151, July 2015.

[50] Supreet Jeloka, Naveen Bharathwaj Akesh, Dennis Sylvester, and David Blaauw. A 28 nm Configurable Memory (TCAM/BCAM/SRAM) Using Push-Rule 6T Bit Cell Enabling Logic-in-Memory. *IEEE Journal of Solid-State Circuits*, 51(4):1009–1021, April 2016.

[51] Roman Kaplan, Leonid Yavits, Ran Ginosar, and Uri Weiser. A Resistive CAM Processing-in-Storage Architecture for DNA Sequence Alignment. *IEEE Micro*, 37(4):20–28, 2017.

[52] Mohsen Imani, Yeseong Kim, and Tajana Rosing. MPIM: Multi-purpose in-memory processing using configurable resistive memory. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 757–763, January 2017.

[53] Mohsen Imani and Tajana Rosing. CAP: Configurable resistive associative processor for near-data computing. In *2017 18th International Symposium on Quality Electronic Design (ISQED)*, pages 346–352, March 2017.

[54] Mohsen Imani, Saransh Gupta, Atl Arredondo, and Tajana Rosing. Efficient query processing in crossbar memory. In *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, July 2017.

[55] Wenqin Huangfu, Shuangchen Li, Xing Hu, and Yuan Xie. RADAR: A 3D-ReRAM based DNA Alignment Accelerator Architecture. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2018.

[56] M. Gokhale, B. Holmes, and K. Iobst. Processing in memory: the Terasys massively parallel PIM array. *Computer*, 28(4):23–31, April 1995.

[57] T.L. Sterling and H.P. Zima. Gilgamesh: A Multithreaded Processor-In-Memory Architecture for Petaflops Computing. In *SC '02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, pages 48–48, November 2002.

[58] Qiuling Zhu, Berkin Akin, H. Ekin Sumbul, Fazle Sadi, James C. Hoe, Larry Pileggi, and Franz Franchetti. A 3D-stacked logic-in-memory accelerator for application-specific data intensive computing. In *2013 IEEE International 3D Systems Integration Conference (3DIC)*, pages 1–7, October 2013.

[59] Dongping Zhang, Nuwan Jayasena, Alexander Lyashevsky, Joseph L. Greathouse, Lifan Xu, and Michael Ignatowski. TOP-PIM: throughput-oriented programmable processing in memory. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, HPDC '14, pages 85–98, New York, NY, USA, June 2014. Association for Computing Machinery.

[60] Dae Hyun Kim, Krit Athikulwongse, Michael B. Healy, Mohammad M. Hossain, Moongon Jung, Ilya Khorosh, Gokul Kumar, Young-Joon Lee, Dean L. Lewis, Tzu-Wei Lin, Chang Liu, Shreepad Panth, Mohit Pathak, Minzhen Ren, Guanhao Shen, Taigon Song, Dong Hyuk Woo, Xin Zhao, Joungho Kim, Ho Choi, Gabriel H. Loh,

Hsien-Hsin S. Lee, and Sung Kyu Lim. Design and Analysis of 3D-MAPS (3D Massively Parallel Processor with Stacked Memory). *IEEE Transactions on Computers*, 64(1):112–125, January 2015.

[61] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. A scalable processing-in-memory accelerator for parallel graph processing. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pages 105–117, June 2015.

[62] Yibin Tang, Ying Wang, Huawei Li, and Xiaowei Li. ApproxPIM: Exploiting realistic 3D-stacked DRAM for energy-efficient processing in-memory. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 396–401, January 2017.

[63] Chenhao Xie, Shuaiwen Leon Song, Jing Wang, Weigong Zhang, and Xin Fu. Processing-in-Memory Enabled Graphics Processors for 3D Rendering. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 637–648, February 2017.

[64] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, Lecture Notes in Computer Science, pages 388–397, Berlin, Heidelberg, 1999. Springer.

[65] Jean-Jacques Quisquater and David Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-measures for Smart Cards. In Isabelle Attali and Thomas Jensen, editors, *Smart Card Programming and Security*, Lecture Notes in Computer Science, pages 200–210, Berlin, Heidelberg, 2001. Springer.

[66] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic Analysis: Concrete Results. In Çetin K. Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2001*, Lecture Notes in Computer Science, pages 251–261, Berlin, Heidelberg, 2001. Springer.

[67] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO*

'96, Lecture Notes in Computer Science, pages 104–113, Berlin, Heidelberg, 1996. Springer.

[68] Duško Karaklajić, Jörn-Marc Schmidt, and Ingrid Verbauwhede. Hardware Designer's Guide to Fault Attacks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(12):2295–2306, December 2013.

[69] Elisabeth Oswald and François-Xavier Standaert. Side-Channel Analysis and Its Relevance to Fault Attacks. In Marc Joye and Michael Tunstall, editors, *Fault Analysis in Cryptography*, Information Security and Cryptography, pages 3–15. Springer, Berlin, Heidelberg, 2012.

[70] Gilles Piret and Jean-Jacques Quisquater. A Differential Fault Attack Technique against SPN Structures, with Application to the AES and Khazad. In Colin D. Walter, Çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003*, Lecture Notes in Computer Science, pages 77–88, Berlin, Heidelberg, 2003. Springer.

[71] Michael Hutter and Jörn-Marc Schmidt. The Temperature Side Channel and Heating Fault Attacks. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications*, Lecture Notes in Computer Science, pages 219–235, Cham, 2014. Springer International Publishing.

[72] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, Lecture Notes in Computer Science, pages 513–525, Berlin, Heidelberg, 1997. Springer.

[73] Alexandre Berzati, Cécile Canovas-Dumas, and Louis Goubin. A Survey of Differential Fault Analysis Against Classical RSA Implementations. In Marc Joye and Michael Tunstall, editors, *Fault Analysis in Cryptography*, Information Security and Cryptography, pages 111–124. Springer, Berlin, Heidelberg, 2012.

[74] Sung-Ming Yen and M. Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Transactions on Computers*, 49(9):967–970, September 2000.

[75] Christophe Clavier. Attacking Block Ciphers. In Marc Joye and Michael Tunstall, editors, *Fault Analysis in Cryptography*, Information Security and Cryptography, pages 19–35. Springer, Berlin, Heidelberg, 2012.

[76] L. Chua. Memristor-The missing circuit element. *IEEE Transactions on Circuit Theory*, 18(5):507–519, September 1971.

[77] Leon O. Chua. How we predicted the memristor. *Nature Electronics*, 1(5):322–322, May 2018.

[78] Dmitri B. Strukov, Gregory S. Snider, Duncan R. Stewart, and R. Stanley Williams. The missing memristor found. *Nature*, 453(7191):80–83, May 2008.

[79] The memristor revisited. *Nature Electronics*, 1(5):261–261, May 2018.

[80] Ting Chang, Sung-Hyun Jo, and Wei Lu. Short-Term Memory to Long-Term Memory Transition in a Nanoscale Memristor. *ACS Nano*, 5(9):7669–7676, September 2011.

[81] Lu Zhang, Zhijie Chen, J. Joshua Yang, Bryant Wysocki, Nathan McDonald, and Yiran Chen. A compact modeling of TiO2-TiO2–x memristor. *Applied Physics Letters*, 102(15):153503, April 2013.

[82] T. D. Dongale, N. D. Desai, K. V. Khot, C. K. Volos, P. N. Bhosale, and R. K. Kamat. An Electronic Synapse Device Based on TiO2 Thin Film Memristor. *Journal of Nanoelectronics and Optoelectronics*, 13(1):68–75, January 2018.

[83] T. Driscoll, H.-T. Kim, B.-G. Chae, M. Di Ventra, and D. N. Basov. Phase-transition driven memristive system. *Applied Physics Letters*, 95(4):043503, July 2009.

[84] Ya Li, Jinxing Chu, Weijie Duan, Guangshuo Cai, Xihua Fan, Xinzhong Wang, Gang Wang, and Yanli Pei. Analog and Digital Bipolar Resistive Switching in Solution-Combustion-Processed NiO Memristor. *ACS Applied Materials & Interfaces*, 10(29):24598–24606, July 2018.

[85] Ya Li, Paiwen Fang, Xihua Fan, and Yanli Pei. NiO-based memristor with three resistive switching modes. *Semiconductor Science and Technology*, 35(5):055004, March 2020.

[86] Zhenyu Zhou, Xiaobing Yan, Jianhui Zhao, Chao Lu, Deliang Ren, Nianduan Lu, Jingjuan Wang, Lei Zhang, Xiaoyan Li, Hong Wang, and Mengliu Zhao. Synapse behavior characterization and physical mechanism of a TiN/SiOx/p-Si tunneling memristor device. *Journal of Materials Chemistry C*, 7(6):1561–1567, February 2019.

[87] Tae-Hyeon Kim, Hussein Nili, Min-Hwi Kim, Kyung Kyu Min, Byung-Gook Park, and Hyungjin Kim. Reset-voltage-dependent precise tuning operation of TiOx/Al2O3 memristive crossbar array. *Applied Physics Letters*, 117(15):152103, October 2020.

[88] Ji-Ho Ryu, Chandreswar Mahata, and Sungjun Kim. Long-term and short-term plasticity of Ta2O5/HfO2 memristor for hardware neuromorphic application. *Journal of Alloys and Compounds*, 850:156675, January 2021.

[89] Naveen Verma, Hongyang Jia, Hossein Valavi, Yinqi Tang, Murat Ozatay, Lung-Yen Chen, Bonan Zhang, and Peter Deaville. In-Memory Computing: Advances and Prospects. *IEEE Solid-State Circuits Magazine*, 11(3):43–55, 2019.

[90] Hoang Anh Du Nguyen, Jintao Yu, Muath Abu Lebdeh, Mottaqiallah Taouil, Said Hamdioui, and Francky Catthoor. A Classification of Memory-Centric Computing. *ACM Journal on Emerging Technologies in Computing Systems*, 16(2):13:1–13:26, January 2020.

[91] Han Bao, Houji Zhou, Jiancong Li, Huaizhi Pei, Jing Tian, Ling Yang, Shengguang Ren, Shaoqin Tong, Yi Li, Yuhui He, Jia Chen, Yimao Cai, Huaqiang Wu, Qi Liu, Qing Wan, and Xiangshui Miao. Toward memristive in-memory computing: principles and applications. *Frontiers of Optoelectronics*, 15(1):23, May 2022.

[92] Ioannis Vourkas and Georgios Ch. Sirakoulis. Emerging Memristor-Based Logic Circuit Design Approaches: A Review. *IEEE Circuits and Systems Magazine*, 16(3):15–30, 2016.

[93] Yibo Li, Zhongrui Wang, Rivu Midya, Qiangfei Xia, and J. Joshua Yang. Review of memristor devices in neuromorphic computing: materials sciences and device challenges. *Journal of Physics D: Applied Physics*, 51(50):503002, September 2018.

[94] Giacomo Indiveri, Bernabé Linares-Barranco, Robert Legenstein, George Deligeorgis, and Themistoklis Prodromakis. Integration of nanoscale memristor synapses in neuromorphic computing architectures. *Nanotechnology*, 24(38):384010, September 2013.

[95] Zhongrui Wang, Saumil Joshi, Sergey E. Savel'ev, Hao Jiang, Rivu Midya, Peng Lin, Miao Hu, Ning Ge, John Paul Strachan, Zhiyong Li, Qing Wu, Mark Barnell, Geng-Lin Li, Huolin L. Xin, R. Stanley Williams, Qiangfei Xia, and J. Joshua Yang. Memristors with diffusive dynamics as synaptic emulators for neuromorphic computing. *Nature Materials*, 16(1):101–108, January 2017.

[96] Xinjiang Zhang, Anping Huang, Qi Hu, Zhisong Xiao, and Paul K. Chu. Neuromorphic Computing with Memristor Crossbar. *physica status solidi (a)*, 215(13):1700875, 2018.

[97] Shimeng Yu. Neuro-inspired computing with emerging nonvolatile memorys. *Proceedings of the IEEE*, 106(2):260–285, February 2018.

[98] Tianqi Tang, Lixue Xia, Boxun Li, Rong Luo, Yiran Chen, Yu Wang, and Huazhong Yang. Spiking neural network with RRAM: Can we use it for real-world application? In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 860–865, March 2015.

[99] D. Bankman, J. Messner, A. Gural, and B. Murmann. RRAM-Based In-Memory Computing for Embedded Deep Neural Networks. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pages 1511–1515, November 2019.

[100] Saeideh Shirinzadeh and Rolf Drechsler. Logic Synthesis for In-memory Computing Using Resistive Memories. In *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 375–380, July 2018.

[101] Leon Chua. If It's Pinched It's a Memristor. In Leon Chua, Georgios Ch. Sirakoulis, and Andrew Adamatzky, editors, *Handbook of Memristor Networks*, pages 15–88. Springer International Publishing, Cham, 2019.

[102] Leon Chua. If it's pinched it's a memristor. *Semiconductor Science and Technology*, 29(10):104001, September 2014.

[103] L.O. Chua and Sung Mo Kang. Section-wise piecewise-linear functions: Canonical representation, properties, and applications. *Proceedings of the IEEE*, 65(6):915–929, June 1977.

[104] Said Hamdioui, Shahar Kvatinsky, Gert Cauwenberghs, Lei Xie, Nimrod Wald, Siddharth Joshi, Hesham Mostafa Elsayed, Henk Corporaal, and Koen Bertels. Memristor for computing: Myth or reality? In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 722–731, Lausanne, Switzerland, March 2017. IEEE.

[105] Jintao Yu, Hoang Anh Du Nguyen, Lei Xie, Mottaqiallah Taouil, and Said Hamdioui. Memristive devices for computation-in-memory. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1646–1651, Dresden, March 2018. IEEE.

[106] Anne Siemon, Dirk Wouters, Said Hamdioui, and Stephan Menzel. Memristive Device Modeling and Circuit Design Exploration for Computation-in-Memory. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, Sapporo, Japan, May 2019. IEEE.

[107] Muath Abu Lebdeh, Uljana Reinsalud, Hoang Anh Du Nguyen, Stephan Wong, and Said Hamdioui. Memristive Device Based Circuits for Computation-in-Memory Architectures. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, Sapporo, Japan, May 2019. IEEE.

[108] Yufei Huang, Shuhui Li, Yaguang Yang, and Chengying Chen. Progress on Memristor-Based Analog Logic Operation. *Electronics*, 12(11):2486, January 2023.

[109] N. Peled, R. Ben-Hur, R. Ronen, and S. Kvatinsky. X-MAGIC: Enhancing PIM Using Input Overwriting Capabilities. In *2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SOC)*, pages 64–69, October 2020.

[110] Saransh Gupta, Mohsen Imani, and Tajana Rosing. FELIX: fast and energy-efficient logic in memory. In *Proceedings of the International Conference on Computer-Aided Design*, pages 1–7, San Diego California, November 2018. ACM.

[111] Eero Lehtonen and Mika Laiho. Stateful implication logic with memristors. In *2009 IEEE/ACM International Symposium on Nanoscale Architectures*, pages 33–36, San Francisco, CA, USA, July 2009. IEEE.

[112] A. Siemon, R. Drabinski, M. J. Schultis, X. Hu, E. Linn, A. Heittmann, R. Waser, D. Querlioz, S. Menzel, and J. S. Friedman. Stateful Three-Input Logic with Memristive Switches. *Scientific Reports*, 9(1):1–13, October 2019.

[113] Pietro Inglese, Elena Ioana Vatajelu, and Giorgio Di Natale. Memristive Logic-in-Memory Implementations: A Comparison. In *SMACD / PRIME 2021; International Conference on SMACD and 16th Conference on PRIME*, pages 1–4, July 2021.

[114] Nishil Talati, Saransh Gupta, Pravin Mane, and Shahar Kvatinsky. Logic Design Within Memristive Memories Using Memristor-Aided loGIC (MAGIC). *IEEE Transactions on Nanotechnology*, 15(4):635–650, July 2016.

[115] Mathias Soeken, Pierre-Emmanuel Gaillardon, Saeideh Shirinzadeh, Rolf Drechsler, and Giovanni De Micheli. A PLiM Computer for the Internet of Things. *Computer*, 50(6):35–40, 2017.

[116] Julien Borghetti, Gregory S. Snider, Philip J. Kuekes, J. Joshua Yang, Duncan R. Stewart, and R. Stanley Williams. 'Memristive' switches enable 'stateful' logic operations via material implication. *Nature*, 464(7290):873–876, April 2010.

[117] Shahar Kvatinsky, Avinoam Kolodny, Uri C. Weiser, and Eby G. Friedman. Memristor-based IMPLY logic design procedure. In *2011 IEEE 29th International Conference on Computer Design (ICCD)*, pages 142–147, October 2011.

[118] Nima TaheriNejad. SIXOR: Single-Cycle In-Memristor XOR. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(5):925–935, May 2021.

[119] Maha Kooli, Henri-Pierre Charles, Clément Touzet, Bastien Giraud, and Jean-Philippe Noël. Software platform dedicated for in-memory computing circuit evaluation. In *Proceedings of the 28th International Symposium on Rapid System Prototyping Shortening the Path from Specification to Prototype - RSP '17*, pages 43–49, Seoul, South Korea, 2017. ACM Press.

[120] Maha Kooli, Henri-Pierre Charles, Clement Touzet, Bastien Giraud, and Jean-Philippe Noel. Smart instruction codes for in-memory computing architectures compatible with standard SRAM interfaces. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1634–1639, Dresden, Germany, March 2018. IEEE.

[121] Eero Lehtonen, Jussi H. Poikonen, and Mika Laiho. Memristive Stateful Logic. In Andrew Adamatzky and Leon Chua, editors, *Memristor Networks*, pages 603–623. Springer International Publishing, Cham, 2014.

[122] P. L. Thangkhiew, Rahul Gharpinde, Dev Narayan Yadav, Kamalika Datta, and Indranil Sengupta. Efficient implementation of adder circuits in memristive crossbar array. In *TENCON 2017 - 2017 IEEE Region 10 Conference*, pages 207–212, November 2017.

[123] Ameer Haj-Ali, Rotem Ben-Hur, Nimrod Wald, and Shahar Kvatinsky. Efficient Algorithms for In-Memory Fixed Point Multiplication Using MAGIC. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, Florence, 2018. IEEE.

[124] Phrangboklang Lyngton Thangkhiew, Rahul Gharpinde, and Kamalika Datta. Efficient Mapping of Boolean Functions to Memristor Crossbar Using MAGIC NOR Gates. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(8):2466–2476, August 2018.

[125] Seyedhamidreza Motaman and Swaroop Ghosh. Dynamic Computing in Memory (DCIM) in Resistive Crossbar Arrays. In *2018 IEEE 36th International Conference on Computer Design (ICCD)*, pages 179–186, October 2018.

[126] Dev Narayan Yadav, Phrangboklang L. Thangkhiew, and Kamalika Datta. Look-ahead mapping of Boolean functions in memristive crossbar array. *Integration*, 64:152–162, January 2019.

[127] Valerio Tenace, Roberto G. Rizzo, Debjyoti Bhattacharjee, Anupam Chattopadhyay, and Andrea Calimera. SAID: A Supergate-Aided Logic Synthesis Flow for Memristive Crossbars. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 372–377, March 2019.

[128] Barak Hoffer, Vikas Rana, Stephan Menzel, Rainer Waser, and Shahar Kvatinsky. Experimental Demonstration of Memristor-Aided Logic (MAGIC) Using Valence Change Memory (VCM). *IEEE Transactions on Electron Devices*, 67(8):3115–3122, August 2020.

[129] Rotem Ben-Hur, Ronny Ronen, Ameer Haj-Ali, Debjyoti Bhattacharjee, Adi Eliahu, Natan Peled, and Shahar Kvatinsky. SIMPLER MAGIC: Synthesis and Mapping of In-Memory Logic Executed in a Single Row to Improve Throughput. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2434–2447, October 2020.

[130] Chunyang Liu, Pengpeng Ren, Bo Zhou, Jianfu Zhang, Hui Fang, and Zhigang Ji. Investigation on the Implementation of Stateful Minority Logic for Future In-Memory Computing. *IEEE Access*, 9:168648–168655, 2021.

[131] Minhui Zou, Junlong Zhou, Jin Sun, Chengliang Wang, and Shahar Kvatinsky. Increase efficiency and lifetime of logic-in-memory by combining IMPLY and MAGIC families. *Journal of Systems Architecture*, page 102232, July 2021.

[132] Yuanfan Yang, Jimson Mathew, Dhiraj K. Pradhan, Marco Ottavi, and Salvatore Pontarelli. Complementary resistive switch based stateful logic operations using

material implication. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–4, March 2014.

[133] Anika Raghuvanshi and Marek Perkowski. Logic synthesis and a generalized notation for memristor-realized material implication gates. In *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 470–477, November 2014.

[134] Mehri Teimoory, Amirali Amirsoleimani, Jafar Shamsi, Arash Ahmadi, Shahpour Alirezaee, and Majid Ahmadi. Optimized implementation of memristor-based full adder by material implication logic. In *2014 21st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 562–565, Marseille, France, December 2014. IEEE.

[135] Anindita Chakraborty and Hafizur Rahaman. Implementation of combinational circuits via material implication using memristors. In *2016 IEEE Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER)*, pages 67–72, August 2016.

[136] Gina C. Adam, Brian D. Hoskins, Mirko Prezioso, and Dmitri B. Strukov. Optimized stateful material implication logic for three-dimensional data manipulation. *Nano Research*, 9(12):3914–3923, December 2016.

[137] Shokat Ganjeheizadeh Rohani and Nima TaheriNejad. An improved algorithm for IMPLY logic based memristive Full-adder. In *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–4, April 2017.

[138] Nuo Xu, Tae Gyun Park, Hae Jin Kim, Xinglong Shao, Kyung Jean Yoon, Tae Hyung Park, Liang Fang, Kyung Min Kim, and Cheol Seong Hwang. A Stateful Logic Family Based on a New Logic Primitive Circuit Composed of Two Antiparallel Bipolar Memristors. *Advanced Intelligent Systems*, 2(1), 2020.

[139] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny. VTEAM: A General Model for Voltage-Controlled Memristors. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(8):786–790, August 2015.

[140] Shahar Kvatinsky, Eby G. Friedman, Avinoam Kolodny, and Uri C. Weiser. TEAM: ThrEshold Adaptive Memristor Model. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(1):211–221, January 2013.

[141] Sergio Vinagrero Gutiérrez, Pietro Inglese, Giorgio Di Natale, and Elena-Ioana Vatajelu. Open Automation Framework for Complex Parametric Electrical Simulations. In *2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 132–135, May 2023.

[142] Pietro Inglese, Elena Ioana Vatajelu, and Giorgio Di Natale. On the Limitations of Concatenating Boolean Operations in Memristive-Based Logic-In-Memory Solutions. In *2021 16th International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–5, June 2021.

[143] National Institute of Standards and Technology. Data Encryption Standard (DES). Technical Report Federal Information Processing Standard (FIPS) 46 (Withdrawn), U.S. Department of Commerce, January 1977.

[144] Morris J. Dworkin, Elaine B. Barker, James R. Nechvatal, James Foti, Lawrence E. Bassham, E. Roback, and James F. Dray Jr. Advanced Encryption Standard (AES). *NIST*, November 2001.

[145] Joan Daemen and Vincent Rijmen. *The Design of Rijndael.* Information Security and Cryptography. Springer, Berlin, Heidelberg, 2002.

[146] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, volume 4727, pages 450–466. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[147] Sina Sayyah Ensan, Karthikeyan Nagarajan, Mohammad Nasim Imtiaz Khan, and Swaroop Ghosh. SCARE: Side Channel Attack on In-Memory Computing for Reverse Engineering. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(12):2040–2051, December 2021.

[148] Li-Wei Chen, Ziang Chen, Werner Schindler, Xianyue Zhao, Heidemarie Schmidt, Nan Du, and Ilia Polian. On Side-Channel Analysis of Memristive Cryptographic Circuits. *IEEE Transactions on Information Forensics and Security*, 18:463–476, 2023.

[149] Mohammad Nasim Imtiaz Khan, Shivam Bhasin, Alex Yuan, Anupam Chattopadhyay, and Swaroop Ghosh. Side-Channel Attack on STTRAM Based Cache for Cryptographic Application. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 33–40, November 2017.

[150] Giorgio Di Natale, Marie-Lise Flottes, and Bruno Rouzeyre. An Integrated Validation Environment for Differential Power Analysis. In *4th IEEE International Symposium on Electronic Design, Test and Applications (delta 2008)*, pages 527–532, January 2008.

[151] Kostas Nomikos, Athanasios Papadimitriou, Mihalis Psarakis, Aggelos Pikrakis, and Vincent Beroulle. Evaluation of Hiding-based Countermeasures against Deep Learning Side Channel Attacks with Pre-trained Networks. In *2022 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6, October 2022.

[152] Emmanuel Prouff and Matthieu Rivain. Masking against Side-Channel Attacks: A Formal Security Proof. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, Lecture Notes in Computer Science, pages 142–159, Berlin, Heidelberg, 2013. Springer.

Glossary

AD Address Decoder.

AES Advanced Encryption Standard.

AS Attack Scenario.

ASIC Application-Specific Integrated Circuit.

BIOS Basic Input-Output System.

BL Bitline.

CAD Computer-Aided Design.

CF Conductive Filament.

CIM Computation in Memory.

CMOS Complementary Metal-Oxide-Semiconductor Field-Effect Transistor.

CPU Central Processing Unit.

CRS Complementary Resistive Switching.

CSV Comma-Separated Values.

DES Data Encryption Standard.

DFA Differential Fault Attack.

DPA Differential Power Analysis.

DRAM Dynamic Random-Access Memory.

ECC Error Correcting Code.

EEPROM Electrically Erasable Programmable Read-Only Memory.

EMA Electromagnetic Analysis.

EPROM Erasable Programmable Read-Only Memory.

FA Fault Attack.

FeFET Ferroelectric Field-Effect Transistor.

FeRAM Ferroelectric Random-Access Memory.

HDD Hard Disk Drive.

HRS High Resistive State.

I/O Input/Output.

IP Intellectual Property.

ISA Instruction Set Architecture.

LIM Logic-In-Memory.

LRS Low Resistive State.

LUT Look-up Table.

MAC Multiply-Accumulate.

MOS Metal-Oxide-Semiconductor Field-Effect Transistor.

MOSFET Metal-Oxide-Semiconductor Field-Effect Transistor.

MRAM Magnetic Random-Access Memory.

MTJ Magnetic Tunnelling Junction.

NVM Non-Volatile Memory.

NVRAM Non-Volatile Random-Access Memory.

PAA Power Analysis Attack.

PCM Phase Change Memory.

PCRAM Phase Change RAM.

PIM Processing In Memory.

POST Power On Self Test.

PROM Programmable Read-Only Memory.

RAM Random-Access Memory.

ROM Read Only Memory.

RRAM Resistive Random Access Memory.

SA Sense Amplifier.

SDPA Simple Differential Power Analysis.

SEA Safe Error Attack.

SPA Simple Power Analysis.

SRAM Static Random-Access Memory.

SSH Secure Shell Protocol.

STT-MRAM Spin-Transfer Torque Magnetic Random Access Memory.

TEAM Voltage ThrEshold Adaptive Memristor.

USB Universal Serial Bus.

UV Ultraviolet.

VNC Virtual Network Computing.

VTEAM Voltage ThrEshold Adaptive Memristor.

WD Write Driver.

WL Wordline.