POLITECNICO DI MILANO

Facoltà di Ingegneria

Scuola di Ingegneria Industriale e dell'Informazione

Dipartimento di Elettronica, Informazione e Bioingegneria

Master of Science in

Electronic Engineering

Statistical model of resistive switching memory arrays for neural network hardware accelerators

Supervisor:

PROF. DANIELE IELMINI

Master Graduation Thesis by:

ARTEM GLUKHOV

Student Id n. 920857

Academic Year 2020-2021

ACKNOWLEDGMENTS

I would like to express my gratitude to my supervisor, Professor Daniele Ielmini, who guided and assisted me through the whole project.

I would also particularly like to thank Doctor Valerio Milo, for all the support that he gave me during the last months, helping me with the study, and offering insights into the project and my path.

I wish to sincerely thank IHP Microelectronics, Eduardo Perez, and Prof. Cristian Zambelli, for the experimental measurements that they kindly provided, highlighting the importance of this collaboration, and for their willingness to face and help me with the problems that gradually arose during my thesis path.

CONTENTS

Abstract xii
Estratto xiv
1 EMERGING MEMORY DEVICES AND ARCHITECTURES FOR NEUROMORPHIC COMPUTING 1
1.1 Introduction 1
1.2 Emerging non-volatile memory devices 5
1.2.1 Resistive-switching RAM 6
1.2.2 Phase Change Memory 9
1.2.3 Ferroelectric RAM 11
1.2.4 Spin Transfer Torque Magnetoresistive RAM 11
1.3 Crossbar arrays 13
1.4 In-Memory Computing 15
1.5 Resistive RAM for Neuromorphic computing implementation 17
1.5.1 Neural networks 17
1.5.2 Backpropagation training method for deep neural networks 19
1.5.3 Neural network implementation with crossbar arrays 21
2 RESISTIVE-SWITCHING RANDOM ACCESS MEMORY 25
2.1 Introduction 25
2.2 RRAM switching mechanism 26
2.3 Variability of RRAM programming 30
2.4 RRAM modeling 32
3 EXPERIMENTAL DATA ANALYSIS 34
3.1 Cell and array structure 34
3.2 Program and verify algorithms 35
3.2.1 ISPVA SET Algorithm 37
3.2.2 IGVVA SET algorithm 39
3.2.3 RESET algorithm 40

3.3 Experimental data 42
3.3.1 Datasets 43
3.3.2 C2C 45
3.3.3 D2D 46
3.3.4 Additional D2D 46
3.3.5 Stuck cells 47
3.4 After switching - End of algorithm distributions 47
3.5 Position and slope of the transient 51
3.6 Noise 52
3.7 Effect of programming steps width on programming time and variability 54
3.8 RRAM transition values extraction 55
4 STATISTICAL ANALYTICAL MODEL OF 4KBIT RRAM ARRAY 58
4.1 Introduction 58
4.2 Model description 59
4.2.1 Differential Equation 60
4.2.2 Integration 61
4.2.3 Program and verify algorithms 64
4.3 Parameters' tuning 65
4.3.1 Ea and A modulations 66
4.4 Fitting of median curves 67
4.5 Distribution fitting through Monte Carlo simulations 69
4.6 Unified model description 70
4.7 Reset transition modeling 74
4.7.1 Reset model rate equation 75
4.7.2 Reset model algorithm 75
4.7.3 Reset model fitting 76
4.8 Simulation container for neural network implementation 77
5 FULLY-CONNECTED NEURAL NETWORK IMPLEMENTATION USING 1T1R HAFNIUM-OXIDE-BASED RRAM ARRAYS 78
5.1 Introduction 78

5.2 Fully connected Neural Network 79
5.3 Incremental network quantization algorithm 81
5.4 Neural network accuracy 84
5.5 Variability, nonidealities, and impact on inference accuracy 86
5.5.1 Program and verify algorithms variability 87
5.5.2 Current consumption 89
5.5.3 Size of the network 89
6 CONCLUSION 91

BIBLIOGRAPHY 93

LIST OF FIGURES

Figure 1.1 Trend in microprocessor's scaling 3
Figure 1.2 Memory Hierarchy 5
Figure 1.3 emerging memories 6
Figure 1.4 RRAM device structure and operation 7
Figure 1.5 Unipolar vs Bipolar switching 8
Figure 1.6 PCM structure 9
Figure 1.7 FeRAM and FeFET example 10
Figure 1.8 STT-MRAM transitions and characteristic 12
Figure 1.9 Neural network and Crossbar Array 13
Figure 1.10 Linear solver 14
Figure 1.11 crossbar array implementations 16
Figure 1.12 Example of Deep Neural Network 19
Figure 1.13 Backpropagation algorithm for neural network 20
Figure 1.14 Implementation of differential array for positive and negative weights polarities 22
Figure 1.15 Weight update for a conductive cell 23
Figure 2.1 RRAM filament forming 27
Figure 2.2 Activation energy lowering 29
Figure 2.3 RRAM filament breaking 30
Figure 2.4 RRAM switching variability 31
Figure 2.5 Ecosystem of physical-based models 32
Figure 3.1 1T1R cell structure and IV characteristic 35
Figure 3.2 1T1R chip structure 36
Figure 3.3 Program and verify algorithms 37
Figure 3.4 ISPVA LRS and MOS IV characteristics 38
Figure 3.5 ISPVA100 LRS4 set transition 39
Figure 3.6 Reset after switching value 41

Figure 3.7 Analysis of finite $r_{ds,on}$ of the NMOS 42
Figure 3.8 IGVVA 4 levels 44
Figure 3.9 ISPVA and RESET 4 levels 45
Figure 3.10 Cycling of IGVVA100 46
Figure 3.11 8 LRS levels from D2D dataset and stuck cells distribution 47
Figure 3.12 IGVVA100 40μA - transition and after switching value 48
Figure 3.13 CDF of after switching values for IGVVA10 49
Figure 3.14 IGVVA100 40μA - after switching C2C distribution 50
Figure 3.15 C2C and D2D variability for three algorithms and four levels 51
Figure 3.16 Vmid transition CDF 52
Figure 3.17 Correlation between the median slope and the programmed accuracy 53
Figure 3.18 Analysis of read noise during IGVVA10 algorithm 54
Figure 3.19 IGVVA, read current vs RRAM extracted conductance 55
Figure 3.20 Analysis of the voltage partition across the 1T1R cell 56
Figure 4.1 Working point of 1T1R 61
Figure 4.2 Program and Verify algorithm basic principle 64
Figure 4.3 A and α parameters span for model definition 65
Figure 4.4 Ea and A modulation 66
Figure 4.5 Medians fitting 68
Figure 4.6 histograms of parameter variability 69
Figure 4.7 Median behavior of the model with unified parameters 70
Figure 4.8 Cumulative distribution of after switching values 71
Figure 4.9 C2C and D2D variability, data vs model 72
Figure 4.10 Vmid transition CDF model 73
Figure 4.11 Correlation between the slope and the programmed accuracy in simulation 74
Figure 4.12 Median value of Iread during reset 76

Figure 4.13 C2C and D2D variabilities, reset data and model comparison 77
Figure 5.1 MNIST dataset example 79
Figure 5.2 MNIST image resize example 80
Figure 5.3 Implementation of FC-NN on 64x64 crossbar arrays 82
Figure 5.4 Differential configuration and levels distributions 83
Figure 5.5 Incremental Network Quantization algorithm 84
Figure 5.6 Inference accuracy for different NN configurations 86
Figure 5.7 PDF of After Switching LRS 87
Figure 5.8 Effect of weights combination on inference accuracy 88

LIST OF TABLES

Table 4.1 Ode solvers speed comparison 63
Table 4.2 best fitting model parameters 67
Table 5.1 INQ training accuracy 85

LISTINGS

ACRONYMS

RRAM	Resistive Random-Access Memory

1T1R	one Transistor - one Resistor
1S1R	one Selector - one Resistor
IMC	In-Memory Computing
MVM	Matrix-Vector Multiplication
C2C	cycle-to-cycle
D2D	device-to-device
CF	Conductive Filament
CPU	Central Processing Unit
PCM	Phase-Change Memory
HRS	High Resistive State
LRS	Low Resistive State
TE	Top Electrode
BE	Bottom Electrode
STT-MRAM	Spin Transfer Torque Magnetic RAM
NMOS	N-channel Metal Oxide Silicon Transistor
ISPVA	Incremental Step pulse Program and Verify Algorithm
IGVVA	Incremental Gate Voltage and Verify Algorithm
FeRAM	Ferroelectric RAM
MNIST	Modified National Institute of Standards and Technologies database
AS	After Switching
EA	End of Algorithm
BEOL	Back End Of the Line
MIM	Metal-Insulator-Metal

OxRAM	Oxide-based RRAM
CBRAM	Conductive Bridge RRAM
SRAM	Static RAM
DRAM	Dynamic RAM
ASIC	Application Specific Integrated Circuit
CMOS	Complementary MOS
GPU	Graphic Processing Unit
DNN	Deep Neural Network
MTJ	Magnetic Tunnel Junction
ALD	Atomic Layer Deposition
FC-NN	Fully connected Neural Network
ANN	Artificial Neural Network
SNN	Spiking Neural Network
DNN	Deep Neural Network
MLC	Multi-Level Cell
STDP	Spiking Time Dependent Plasticity
SRDP	Spiking Rate Dependent Plasticity

ABSTRACT

For over 50 years, performance of computing systems has been growing exponentially, mostly driven by the CMOS technology scaling predicted by Moore's law.

However, over the past decade, this trend has been slowing down because of strict physical limitations.

At the same time, another limit is becoming more and more important: the memory wall bottleneck due to the traditional Von Neumann architecture.

This architecture is characterized by the physical separation between memory and computing unit, causing high latency and high energy consumption in data-extensive applications.

These limitations are pushing research towards exploring some innovative architectures that allow increased computing power and efficiency while reducing energy consumption, size, and costs.

Among the considered architectures, the concept of In-Memory Computing (IMC) has recently attracted great interest. Because of the compelling physical properties of emerging non-volatile memories based on innovative materials such as Resistive Random-Access Memory (RRAM), the IMC paradigm allows unifying the storage and elaboration of data inside the same nanometric-sized physical device, enabling the performance of complex algebraic operations such as Matrix-Vector Multiplication (MVM), leading the way towards new emerging application such as, for example, the hardware implementation of neural networks.

To build a hardware neural network, the IMC paradigm uses a matrix of RRAM memory elements in one Transistor - one Resistor (1T1R) configuration that can be selected through a gate line.

A vector of voltages, corresponding to the input of the network is supplied to the rows of the matrix, the synaptic weights are mapped on the conductance of

1T1R cells, and along the columns you will naturally find the product between the input vector and the weight matrix, which is the output of the synaptic layer.

The conductance programming of each 1T1R cell is done through SET and RESET processes. They consist respectively in forming (an increase of conductance) and breaking (a decrease of conductance) a conductive filament between the two metallic electrodes of the device, by applying a sequence of voltage pulses across the cell and to the gate terminal of the transistor.

This thesis focuses on the study of multilevel programming variability of a 4kbit Hafnium Oxide RRAM array under different program/verify algorithms, and proposes a statistical model able to accurately predict the cycle-to-cycle (C2C) and device-to-device (D2D) variability seen on the experimental data.

Chapter 1 focuses on introducing the challenges of the modern semiconductor industry and proposes a brief overview of the most important emerging memory technologies, such as RRAM, Phase-Change Memory (PCM), Spin Transfer Torque Magnetic RAM (STT-MRAM), and Ferroelectric RAM (FeRAM). Then, the main solutions adopted for the physical implementation of neural networks are addressed.

Chapter 2 focuses particularly on the Resistive switching memory, describing its working principles from a physical standpoint and introducing the typology of the algorithms used for its programming. Then it explains the sources of variability that are expected in a physical array of these devices.

Chapter 3 presents the analysis performed on experimental data, with particular importance given to the differences between the proposed algorithms for set and reset, and the extracted variability.

Chapter 4 focuses entirely on the development of a stochastic model for the SET algorithms.

Chapter 5 introduces a real-world application scenario for the RRAM crossbar arrays, the hardware implementation of a fully connected neural network, trained in recognizing images from a Modified National Institute of Standards and Technologies database (MNIST) dataset.

ESTRATTO

Per oltre 50 anni le prestazioni di calcolo sono cresciute esponenzialmente, grazie allo scaling della tecnologia CMOS predetto dalla legge di Moore. Nell'ultimo decennio però, il processo di miniaturizzazione ha subito un rapido rallentamento a causa del raggiungimento di dimensioni tali da scontrarsi con limiti fisici sempre più stringenti. Inoltre, siccome le applicazioni moderne necessitano di processare una quantità di dati sempre maggiore, a causa della separazione fisica tra unità di calcolo e unità di memoria tipica dell'ormai universalmente utilizzata architettura di Von Neumann, i sistemi odierni sono caratterizzati da un enorme consumo di potenza e da una sempre maggiore latenza.

Queste problematiche hanno spinto la ricerca verso lo studio di architetture innovative che consentano di continuare a incrementare la potenza e l'efficienza di calcolo riducendo ulteriormente il consumo energetico, le dimensioni e i costi.

Tra i vari schemi presi in considerazione, il concetto di calcolo in memoria (IMC) sta attirando grande interesse. Grazie alle interessanti proprietà fisiche che contraddistinguono le memorie emergenti non-volatili basate su materiali innovativi, come per esempio le memorie a switching resistivo (RRAM), il paradigma IMC prevede di combinare la funzionalità di memorizzazione e di elaborazione in un unico dispositivo fisico. Ciò permette di eseguire complesse operazioni matriciali e apre la strada verso nuove applicazioni emergenti come l'implementazione di reti neurali su hardware.

Per realizzare una rete neurale su hardware, il paradigma IMC prevede tipicamente l'utilizzo di una matrice di elementi di memoria RRAM in configurazione one Transistor - one Resistor (1T1R), ovvero selezionabili tramite un transistore in serie al dispositivo. Lungo le righe della matrice si applicano le tensioni che descrivono i dati forniti in ingresso alla rete, la conduttanza delle celle 1T1R rappresenta i pesi delle connessioni sinaptiche tra i neuroni appartenenti a strati successivi

della rete, e lungo le colonne è possibile leggere le uscite dello strato sinaptico in corrente/tensione.

La programmazione della conduttanza delle celle 1T1R si ottiene attraverso i processi di SET e RESET. Essi consistono rispettivamente nella formazione (aumento di conduttanza) e rottura (riduzione di conduttanza) di un filamento conduttivo tra i due elettrodi metallici del dispositivo tramite l'applicazione di una sequenza di impulsi di tensione ai capi della cella e al terminale di gate del transistore. Ad oggi sono già stati proposti diversi algoritmi di programmazione che permettono di avere il controllo abbastanza preciso di più livelli conduttivi in un singolo dispositivo RRAM.

Per determinare i valori di conduttanza dei pesi sinaptici della rete neurale tipicamente si effettua una procedura di apprendimento, detta training, che consiste nel fornire ripetutamente in ingresso un set molto ampio di dati e nel minimizzare l'errore all'uscita della rete tramite una tecnica di ottimizzazione detta propagazione all'indietro o backpropagation. Una volta conclusa la fase di training, la rete sarà in grado di riconoscere la classe associata a nuovi dati mai visti prima grazie all'avvenuto adattamento dei pesi sinaptici.

Per facilitare lo studio e la progettazione hardware di reti neurali è quindi importante disporre di un modello analitico basato sulla fisica del dispositivo di memoria capace di riprodurre accuratamente il funzionamento delle singole celle, tenendo conto dell'impatto di non idealità deterministiche e statistiche della matrice, e rimanendo compatto dal punto di vista computazionale.

Questo lavoro di tesi si concentra sull'analisi degli algoritmi di programmazione di celle 1T1R e propone un modello analitico compatto in grado di simularne accuratamente il funzionamento.

Il primo capitolo riassume brevemente lo scenario tecnologico in cui ci troviamo e introduce il motivo per la ricerca di architetture alternative. Dopo una breve presentazione delle principali tecnologie di memorie emergenti, viene illustrata l'architettura cross-point focalizzandosi in particolare sui dispositivi RRAM.

Il capitolo 2 si concentra sulle memorie non volatili a filamento resistivo, analizzando le procedure di formazione e rottura del filamento resistico e le cause primarie di variabilità. Inoltre vengono brevemente descritti alcune tipologie di modello già esistenti in letteratura.

Il capitolo 3 tratta dell'analisi svolta sui dati sperimentali, ponendo particolare importanza alla differenza tra i diversi algoritmi e i vari livelli conduttivi e a come viene influenzata la variabilità ciclo-ciclo e device-device.

Il capitolo 4 presenta un modello analitico statistico per i diversi algoritmi di SET e di RESET, e un confronto tra le simulazioni e i dati sperimentali focalizzandosi sui diversi tipi di variabilità.

Il capitolo 5 propone un esempio di applicazione reale di utilizzo di matrici cross-bar di memorie resistive con l'implementazione hardware di una rete neurale allenata al riconoscimento e la classificazione di immagini tratte dal dataset MNIST.

1

EMERGING MEMORY DEVICES AND ARCHITECTURES FOR NEUROMORPHIC COMPUTING

This chapter presents the challenges that the semiconductor industry is facing nowadays. As data-intensive applications such as machine learning or the Internet of Things begin to increase, current computing systems and architectures begin to show their limits in terms of power efficiency, latency, and scalability.

One of the most promising solutions is represented by emerging memory technologies, that thanks to their unique physical properties, can overcome such limitations by replacing current memory technology and by opening the way for a new computing paradigm, called neuromorphic computing, which aims at replicating more energy-efficient architecture, such as a human brain, capable in performing complex operations by consuming an extremely low amount of energy and at low frequencies. After an exhaustive introduction of the context, an overview of the available memory technologies is proposed, along with the architecture where they can be assembled.

1.1 INTRODUCTION

Over the past 50 years, the growth of computing power in electronic systems followed almost precisely the prediction made by Gordon Moore in 1965 [1]. He claimed that the number of transistors per chip would double circa every two years. This claim, known as Moore's law, was for many decades a path and a goal for engineers and manufacturers, who pushed forward the knowledge and technology to accomplish this incredible growth rate, mainly by scaling the Complementary MOS (CMOS) technology.

Miniaturization of electronic devices based on silicon technology resulted in an exponential growth of device density and enabled an exponential increase of computing power and operating frequencies, giving birth to the concept of personal computers, mobile phones, wearable gadgets, and so on. The trend initially followed Dennard's geometrical scaling rule [2, 3], also called *constant field scaling*, in which the transistor's gate length, gate width, gate oxide thickness, and supply voltage were reduced all by the same scale factor. Then thanks to the technological advances in the process integration, a new scaling era begun, called effective scaling, during which unconventional materials such as high-k dielectric and unconventional non-planar structures such as FinFETs were introduced to further scale the effective parameters [4].

However, several problems arose in recent years, causing the slow down of this trend and giving researchers and manufacturers complex challenges to solve to continue supply the market with faster and smaller devices at every generation.

With the miniaturization of the CMOS technology, leakage currents limit the further downscaling of the threshold voltage of MOSFET devices, hindering the supply voltage and size scaling in digital CMOS devices. In addition, the power density of modern CMOS-based microprocessors reached values up to $100W/cm^2$ [5], limiting further operating frequency increase for such devices due to excessive heating of the chip. This translates into a frequency plateau referred to as Heat wall [6], which is difficult to overcome, and since the early $2000s$, the maximum frequency of CPUs remained almost unchanged (Figure 1.1).

On the other hand, another limit to modern computing systems is the so-called memory wall, or von Neumann bottleneck [8]. It is naturally caused by the physical separation between the processing unit and the memory unit typical of any modern computer architecture, such as von Neumann architecture [9]. The continuous increase in processing speed and a slower increase in memory performance translates into an increase in latency with every technological generation. Statistically, for every handful of CPU instructions, there is the need to access the memory to read or write data, and nowadays this can use up to thousands of CPU cycles during which the processing unit has nothing to do but wait for the memory operation to complete [8]. To partially mitigate this bottleneck different solutions

1 https://creativecommons.org/licenses/by/4.0/

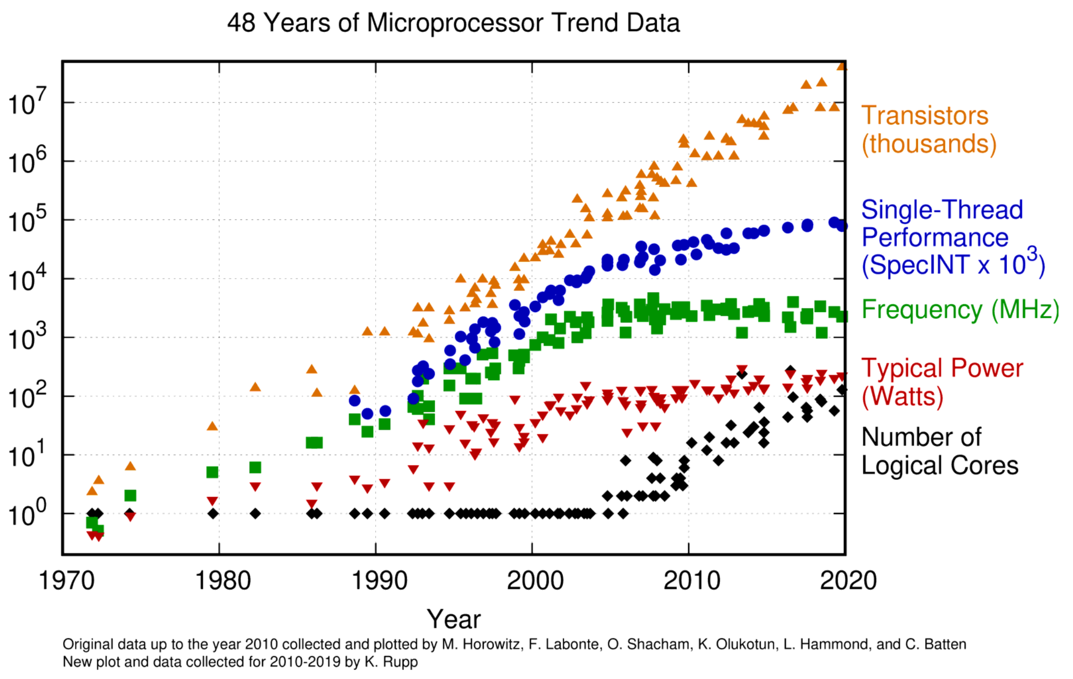

Figure 1.1: Trend of microprocessor's technological scaling over the last 48 years. (Orange) The transistor's count following an exponential increase predicted by G.E. Moore in 1965; (Blue) Increasing performance of a single processing core. It is noticeable the reach of an asymptotic value, with the increase slowing down over the last years; (Green) Frequency reached a plateau around mid-2000 due to the Heat wall problem; (Red) Total power consumed by a single microprocessor chip; (Black) the number of processing cores integrated inside a single CPU die, increasing since 2005. Reprinted from [7], licensed under CC BY 4.0[1]

were proposed, one of them consisting in exploiting the memory hierarchy [10]. By moving the fastest memory closer to the Central Processing Unit (CPU) and using it to store the most commonly accessed information, i. e. to exploit the principle memory locality [11], the CPU has more chances to find the data it is looking for and to access it within few CPU cycles. This is the principle of caching and it allowed to push forward the reach of the bottleneck for several years, but it still is not the solution to completely solve the problem. Since it is not possible to store all the needed data inside the small cache memory, every now and then a *cache miss* occurs, which means that a slower and further memory, such as DRAM, has to be accessed or even worse, a storage memory like a mechanical HDD or an SSD device, whose access times can be in the order of tens of thousands of CPU cycles (Figure 1.2).

Static RAM (SRAM) is a volatile memory device that gives the fastest read and write speed, in the order of 1-10ns, yet it is quite large, occupying a cell area larger than $100F^2$, with F being the minimum lithographic feature size (Figure 1.3). For this reason, SRAM is used mostly for cache memory and it is integrated directly into the CPU die. Its capacity typically is in the order of few Megabytes. Moving further away from the CPU core, the next memory in the hierarchy that is usually found is the Dynamic RAM (DRAM). It is a volatile type of memory but as opposed to the SRAM, DRAM occupies a much smaller area, typically in the order of $6F^2$, but features a slower read and write performance. This allows for a much higher density enabling chips in the order of few Gigabytes. Further down the hierarchy path, it is common to find a non-volatile memory for the storage of the operating system and the user data. This memory can be made of solid-state devices, such as Flash memory, eMMC, or can be mechanical memory (HDD, magnetic tape, compact disc,...). These devices are much slower than the non-volatile memory, but enable really big storage sizes, in the order of up to several Terabytes, and can retain information for many years.

Despite the architectural effort made in delaying the memory wall problem, unfortunately, it is still one of the major problems of modern computing systems.

Several studies were made on the emerging memory technologies, trying to find a new device capable of the SRAM performance while maintaining a high bit-density typical of flash technology. This led to the investigation of a new type of device that presents itself in a 2-terminal structure, exhibits non-volatile properties along with high speed, high density, and CMOS process compatibility. It is called *memristor*.

As opposed to flash, SRAM, and DRAM technologies, which store information in form of electrical charge, a memristor stores information by changing the physical properties of the device. However, although some commercially available memory devices based on these emerging technologies were produced [12], due to higher cost and small benefits compared to already existing solutions, while still not

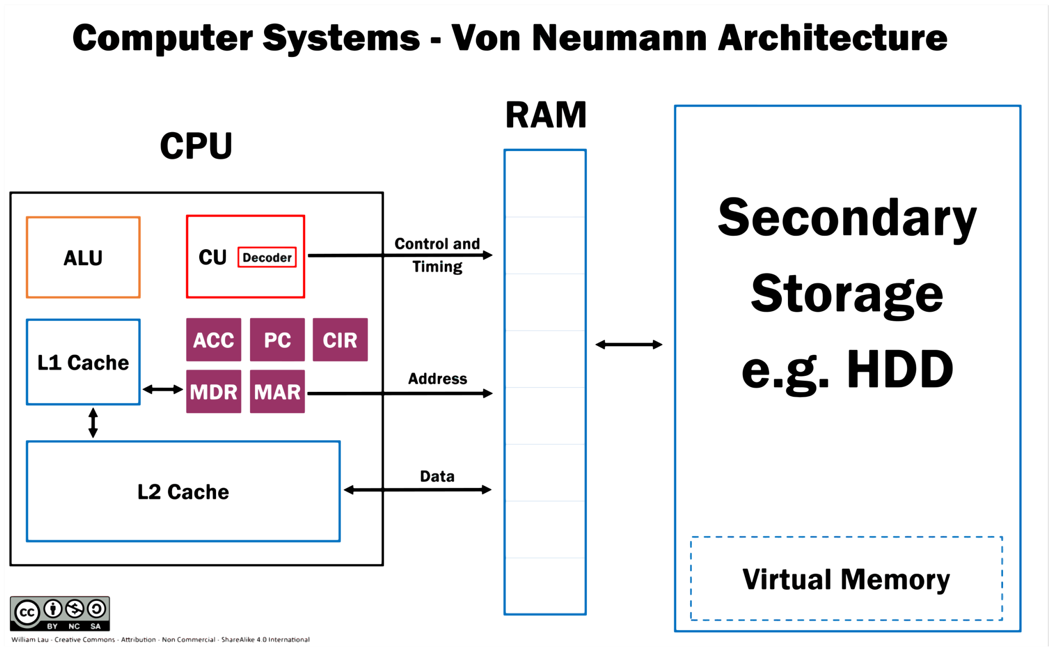
Computer Systems - Von Neumann Architecture
CPU
RAM
ALU
CU
Decoder
Control and Timing
ACC
PC
CIR
L1 Cache
MDR
MAR
Address
L2 Cache
Data
Secondary Storage e.g. HDD
Virtual Memory
William Lau - Creative Commons - Attribution - Non Commercial - ShareAlike 4.0 International

Figure 1.2: From left to right: two levels of cache memory, usually manufactured in SRAM technology and integrated inside the same package of the CPU; RAM, a volatile DRAM memory in the order of few Gigabytes; storage devices such as mechanical Hard Disk Drive (HDD) or Solid State Drive (SSD). Reprinted from Wikipedia, authored by William Lau, licensed under CC BY-SA 4.0[2]

overcoming the memory bottleneck, they did not have the hoped-for commercial success.

1.2 EMERGING NON-VOLATILE MEMORY DEVICES

Figure 1.3 highlights some of the main emerging non-volatile memory technologies and compares them to the traditional devices, both volatile and non-volatile such as DRAM and flash memory respectively. The emerging ones, namely Spin Transfer Torque Magnetic RAM (STT-MRAM), Ferroelectric RAM (FeRAM), Phase-Change Memory (PCM) and Resistive Random-Access Memory (RRAM), are all comparable in terms of speed, size, and operating voltages. They are suitable to

2 https://creativecommons.org/licenses/by-sa/4.0/legalcode

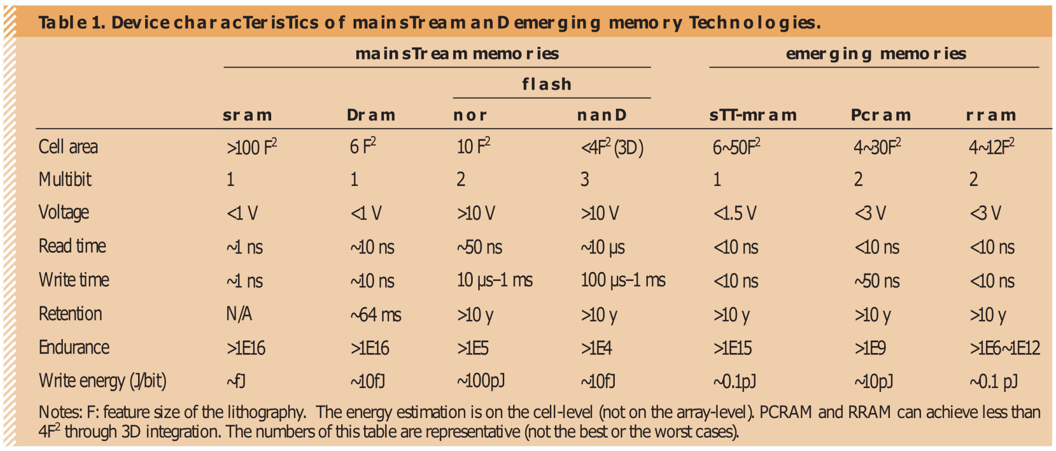
Table 1. Device characteristics of mainstream and emerging memory technologies.

	Mainstream memories				Emerging memories		
			Flash				
	SRAM	DRAM	NOR	NAND	STT-MRAM	PCRAM	RRAM
Cell area	>100 F^2	6 F^2	10 F^2	<4F^2 (3D)	6~50F^2	4~30F^2	4~12F^2
Multibit	1	1	2	3	1	2	2
Voltage	<1 V	<1 V	>10 V	>10 V	<1.5 V	<3 V	<3 V
Read time	~1 ns	~10 ns	~50 ns	~10 μs	<10 ns	<10 ns	<10 ns
Write time	~1 ns	~10 ns	10 μs–1 ms	100 μs–1 ms	<10 ns	~50 ns	<10 ns
Retention	N/A	~64 ms	>10 y	>10 y	>10 y	>10 y	>10 y
Endurance	>1E16	>1E16	>1E5	>1E4	>1E15	>1E9	>1E6~1E12
Write energy (J/bit)	~fJ	~10fJ	~100pJ	~10fJ	~0.1pJ	~10pJ	~0.1 pJ

Notes: F: feature size of the lithography. The energy estimation is on the cell-level (not on the array-level). PCRAM and RRAM can achieve less than 4F^2 through 3D integration. The numbers of this table are representative (not the best or the worst cases).

Figure 1.3: Table of memory device characteristics and performance metrics, comparison between mainstream CMOS-based memories and emerging resistive-switching memory technologies. Reprinted with permission from [13]. Copyright 2016, IEEE.

fill the performance gap between DRAM and flash, being at the same time non-volatile and having high read and write speeds (in the order of up to $\sim 5 - 10ns$) comparable to the traditional volatile RAMs. Moreover, they use different materials and can be scaled down much more than flash memory, and can be monolithically integrated on-chip with the CPU cores, supporting the Back End Of the Line (BEOL) process. Next, some of the main typologies of emerging memory will be briefly introduced.

1.2.1 *Resistive-switching RAM*

Due to their promising characteristics, Resistive Random-Access Memory (RRAM) devices are receiving widespread interest as future non-volatile and dynamic memory technologies. They are promising in terms of BEOL integration and miniaturization, and offer fast read/write operations allowing also for analog multi-level programming. By stacking these devices in multiple layers 3D structures, it is possible to create high-density multi-level memory.

RRAM usually are composed of a sandwiched insulating layer (for example a transition metal oxide MeO_x) between two metal electrodes. The device is initially

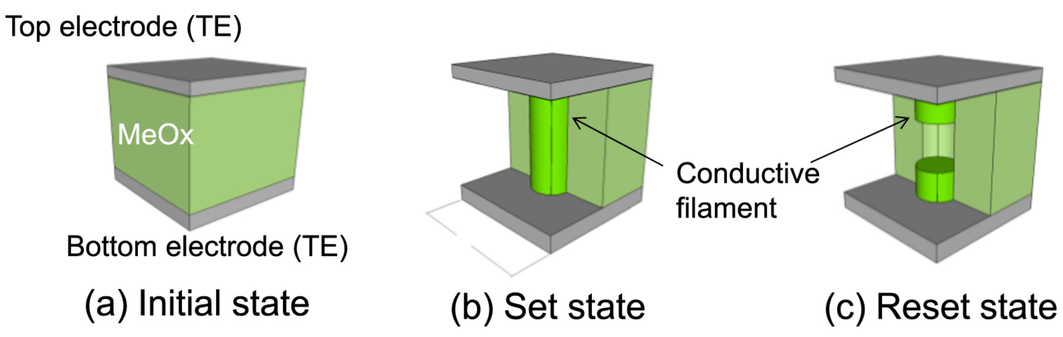

Figure 1.4: RRAM device structure and operation. (a) The pristine device consists of a Metal-Insulator-Metal (MIM) stack, where the insulator typically is a binary metal oxide. (b) After the electroforming, a Conductive Filament (CF) connects the Top Electrode (TE) and the Bottom Electrode (BE) resulting in a Low Resistive State (LRS), or set state. (c) The CF can be disrupted by a reset operation, restoring a High Resistive State (HRS) condition, or reset state. Reprinted with permission from [14]. Copyright 2014, John Wiley & Sons, Inc. License number 5140710690589.

subject to an operation called electroforming, in which by applying a polarization voltage across the electrodes, a Conductive Filament (CF) is formed through the isolating layer by moving charged ions or oxygen vacancies, connecting the two electrodes in a Low Resistive State (LRS). The reset operation will break this CF returning the device into a High Resistive State (HRS) (Figure 1.4). By alternating set and reset operations the CF can be formed and disrupted multiple times [15] with relatively high endurance (more than 10^7 cycles [4]).

There are two main methods of resistive switching behavior, namely unipolar switching and bipolar switching. The first one is characterized by having both the set and the reset programming voltages of the same polarity, it is based on thermally accelerated Red-Ox transitions [16], and despite allowing for a simpler peripheral circuit, it usually shows much more variability and less endurance retention compared to bipolar switching RRAM devices [17]. In fact, in bipolar devices, the same defects are re-injected into the depletion region during the next set operation, maintaining the total number of defects over the cycling [15, 18] (Figure 1.5a). This last type is the most common one, it is based on ion migration assisted by temperature and electric field [18] (Figure 1.5b). The bipolar switching

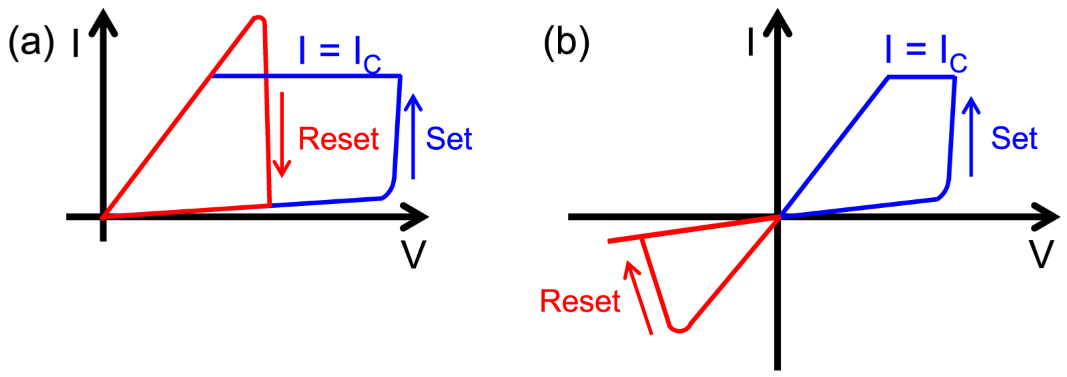

Figure 1.5: (a) A typical set/reset IV characteristic in a unipolar switching device. (b) Bipolar switching IV characteristic, set is given by applying a positive voltage while reset switching is performed by negatively polarizing the device. Reprinted with permission from [14]. Copyright 2014, John Wiley & Sons, Inc. License number 5140710690589.

devices can be further subdivided into two categories, Oxide-based RRAM (OxRAM) and Conductive Bridge RRAM (CBRAM):

- OxRAM: Electrodes are typically made of transition metal (such as Ti or Hf), and the insulating layer by a transition metal oxide (for example HfO_x). The charged ions are usually oxygen vacancies moved through the insulator by means of an electric field.
- CBRAM: Electrodes are made of active metal (Ag or Cu) and the charged ions are cations that pass through a chalcogenide material.

The main difference between the two types of devices is the resistive window between the high and low resistive states, with CBRAM having a span of circa 10^4 while OxRAM showing a window in the order of $10 - 100$ between the two states. Apart from that, both device types show similar electrical and functional properties and are similar in terms of performance [19, 20]. RRAM in general is convenient in terms of cycle endurance, power consumption, speed, and miniaturization, with complete CMOS BEOL compatibility. Despite all these positive properties they suffer from random telegraph noise (RTN) [21] and a severe variability and relaxation caused by time and temprature fluctuations, reducing their reliability and undermining their use in precision-requiring applications.

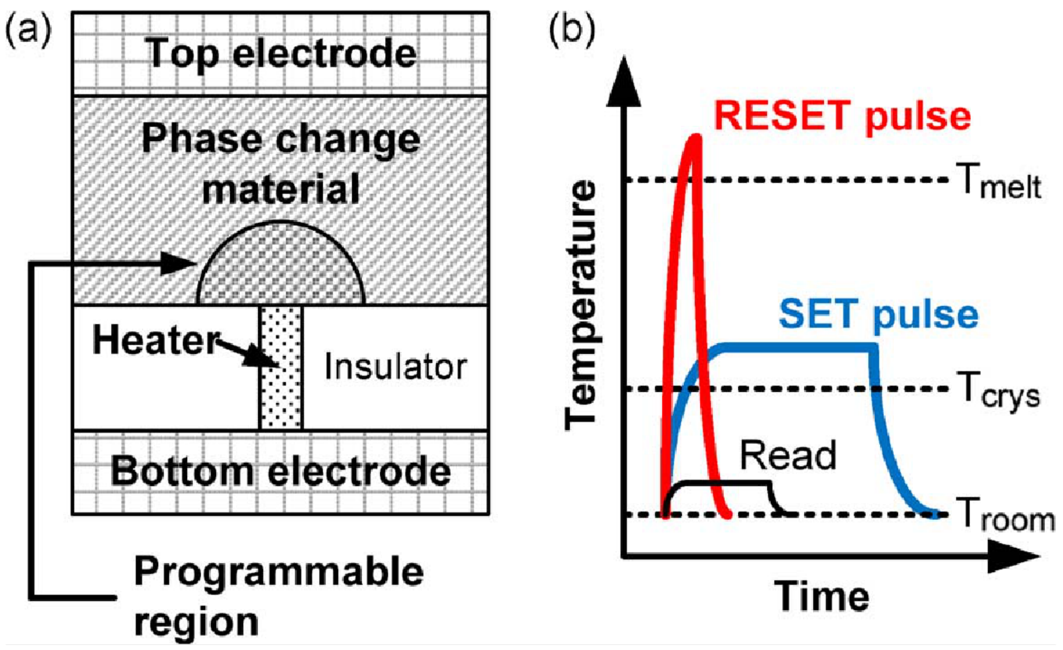

Figure 1.6: (a) Cross-section of a mushroom structure in a PCM device, composed by a pillar-like bottom electrode that acts as a heater, and by a hemispherical mushroom-shape for the amorphous material development. (b) Temperature evolution in time for the set, reset, and read pulses. To obtain an amorphous reset state, a very short but high amplitude pulse is applied to the top electrode, leading to an overcome of the melting temperature. The set state instead, is achieved by a longer, lower-amplitude pulse that leads to crystallization. Reprinted with permission from [22]. Copyright 2010, IEEE.

1.2.2 *Phase Change Memory*

Phase-Change Memory (PCM) devices rely on a phase transformation of an active material, a chalcogenide such as $Ge_2Sb_2Te_5$ (GST) [22]. The active material can switch between the high resistive - amorphous phase and the low resistive - crystalline phase. In the case of reset transition, i. e. from LRS to HRS, by applying a short and high amplitude electrical pulse the chalcogenide material melts and the resulting amorphous phase has a high resistivity. The crystalline phase can be obtained back again by the application of a set pulse, a longer and lower-amplitude pulse, that causes the fast crystallization thanks to Joule heating (Figure 1.6b). Differently from most RRAM devices, PCMs usually are operated with unipolar pulses. The cell can be read through a sufficiently small current/voltage so that

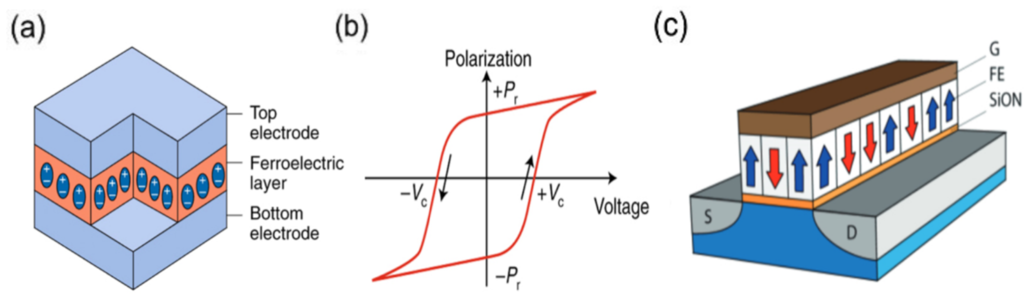

Figure 1.7: (a) sandwich MIM structure of a FeRAM device, consisting of a ferroelectric layer between two metallic electrodes. (b) Hysteresis cycle of polarization - voltage (PV). (c) FeFET structure example with the ferroelectric layer in place of the gate oxide. Reprinted with permission from [25]. Copyright 2019, IOP Publishing Ltd.

the programmed resistive state is not perturbed. The most common PCM device is a mushroom-type, depicted in figure 1.6a. It consists of two metal electrodes separated by a pillar-like heater element surrounded by insulating material, and a layer of phase-change material in which the amorphous state creates a mushroom-shaped structure.

PCM devices present a high resistance window between their LRS and HRS states, enabling for multi-level operation, i. e. programming and storing a wide range of different resistive values. they also show high endurance ($> 10^{12}$ cycles, figure 1.3) and long data retention, up to 10 years even at high temperatures [23]. A drawback of PCM devices is that the set transition is much slower than the reset transition, and it is a limiting factor to the write speed of this kind of technology. Like RRAM, also PCM devices suffer from state relaxation, particularly in the amorphous phase. Such drift consists in the structural relaxation due to temperature-activated atomistic rearrangement that leads to an increase of the electrical resistivity over time [24]. Unfortunately, it is not a controllable effect and it might undermine the multilevel operation of the device.

1.2.3 *Ferroelectric RAM*

Ferroelectric RAM (FeRAM) devices typically present themselves in a MIM structure with a ferroelectric material as an insulator (Figure 1.7a). By polarizing the insulator the memory state is determined. Usually, doped-HfO_2 [26] or a perovskite material ($PbZrTiO_3$ or $SrBi_2Ta_2O_9$) [27] are employed. Applying an external voltage, the orientation of the electric dipoles inside the ferroelectric materials is imposed, and it is maintained after the removal of the external bias. To trigger the polarization switching it is necessary to reach the so-called coercive voltage (V_c), resulting in a hysteresis cycle (Figure 1.7b). Unfortunately, the readout of the cell state destroys the stored value, increasing the complexity of the readout circuitry and costs, in terms of time and energy consumption. Despite the non-volatility of the technology and the fast read/write operations [28], in the order of $\sim 10ns$ and comparable with other emerging memories, FeRAM devices are not suitable for neuromorphic computing, because of the destructive readout and the absence of multi-level programming capabilities.

A proposed solution to overcome the destructive readout relies on employing the ferroelectric device in a FeFET structure [29, 30], as shown in figure 1.7c. A FeFET is essentially a MOS transistor with a ferroelectric layer as the gate dielectric. By switching the polarization state it is possible to tune the threshold voltage resulting in a change of the channel resistance, and thus enabling a non-destructive readout operation. A drawback of this structure is the presence of three terminals instead of just two, increasing the complexity of routing and the occupied size of the device.

1.2.4 *Spin Transfer Torque Magnetoresistive RAM*

The Spin Transfer Torque Magnetic RAM (STT-MRAM) is a memory device based on Magnetic Tunnel Junction (MTJ) and made of two thin ferromagnetic layers, typically of $CoFeB$, separated by a tunnel oxide barrier as shown in figure 1.8a. The magnetization direction of one ferromagnetic layer is fixed and used as a reference,

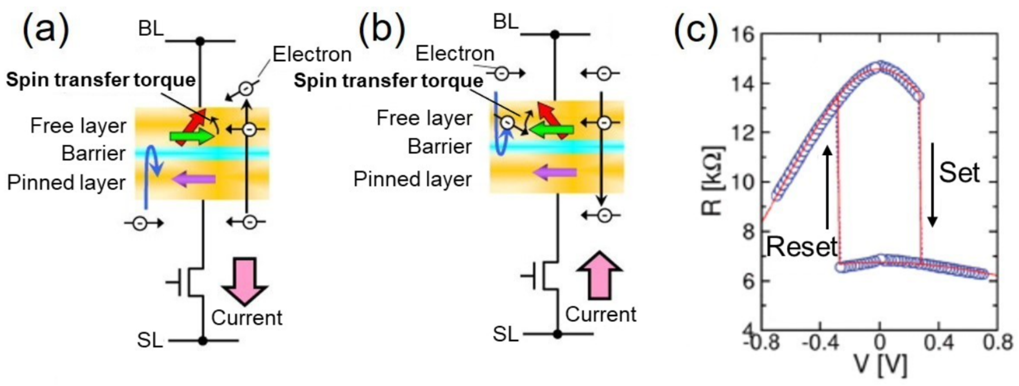

Figure 1.8: (a) Set transition of STT-MRAM device. Electrons with the same spin polarization to the PL reach the FL and trigger the switching; (b) Reset transition of STT-MRAM. Electrons with the opposite polarization to the PL are reflected back to FL causing the switching to HRS. Reprinted with permission from [31]. Copyright © 2011 Elsevier Ltd. All rights reserved. License number 5093291368901; (c) R-V characteristic of an STT-MRAM device. Noteworthy is the factor two between set and reset states, and the abrupt transitions during both set and reset. Adapted with permission from [32].

it is called pinned layer (PL), while the other layer has a magnetization direction that is free to rotate under the influence of an external bias, and it is called free layer (FL). The resistance depends on the mutual orientation of the two magnetic layers. When the two layers have the same magnetic polarization direction (parallel configuration - P), the resistance of the MTJ is in its low state. Otherwise, when the two layers have antiparallel magnetization (anti-parallel configuration - AP), the resistance has a higher value. During the set transition (from AP to P), electrons flow from the PL to the FL, and only the ones with the same spin direction as the pinned layer can pass the barrier (Figure 1.8b). When the current exceeds the threshold value, the polarization of the free layer is switched, bringing the device to LRS. On the opposite transition, from P to AP, equivalent to the reset, electrons flow from FL to PL, and the ones having an opposite spin direction compared to the pinned layer are reflected and re-injected into the free layer, triggering the magnetic switching to HRS [31].

The ratio between the high resistive state and the low resistive state is much lower than in RRAM or in PCM, and it is in the order of 2, making this type of

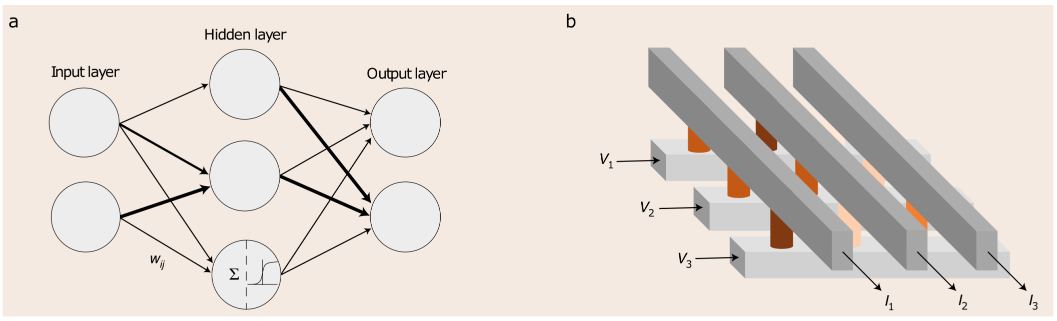

Figure 1.9: (a) Schematic representation of a fully connected neural network. Each neuron performs the sum of the synaptic signals and passes the result through a nonlinear activation function; (b) Schematic representation of a 3x3 crossbar array performing the matrix-vector multiplication $I = GV$. Reprinted with permission from [35]. Copyright © 2019, Springer Nature Limited. License number 5093580908010.

memory not suitable for IMC applications. Despite the low resistive window, they feature an incredibly high endurance ($> 10^{15}$) since no mechanical movement is performed during switching, and very fast switching times [33], as shown in figure 1.8c, so they are interesting in the role of fast caching memory.

1.3 CROSSBAR ARRAYS

Except for FeFETs, which are three-terminal devices, all other technologies presented in section 1.2 are two-terminal devices. A feature that allows them to be integrated into a compact architecture, known as the crossbar array. A crossbar array is a matrix of metal vertical rows and horizontal columns placed on separate planes, interconnected through a memristive device, as depicted in figure 1.9b. Such configuration allows building highly dense structures, with a single device requiring an area of just $4F^2$ [4]. This device density is not reachable when using CMOS technology. It is possible to further increase the device density by stacking these arrays on top of each other creating a 3D structure. In this way, the maximum density reachable is $\frac{4F^2}{N}$, being F the lithographic feature and N the number of stacked layers [34].

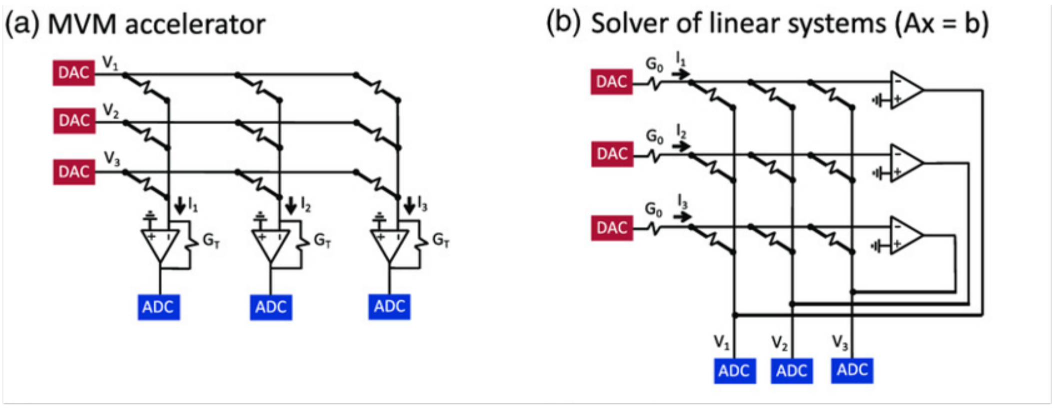

Figure 1.10: Schematics of two possible applications of a cross-point array: a circuit able to solve a linear system (a) and a solver of linear regression (b). Adapted with permission from [36]. Licensed under CC BY 4.0

By simply following Ohm's law and Kirchhoff's current law, this array structure performs a Matrix-Vector Multiplication (MVM) between the matrix G of programmable conductances and the vector V of applied voltages, giving as the output at the columns a vector of currents $I = GV$ in just a single step (Figure 1.9b).

The ability to perform instantly this operation is a key factor that enables this structure to be the perfect candidate for the implementation of In-Memory Computing (IMC) applications such as machine learning with Artificial Neural Network (ANN) [35], in which each neuron of a synaptic layer is linked to all the neurons of the next layer through a matrix of synaptic weights that can be encoded into the conductance values of the crossbar array (Figure 1.9a).

However, this is just one of the many possible applications for these structures. Integrating the cross-point arrays with adequate peripheral circuitry and exploiting a feedback loop it is possible to obtain, for example, linear systems or linear regression solvers [36] (Figure 1.10).

Unfortunately, various non-idealities affect the performances of this kind of structure, imposing limitations on the array size and the accuracy of the calculations. For example, at the device level, a conductance drift might happen, or imprecise programming causing a variability that lowers the overall precision of the MVM. At array level, instead, two effects are present: the IR drop, i. e. ohmic drop caused by finite conductance of row's and column's wires, and sneak paths, that is a current flowing through unwanted parallel paths resulting in spurious resistive partitions that pollute the result.

While IR drop can be mitigated by using smaller arrays or higher resistive values, sneak paths can be limited by placing in series to the memory element a one-way selector, creating one Selector - one Resistor (1S1R) configuration (Figure 1.11c). The purpose of the selector is to act like a diode and let the current flow only in a single direction when a certain threshold voltage across the cell is reached.

Another technical difficulty with crossbar arrays is that devices must be selectively programmed, and this can be done by exploiting different schemes. Taking as an example the simple circuit in figure 1.11a, if we want to program only the conductance G_{21}, and we apply voltage V_1 to the leftmost column, both G_{11} and G_{21} will sense the voltage drop. A possible solution would be to use the $V/2$ bias (figure 1.11b), by applying $V/2$ and $-V/2$ to the desired row and column. This way, only the interested device will have the full V drop across its terminals [34].

Another solution, as already discussed earlier, would be to use a selector in series to the device, forming a 1S1R configuration (Figure 1.11c). A more radical approach is to use a transistor in series to the memory device, in one Transistor - one Resistor (1T1R) configuration (Figure 1.11d). While this seems to be the best solution for this problem, it adds the necessity of a third line for the control of the gate voltage, thus it increases the complexity and the size of both the array and the peripheral circuit.

1.4 IN-MEMORY COMPUTING

Another strategy that is being explored in order to continue the performance growth trend despite the technological difficulties, is the exploit of parallel processing. Since it is unfeasible to increase operating frequencies over a certain value mainly due to the already discussed heat wall, around mid-2000 manufacturers began to integrate more computing cores inside the same CPU die, enabling more operations to be performed in the same clock cycle. This allowed increasing

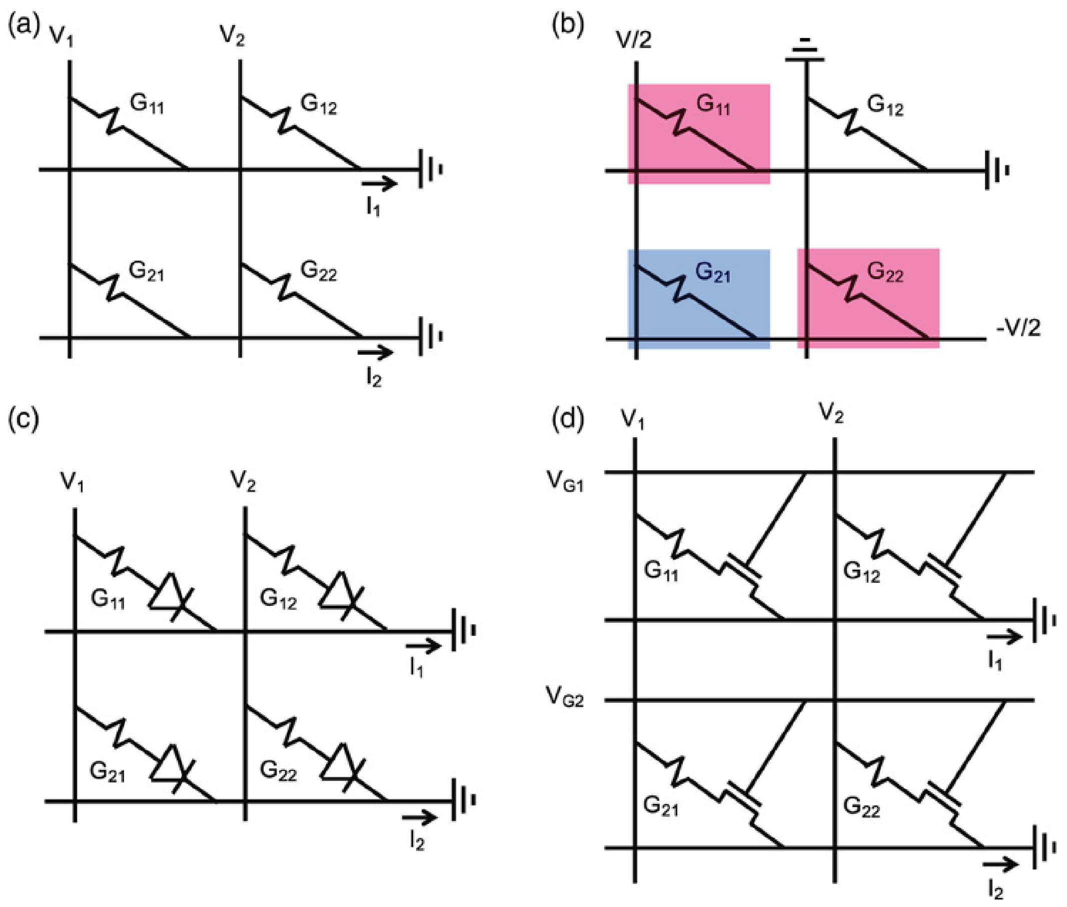

Figure 1.11: Different implementations of the cross-point arrays. (a) represents a traditional 1R structure, (b) shows the $V/2$ scheme applied to it. (c) represents 1S1R implementation while (d) represents 1T1R configuration. Reprinted with permission from [36], licensed under CC BY 4.0[3]

the processing performance of many applications while keeping the same power density and same frequencies as before (Figure 1.1).

In parallel to the already mentioned workarounds, research is studying new architectures and alternative computing paradigms that can overcome both the scaling limitations and architectural bottlenecks.

One of the paths of research is directed towards brain-inspired computing and In-Memory Computing (IMC), by exploiting properties of the already introduced emerging memory devices. In particular, IMC aims at unifying both the computing unit and the storage of data into a single physical device, enabling the acceleration of complex algebraic operations such as matrix inversion [37] and MVM by computing the operations directly in-situ, eliminating the burden of the von Neumann bottleneck.

Many modern applications that are now widely explored, such as, for example, image classification or voice recognition, are achieved by running multi-layer neural networks [38], such as Deep Neural Network (DNN), which are capable of extracting useful information from huge datasets using deep learning techniques. Nowadays, these neural networks, and in general, *neuromorphic computing* are encoded in software and rely on existing hardware and traditional architectures to compute the results. The main operation during the training and the inference of a neural network is the Matrix-Vector Multiplication (MVM) between an input vector and a matrix of synaptic weights. The combination of Multiply - Accumulate operations repeatedly performed with the modern hardware based on von Neumann architecture consumes an incredibly high amount of power and time. By exploiting parallel vector computation typical of modern Graphics Processing Units (GPU) it is possible to drastically increase the power and time efficiency of such operations. Yet, the extensive access to memory containing the synaptic weights suffers from the same von Neumann bottleneck.

Lately, some Application-Specific Integrated Circuits (ASIC) are designed specifically to perform these operations, but since they rely on the traditional CMOS technology they suffer the same problems of miniaturization and power dissipation, limiting factors for high-density devices.

1.5 RESISTIVE RAM FOR NEUROMORPHIC COMPUTING IMPLEMENTATION

1.5.1 *Neural networks*

Neuromorphic computing aims at replicating the operations of a human brain trying to mimic its high energy efficiency and low operating frequency. While a supercomputer can have energy consumption in the order of an entire building, a human brain consumes only a power equivalent to circa $10 - 20W$ and it is capable of performing very complex operations [39]. Neuromorphic computing is performed by neural networks, highly interconnected architectures in which different layers of neurons are connected through synapses [25]. Neural networks can be divided into two main classes: Artificial Neural Network (ANN) and Spiking Neural Network (SNN) [40]. The main difference between the two relies on how the network is trained. An ANN features the presence of several hidden layers, and it is called also Deep Neural Network (DNN). It typically adopts a supervised type of training, such as backpropagation technique, in which the error between the network output layer and the desired output given by a known label is propagated backward by differentiation, updating the synaptic weights to adapt the network response to the input in an iterative fashion [41].

SNN, instead, aims at replicating and implementing a brain-inspired Hebbian-type learning scheme, such as Spiking Time Dependent Plasticity (STDP) [42] or Spiking Rate Dependent Plasticity (SRDP) [43], in which the weight adaptation is governed by biological rules with no external supervision.

Recently, fundamental machine learning tasks such as face recognition, speech recognition, image and pattern classification, have been successfully achieved mainly due to the implementation of ANNs.

A deep neural network can be schematically represented as in figure 1.12, and it is composed bofy several neuron layers. The first one, the input layer, receives the raw signals from the outer world that have to be processed and pass it to the next layers. Then there are one or more hidden layers that compute an intermediate solution, and finally, an output layer that gives the final solution. Each neuron of a layer is connected to all the neurons of the next layer through a synaptic weight, and it is responsible for performing the summation and the integration of the signals. The neuron is also responsible for applying the non-linear function to the weighted sum and for propagating the result to the next synaptic layers.

Synapses, instead, represent the connections between neurons and the weights by which each signal is multiplied. Synapses, just like in a biological brain, are in a much larger number than neurons, thus they must be built using extremely fast and efficient technology, both in terms of occupied size and of consumed power. In the software implementation of the neural networks, the weight values of each

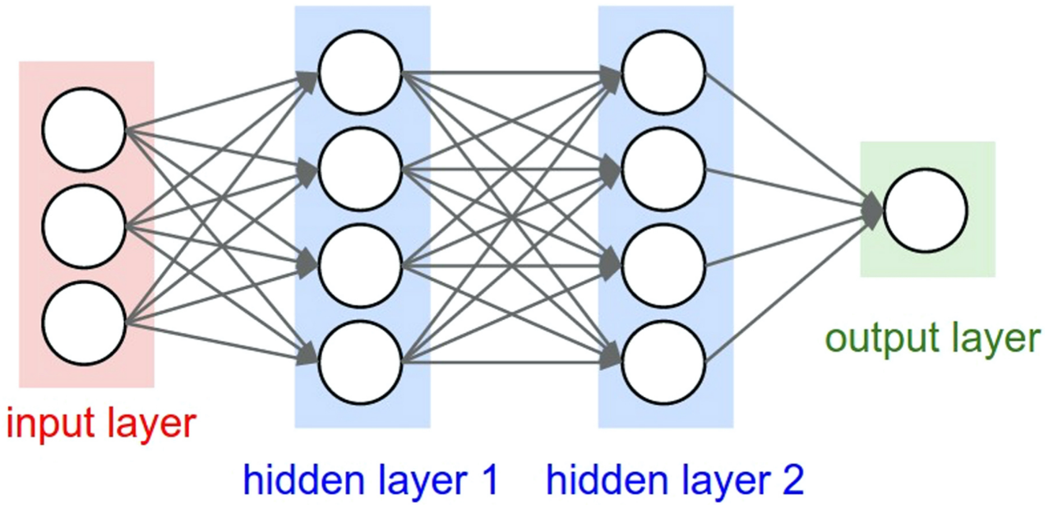

Figure 1.12: Example of a Deep neural network implementation. It is composed of an input layer, several hidden layers, and an output layer. Each of the layers is connected through synaptic connections. Each neuron receives signals from the previous layer, that are weighted by the synapse and summed to the output of the other neurons. Adapted with permission from [25]. Copyright 2019, IOP Publishing Ltd.

synapse are stored in memory, and when the multiplication between the signal and its connected weights occurs, it requires an enormous amount of memory access, wasting CPU cycles and causing severe inefficiencies [44].

A much more elegant solution can be given by implementing such networks in large crossbar arrays. Their natural predisposition to perform algebraic operations like MVM, and their small size and low power consumption make them one of the most promising devices for artificial synapses implementation.

1.5.2 *Backpropagation training method for deep neural networks*

Backpropagation is the main supervised machine learning technique for neural network training [45]. It consists in computing the gradient of the loss function with respect to each synaptic weight, by exploiting the chain rule. The gradient of each layer is computed one at a time and the process is iterated from the last layer backward. The loss function, in this case, is the error function, i. e. the difference

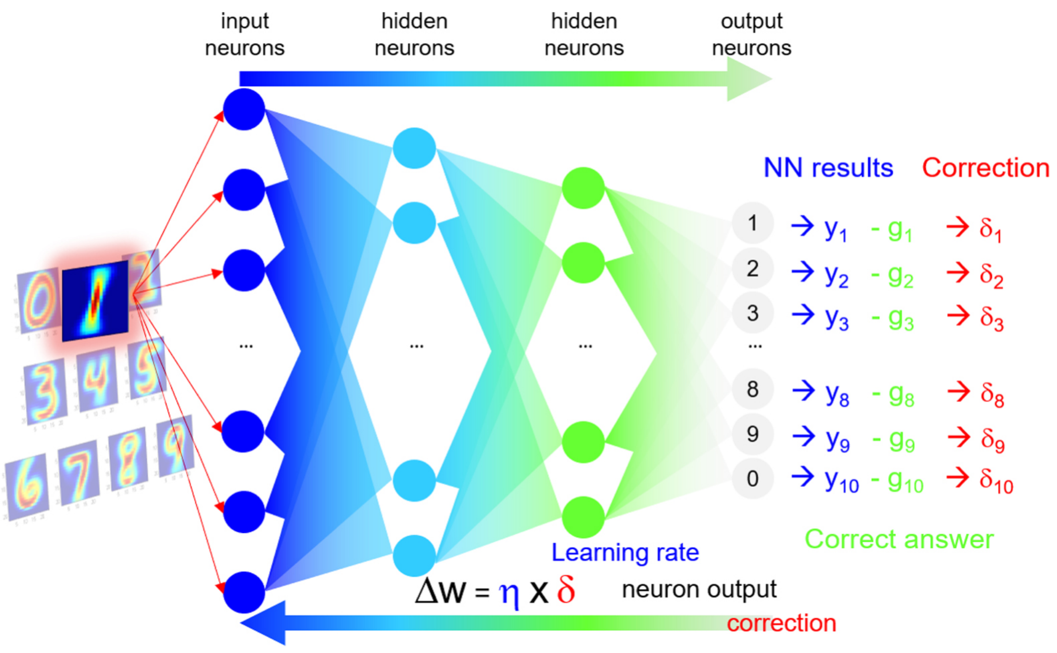

Figure 1.13: Backpropagation algorithm for a multilayer fully-connected neural network. The network is trained with the backpropagation technique to classify handwritten digits from the MNIST dataset. Input signals are forward-propagated, then the y_i output results are compared with the corresponding correct answer g_i, and then the errors $\delta_i = y_i - g_i$ are backward-propagated to previous layers, updating the synaptic weight accordingly to equation (1.1). Reprinted with permission from [25]. Copyright 2019, IOP Publishing Ltd.

between the output of the network and the expected result of a label. Since each neuron's output can be written as a function of its inputs, and each input is an output of the previous neuron's layer, the loss function can be also computed as a function of the outputs of the previous layer, and this procedure can be *backpropagated* all the way to the input layer by the chain rule for derivatives.

Figure 1.13 shows an implementation of a DNN with two hidden layers and the backpropagation algorithm in action to train the network on handwritten images from the MNIST dataset. It is performed first by supplying the training dataset to the input of the network. The signals propagate between each synaptic layer modulated by connection weights. At the output of the last neuron layer, the signals y_i is compared to the label corresponding to the desired output, g_i, and the error is computed $\delta_i = y_i - g_i$. The error function is calculated as $C = \frac{1}{2}\sum_i^N \delta_i^2$ where N is the number of output neurons. δ_i is then backpropagated through all layers of the network, allowing to extract the synaptic weight update for each weight w_{ij}:

$$\Delta w_{ij} = \eta \cdot x_j \cdot \delta_i \tag{1.1}$$

In the formula, η is the learning rate. It defines the speed and the grain of the training. The procedure is repeated several times, adjusting the learning rate and shuffling the input training dataset for the best result, and each training cycle is called an *epoch*. If η is too small, the network needs to perform many training epochs to reach the optimal weight configuration. Instead, if η is too large, the weights update risks to surpass the optimal point and diverge.

After the training is completed, the network is ready to classify images of the same type without any kind of supervision. This phase is called *inference*.

1.5.3 *Neural network implementation with crossbar arrays*

As already mentioned in section 1.5.1, the neurons of a neural network compute the sum of their input signals and pass the result through a non-linear activation function, while the synaptic connections between each neuron modulate each signal accordingly to their training. Since the highly interconnected nature of this type of architecture, the continuous need to access the weight values and then perform the *multiply and accumulate* (MAC) operations, the ideal hardware solution would be to exploit the properties of the crossbar arrays featuring emerging memory devices such as RRAM cells.

For each synaptic connection, it is possible to define a *pre-synaptic* and a *post-synaptic* neuron. The signal reaching the POST neuron y_i will be a function of its PRE neurons x_j and the connecting weights w_{ij} connecting the *j-th* PRE neuron to the *i-th* POST neuron.

$$y_i = \sum_i w_{ij} \cdot x_j \tag{1.2}$$

With the crossbar array architecture discussed earlier, the implementation of the function in equation (1.2) is straightforward. The multiply and accumulate

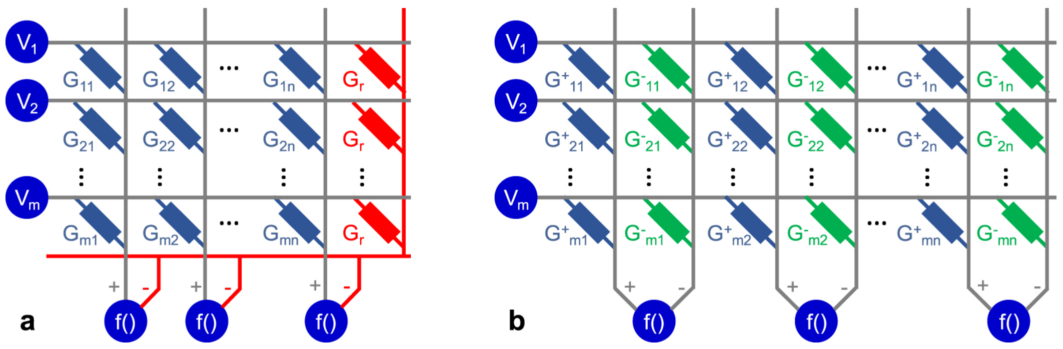

Figure 1.14: Schematic representation of a crossbar array. (a) Synaptic weight is represented as a differential pair of conductances, consisting in a positive G_{ij} variable conductance and a reference G_r with a fixed value; (b) Synaptic weight is represented as a differential pair of two variable conductances: G_{ij}^{+} and G_{ij}^{-}. Reprinted with permission from [25]. Copyright 2019, IOP Publishing Ltd.

operation can be naturally processed by the array by simply exploiting the Ohm's law and Kirchhoff's law of current, i.e. by multiplying each input signal by the corresponding weights and summing all the resulting currents along a row, respectively, as summarized in the following equation:

$$I_i = \sum_i G_{ij} \cdot V_j \tag{1.3}$$

The PRE signals are given in the form of voltages V_j along the columns of the array, and each RRAM cell's conductance G_{ij} at the intersections represents the synaptic weight between the $i-th$ PRE neuron and the $j-th$ POST neuron. Currents I_i are collected along the rows as a result of the MAC operation in equation (1.3).

This implementation is several orders of magnitude faster and more energy-efficient than traditional CMOS-based architectures since it can propagate the result of each synaptic layer in just one clock cycle without the need to access separate memory storage. However, since the weights trained with the backpropagation algorithm can have either positive or negative signs, whereas the conductance value of the RRAM cells can only have a positive value, one of the following strategies should be implemented.

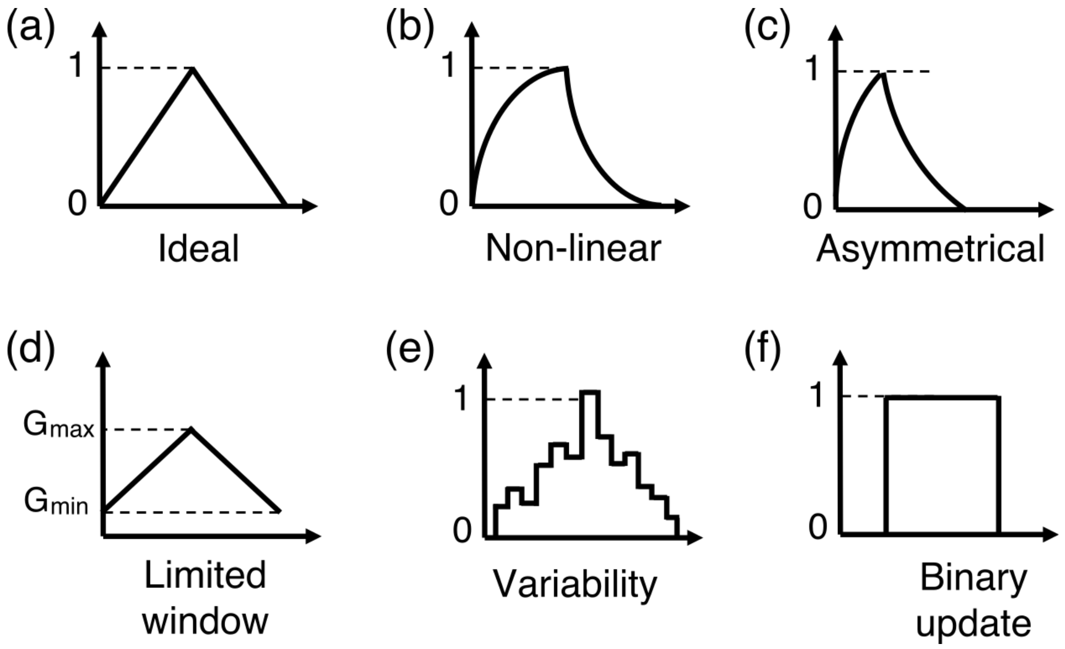

Figure 1.15: Illustration of the weight update characteristic. Representing the conductance G as a function of the number of programming pulses. (a) Ideal linear characteristic; (b) Nonlinear characteristic, with G increases and decreases faster at the beginning of the curve; (c) Asymmetrical weight update with the increase faster than the decrease of G; (d) Limited dynamic range of the weight update; (e) Weight update suffering cycle to cycle variability; (f) binary update, where no intermediate G values are possible. Reprinted with permission from [36], licensed under CC BY 4.0[4]

Several solutions were proposed to solve this problem. As shown in figure 1.14a, one solution is to employ an additional row of conductances with a fixed reference value of G_r, which will be subtracted from the variable and programmable conductance G_{ij}. This is the most compact solution as it only requires a single additional row for each array, however, it reduces the conductive span for each weight, resulting in a decreased number of programmable levels.

On the other hand, by implementing a fully differential pair of programmable conductances, as depicted in figure 1.14b where both G_{ij}^+ and G_{ij}^- can span across the whole conductive range, the number of programmable levels increases but in the trade-off with increased current consumption of the array [25].

To accurately implement the backpropagation training, the linear dependence of the correction value Δw from the product of x and δ must be preserved during the weight update and this requires a highly linear response of the cell's conductance value for a given input voltage. Unfortunately, most of the emerging memory technologies do not exhibit the aforementioned required linearity (Figure 1.15).

The nonlinearity of the weight updates can be solved by performing offline training, by means of a computer simulation exploiting an accurate model representing the RRAM crossbar array with its variabilities and non-idealities. After the offline training, just the final value of synaptic weights has to be programmed to the cells of the array, and the device is then ready for the inference.

However, the nonlinearity of the programming characteristic is only one of the problems affecting the crossbar array implementation of a DNN. In fact, the RRAM devices present a high cycle-to-cycle (C2C) and device-to-device (D2D) variabilities, and some cells in the array happen to be stuck at LRS or HRS levels and not responding to the programming. In some technologies such as RRAM or PCM, the conductance relaxation occurs naturally to the cell after some amount of time and it is accelerated by the temperature increase, and it decreases the fidelity of the network. On top of that, for bigger array sizes and with the scaling of the technology, the parasitic wire resistances along rows and columns cause a non-negligible Ohmic drop, called IR drop [34], reducing the linearity of the MAC operations and affecting the final neural network accuracy.

In conclusion, despite the crossbar array architecture for DNNs shows many great advantages over the traditional computing solutions, its practical implementation suffers from many different non-negligible drawbacks that degrade the overall functioning of the network. Research is exploring methods to mitigate these problems with particular programming algorithms such as program and verify schemes [46] that allow finer control of the programmed conductance enabling for multilevel cell operations [47], or by exploring different architectural choices. The next chapters will focus on these topics.

2

RESISTIVE-SWITCHING RANDOM ACCESS MEMORY

This chapter presents a detailed description of resistive switching random access memory devices. After a brief introduction to the technology, the mechanisms of resistive switching are addressed, both during SET and RESET transitions. In the second part of this chapter, physical mechanisms causing cycle-to-cycle (C2C) and device-to-device (D2D) variabilities are investigated. To conclude the chapter, an overview of the models existing in literature is briefly presented.

2.1 INTRODUCTION

RRAM devices are among the most promising candidates for next-generation memory. They feature exceptionally low power operations and allow for high-speed switching between states. Along with that, they present a very high cycling endurance, a very important feature for non-volatile memory applications. In addition to this, their low fabrication cost and compatibility with the BEOL process at relatively low temperatures enables highly scalable high-density chips compatible with CMOS devices and 3D stacking integration [15], optimal for the realization of crosspoint arrays. RRAM devices present also some drawbacks and limitations, the most important being the programming variability and state fluctuation due to noise, and Conductive Filament (CF) relaxation. This affects the cell reliability and puts some limitations on the multi-level operation of such devices.

This work focuses on different program/verify schemes that try to reduce the effects of variability, allowing this technology to be used for tasks such as hardware implementation of neural networks.

2.2 RRAM SWITCHING MECHANISM

In literature, switching mechanisms have been attributed to the filamentary modification of conduction properties [15]. An analytical model for RRAM describes switching as a voltage-controlled change of the CF size.

Initially, an RRAM cell must be subjected to the operation of electroforming, where the conductive filament is formed by dielectric breakdown. The forming process is usually performed by applying a high positive voltage across the device and results in the oxide layer reduction and creation of oxygen vacancies. This process must be limited by a compliance current, otherwise, the dielectric layer of the device may suffer a hard breakdown, becoming unusable.

After forming, the conductance of the device is higher because now there is a CF connecting the TE and the BE through the oxide layer. Now, by applying proper voltages across the cell, the CF will break by the effect of Joule's heating resulting in a decrease of the conductance between the two electrodes and pushing the cell into HRS. This conductance will be a little bit higher than the initial state before forming, because the CF is not completely dissolved, rather only disconnected via a relatively small depletion gap.

Starting from HRS, now, it is possible to perform multiple transitions between HRS and LRS by alternating set and reset operations. In RRAM technology, differently from what happens in modern memory technologies, such as flash, the state changes by physically altering the structure of the device rather than just modifying the charge stored in the floating gate. For this reason, this kind of memory is strongly affected by switching variability, and every time the filament is created and dissolved, the final value of conductance will be slightly different.

Two main methods of resistive switching exist, and they differ by the polarity of the set and reset operations. If both set and reset processes take place under positive voltage, the effect is called unipolar switching, while if the polarity of the set is the opposite of the polarity of the reset, the process is called bipolar switching [15].

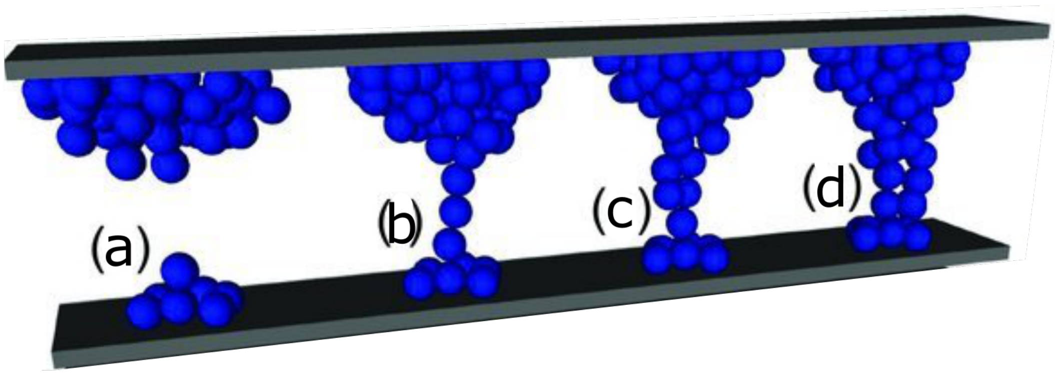

Figure 2.1: Starting from the reset state (a), first the nucleation process takes place (b) and then the filament grows (c) until the process is limited by the compliance current (d). Reprinted with permission from [48]. Copyright 2013, John Wiley & Sons, Inc. License number 5141851303192.

Unipolar switching has the advantage of being easier to implement from an architectural standpoint since only positive voltages and unipolar diodes for device selection are required for all the applications. On the other hand, unipolar switching typically shows larger switching variability and lower cycling endurance compared to bipolar switching devices. For this reason, most research and also this work focuses on the bipolar switching RRAM.

Another distinction can be made among bipolar RRAM devices, depending on the employed materials in the MIM structure and thus, on the type of migrating defects: oxygen vacancies for OxRAM and metallic cations for CBRAM. The former devices are made by employing a transition metal oxide such as HfO_2, with a metallic cap, that is made also with a transition metal. The metallic cap is a buffer layer that acts as an oxygen getter [14], lowering the voltage barrier for the forming and set transitions and also increasing the ratio between HRS and LRS. It also dictates the polarity of the bipolar switching, with the set transitions usually taking place when a positive voltage is applied to the electrode at the cap side.

CBRAM instead utilizes an electrolyte dielectric layer for cations supplied by the cap, consisting of a chalcogenide material. Usually, CBRAM devices show a larger resistance window than OxRAM [15].

This work from now on focuses on OxRAM devices.

The set mechanism consists of two separate parts: nucleation and growth of the CF. After a reset operation, the filament is not present (Figure 2.1a). By applying a voltage across the device the filament nucleation takes place (Figure 2.1b), i. e. some positively charged ions migrate from the ion's reservoir through the oxide creating a thin conductive path connecting the TE to the BE. This migration is possible by the decrease of the energy barrier described by equation (2.2) and it is assisted by the temperature increase due to the Joule's heating effect. By further increasing the voltage (Figure 2.1c), the CF grows by ion migration and deposition, enlarging its diameter ϕ and thus decreasing the resistance between the two electrodes until the desired level is reached (Figure 2.1d). With every increase in the filament's diameter ϕ also the conductance increases. It is a positive feedback for $V > V_{SET}$ that is limited by the compliance current I_C.

This growth dynamic can be modeled by a rate equation following the Arrhenius expression [49]:

$$\frac{d\phi}{dt} = Ae^{\frac{-E_A}{kT}} \tag{2.1}$$

where A is a preexponential constant factor, E_A is the energy barrier for ion migration, k is Boltzman's constant, and T is the local temperature along the CF. Equation (2.1) states that the growth rate of the CF is controlled by ion migration. This equation is an Arrhenius function of temperature, and it describes the probability of ion hopping by overcoming the energy barrier E_A given by:

$$E_A = E_{A0} - \alpha qV \tag{2.2}$$

where E_{A0} is the energy barrier for null voltage, V is the voltage across the cell, α is the barrier lowering coefficient (Figure 2.2). This enhances the ion migration rate in presence of an applied electric field, easing the growth of the filament.

The temperature T in equation (2.1) takes into account the Joule heating effect, which is non-negligible due to the high current density and the electric field across the filament. The temperature can be found by solving the following 1D steady-state Fourier equation [50]:

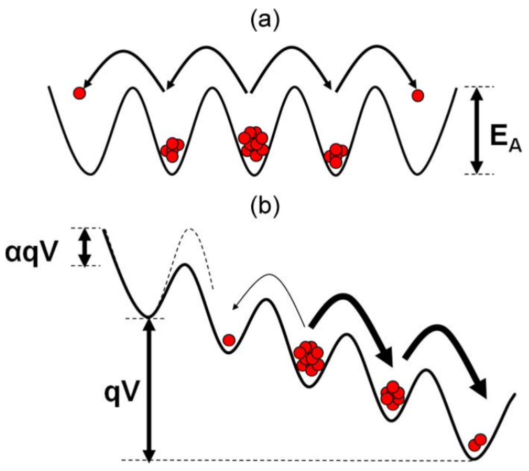

Figure 2.2: Schematic for potential energy ion hopping at (a) zero potential or (b) with a positive voltage applied. Reprinted with permission from [18]. Copyright 2012, IEEE.

$$\kappa_{th}\frac{d^2T}{dz^2} + J^2\rho = 0 \tag{2.3}$$

with κ_{th} being the thermal conductivity, z the direction along the CF, J the current density across the device, and ρ the electric resistivity of the medium.

Solving equation (2.3) with boundary conditions of $T = T_0$ at the two metal contacts considered as ideal heat sinks, i. e. at $z = 0$ and $z = L$ with L being the total length of the oxide layer, leads to a parabolic solution with the maximum temperature T_{max} found in $z = L/2$ with the value of:

$$T_{max} = T_0 + \frac{J^2\rho}{8\kappa_{th}}L^2 \tag{2.4}$$

that can be rewritten as:

$$T_{max} = T_0 + R_{th}P \tag{2.5}$$

by using the following substitutions:

$$J = \tfrac{V}{R}\tfrac{1}{A_{CF}}$$
$$P = \tfrac{V^2}{R}$$
$$R_{th} = \tfrac{L}{A_{CF}\kappa_{th}}$$
$$R = \tfrac{\rho L}{A_{CF}}$$

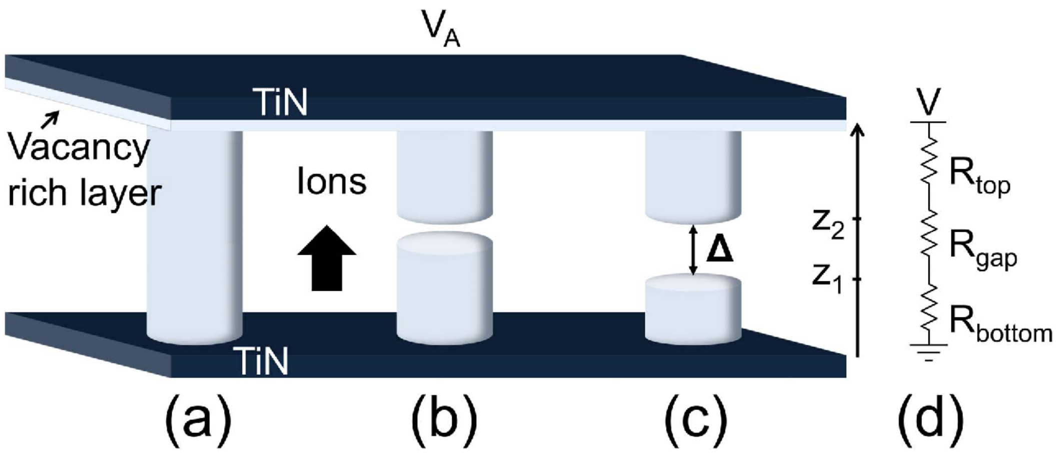

Figure 2.3: Starting from the set state (a), by applying the voltage across the cell the filament breaks (b), and then the gap is created (c). Reprinted with permission from [50]. Copyright 2014, IEEE.

T_0 is the room temperature, R_{th} is the thermal resistance of the CF and P is the power flowing through the device $P = VI$.

The reset mechanism, instead, consists in creating and enlarging a gap in the just-formed CF and it is controlled by applying an opposite polarity voltage to the electrodes. Starting from the LRS condition featuring a continuous filament as shown in figure 2.3, as the voltage across the device increases, positively charged oxygen vacancies start to migrate towards the TE driven by the electric field and helped by the temperature increase, thus creating a gap in the CF. The gap growth is controlled by a similar Arrhenius equation already seen for the set transition.

$$\frac{d\Delta}{dt} = Ae^{\frac{-E_A}{kT}} \tag{2.6}$$

Differently from the set process that shows a quadratic relation between the rate equation of the diameter ϕ and the conductance of the cell, the reset transition has a linear relation between the resistance and the gap size Δ.

2.3 VARIABILITY OF RRAM PROGRAMMING

The Resistive Random-Access Memory (RRAM) devices rely on the physical voltage-driven formation/disruption of a Conductive Filament (CF) across a thin insulating

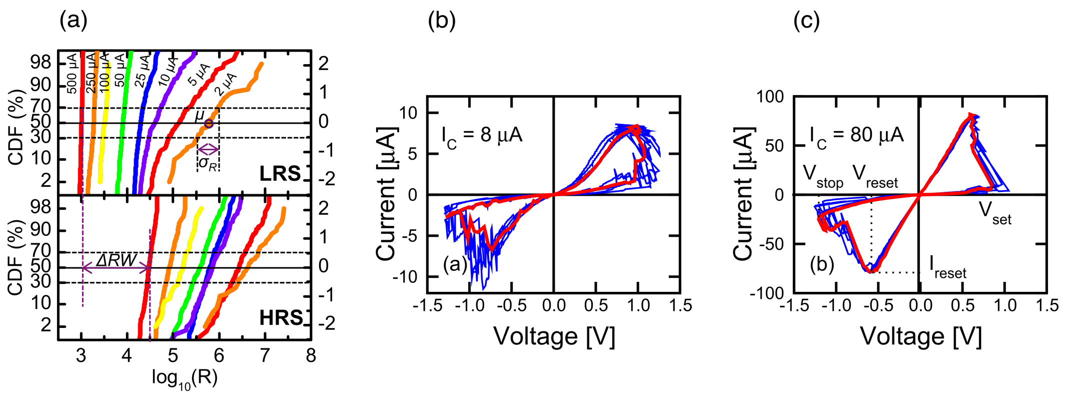

Figure 2.4: (a) C2C cumulative distribution of LRS (top) and HRS (bottom) for different compliance current I_C, from $2\mu A$ to $500\mu A$. As I_C decreases, the conductance follows, and the distributions bend. Distributions are circa lognormal.; Adapted with permission from [51]. Copyright 2013, IEEE. (b) Measured I-V characteristics for $I_C = 8\mu A$ and $I_C = 80\mu A$ (c) blue lines, while the median behavior is highlighted with the red curve. Adapted with permission from [52]. Copyright 2014, IEEE.

layer. With the scaling process of the devices and the oxide layer approaching few-atom size, RRAM becomes vulnerable to variability, statistic fluctuation effects, and noise, affecting switching precision.

The set and reset processes are performed by moving ions and defects back and forth through the insulator, and since the defects are discrete, they tend to assume different geometrical conformations in a stochastic way from cycle to cycle and from device to device, thus influencing the final conductance value of the CF. As shown in figure 2.4a, when the compliance current I_C increases, the cumulative distribution of the resistance value becomes steeper, meaning that the programming variability decreases [51, 52]. This figure highlights also a direct proportionality between the median value of the resistance μ_R and its standard deviation σ_R. From a practical standpoint, to obtain a cleaner programmed conductance, one should aim at higher LRS values, thus limiting the usage in very large arrays due to increased parasitic effects such as the ohmic IR drop. Moreover, a high variability and the relatively small resistance window affect also the Multi-Level Cell (MLC) operation limiting the programming accuracy needed for sensible applications such as synaptic weights.

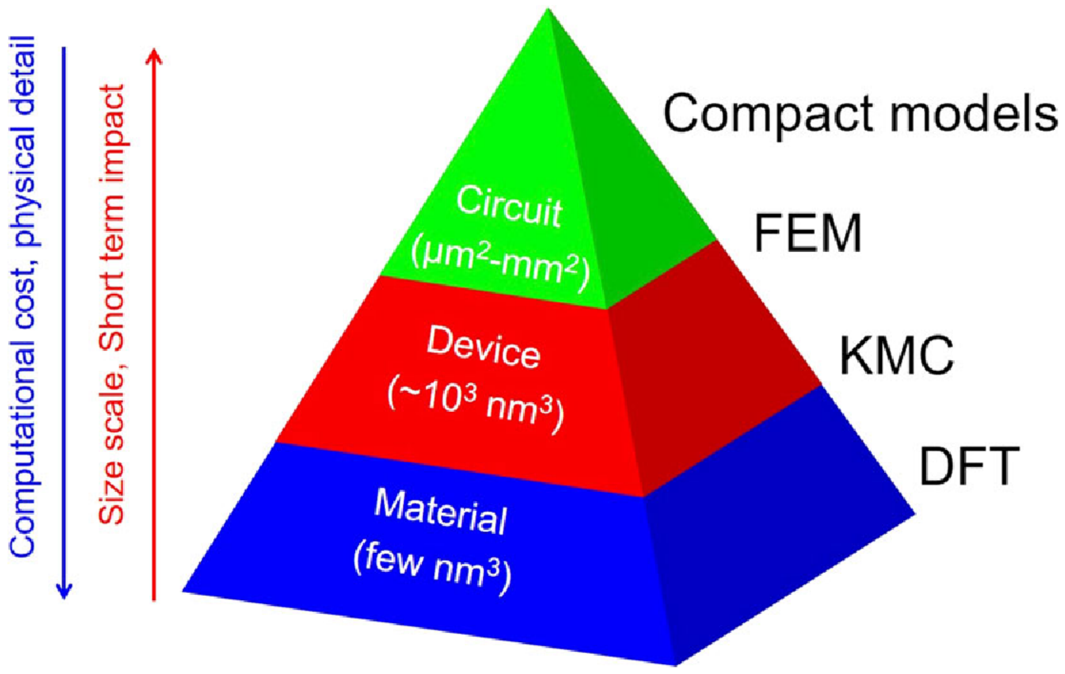

Figure 2.5: Physical models hierarchy. From bottom to top: material atomistic modeling relying on density functional theory (DFT); statistical modeling - Kinetic Monte Carlo (KMC) and finite element method (FEM) to describe the device-level behavior; Compact modeling describing the global characteristics of the device using a simple analytical formula. Reprinted with permission from [53], licensed under CC BY 4.0[1]

2.4 RRAM MODELING

As with every emerging technology, Resistive RAM needs the availability of accurate models that can to predict the devices' operation, the variability, and how they behave with the scaling of technology. Several models have been developed ranging from materials-level atomistic simulations to device simulations, and to compact models able to describe the behavior of the RRAM devices at a system level [53].

As summarized in figure 2.5, different categories of physical-based models exist. Focusing on RRAM, the main interest resides in understanding how the device behaves under proper voltage-current conditions, and how the process of resistive switching happens. Starting from the most precise modeling techniques, such as ones relying on density functional theory (DFT), the device at the atomistic scale can be described, yielding to the understanding of the physical structure of the material and ion migration mechanisms. While providing the most accurate physical description, this kind of model has a very high computational cost.

Some less heavy models exist, computationally speaking, such as kinetic Monte Carlo (KMC) or finite element method (FEM). FEM models solve transport equations in 2D or 3D geometries, discretizing the volume with finite elements. The aim is to describe the thermal and ionic migration effects in a continuous domain. On the other hand, KMC models deal with discrete quantities and consider the generation-recombination of oxygen vacancies as the main factor contributing to the forming of the conductive filament [54]. By running several simulations and exploiting the stochasticity of the position of the defects, a Monte Carlo method can extract an average switching characteristic.

The just-introduced models offer an accurate and detailed physical description of the device, but are not suitable for a large-scale simulation, for example of a crossbar array or a memory device, employing both RRAM devices and CMOS circuitry. A compact model, instead, offers a computationally light description of the switching phenomena, for example, the conductive filament geometrical growth/rupture or directly the conductance value, by employing simplified analytical formulae [50] [49].

3 EXPERIMENTAL DATA ANALYSIS

This chapter focuses on the analysis of cycling endurance experimental data, measured on a $4 - kbit$ crossbar array of RRAM devices in a 1T1R configuration manufactured by IHP Microelectronics. The measurements focus on highlighting statistical C2C and D2D variability of the resistive switching mechanisms, by employing different program and verify schemes, namely Incremental Step pulse Program and Verify Algorithm (ISPVA) and Incremental Gate Voltage and Verify Algorithm (IGVVA) with different voltage steps. Particular emphasis is put on the algorithms and methods used during this works for extracting information from the devices.

3.1 CELL AND ARRAY STRUCTURE

All of the measurements discussed and analyzed in this chapter were performed and provided by IHP Microelectronics. The chip sample used in the present study (Figure 3.2) is a $4 - kbit$ crossbar array combined in 64 rows and 64 columns. The chip contains all the necessary architectural blocks needed to drive the array (Figure 3.2c). Each cell in the memory array is selected with two address decoders. The *row decoder* (XDC MUX) selects a single word line (WL), i. e. one of the 2^6 gates of the access transistors. The *column decoder* (YDC MUX) selects a single bit line or source line (BL, SR) of the array. An operation control circuitry (Mode) is also present to control the overall circuitry and select the operation modes. The device under test is an RRAM element arranged in one Transistor - one Resistor (1T1R) configuration. Each 1T1R cell is constituted by an N-channel Metal Oxide Silicon Transistor (NMOS) transistor manufactured in IHP's $0.25\mu m$ CMOS technology, whose drain is connected in series to the RRAM. This resistor is a MIM device

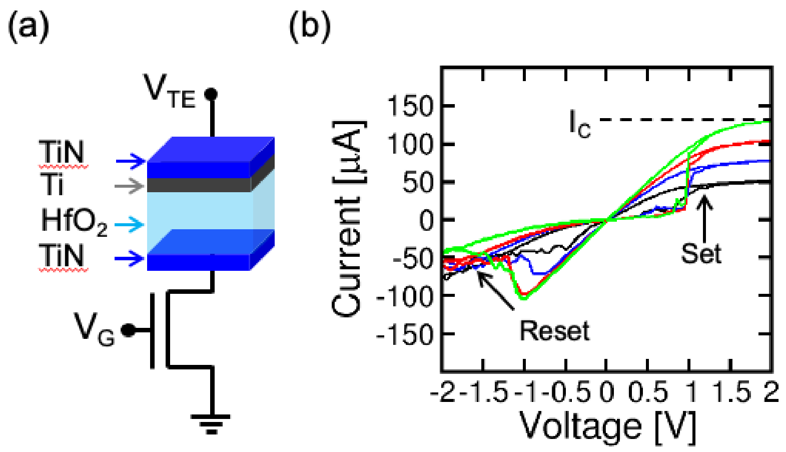

Figure 3.1: (a) Structure of the 1T1R cell and the MIM stack description. (b) bipolar I-V characteristic for set and reset transition. Reprinted with permission from [55]. Copyright 2021, IEEE.

located on metal line 2 of the CMOS process. The MIM device consists of a stack of $TiN/Ti/Hf_{1-x}Al_xO_y/TiN$ (Figure 3.1a).

The $Hf_{1-x}Al_xO_y$ dielectric layer is grown by Atomic Layer Deposition (ALD) and has a thickness of $6nm$, with an Al concentration of about 10%. Metal layers were deposited by magnetron sputtering with a thickness of $150nm$ for the top and bottom TiN layer, and $7nm$ for the Ti layer. The final area of the device is about $0.4\mu m^2$, and it is protected by an additional thin Si_3N_4 layer.

3.2 PROGRAM AND VERIFY ALGORITHMS

This work focuses on studying and analyzing the impact of statistical variability of RRAM cells in multilevel operations. Since the resistive switching mechanism is the result of ion migration across the device under a certain applied voltage, the final result can change considerably even between two consecutive programming cycles, within the same device. This effect can be mitigated by performing a more controlled transition from one resistive state to another. For this reason, several different programming algorithms were developed allowing for finer control of the reached conductive value.

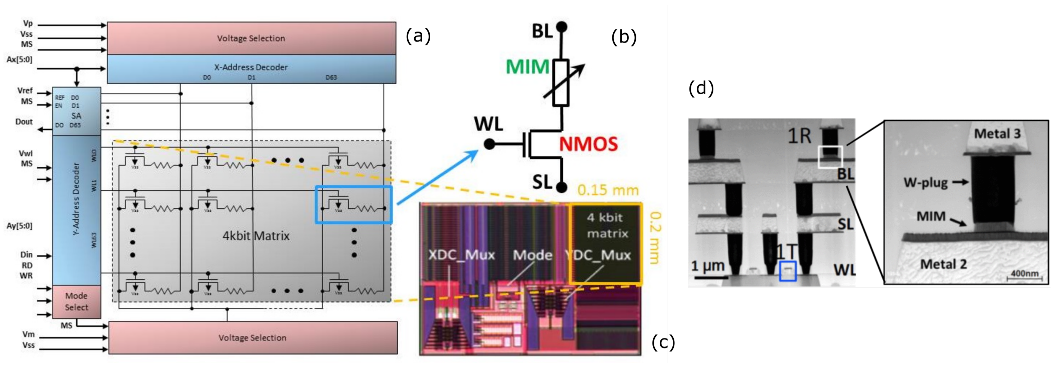

Figure 3.2: (a) Block diagram and (c) micrograph of the 4 kbit memory array with the surrounding circuitry. (b) Schematic of the 1T1R cells integrated into the array. Reprinted with permission from [56]. Copyright 2019, IEEE. (d) TEM cross-sectional image of 1T1R architecture. Reprinted with permission from [57]. Copyright 2017, IEEE.

Instead of directly imposing a high voltage and letting a high amount of current to flow across the device, the MOSFET device in the 1T1R structure is exploited in order to limit the compliance current during the set and reset operations, and to control precisely the formation and the rupture of the CF by applying finely stepped pulsed voltages to the three terminals of the device (TE, BE, Gate).

Moreover, to increase even more the accuracy and to partially reduce the variability, between each pulsed programming step P a verify step V is performed, by applying a small fixed voltage across the 1T1R device to read the resulting current. The voltage of the Verify phase is chosen to be small enough not to perturb the RRAM state.

These kinds of programming schemes are called *Program and Verify algorithms*.

In literature and practice, several different program and verify algorithms were proposed, for both set and reset transitions. In this work, two different algorithms are discussed.

Incremental Step pulse Program and Verify Algorithm (ISPVA) (Figure 3.3b) consists in defining a compliance current for a certain level, and gradually increasing the voltage across the device until the correct programming value is reached;

Incremental Gate Voltage and Verify Algorithm (IGVVA) (Figure 3.3a), instead, imposes a voltage across the device that is surely high enough for the transition to

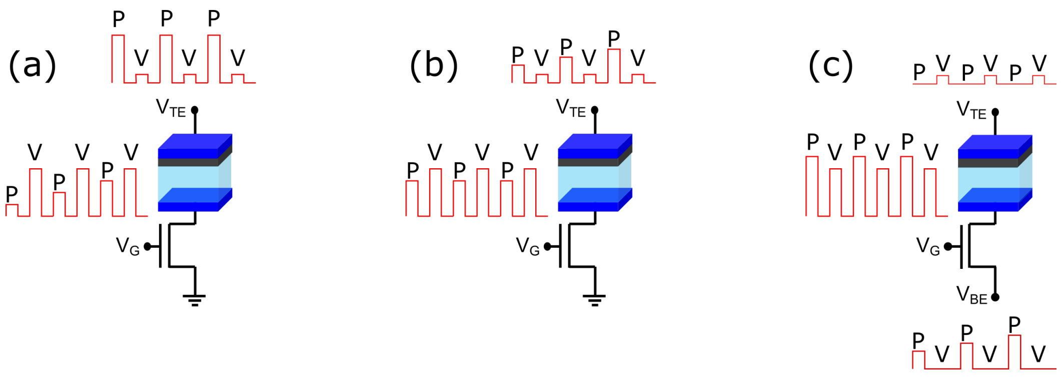

Figure 3.3: Examples of the pulsed program and verify algorithms in which are alternated program pulses (P) and verify pulses (V) at the different nodes: (a) IGVVA, (b) ISPVA, (c) reset.

occur, and then reaches the desired programming value by gradually increasing the compliance current, by modulating the gate terminal of the NMOS transistor.

The reset algorithm (Figure 3.3c) used in this work is similar to the ISPVA programming scheme, but the voltage applied across the device has the opposite sign.

In the following sections, these algorithms will be analyzed in detail.

3.2.1 *ISPVA SET Algorithm*

The first program and verify algorithm present in literature is the Incremental Step pulse Program and Verify Algorithm (ISPVA) [57] (Figure 3.3b), and it is performed by applying pulsed voltages to both the Top Electrode (TE) and the gate of the NMOS, keeping BE at ground. The V_{TE} is incremented from $0.5V$ to $2.0V$ with steps of $100mV$, while the gate voltage depends on the desired compliance current. In this work, four different target levels of LRS are studied in-depth, corresponding to the following four touples of $< I_{target}, V_G >$, in particular: $< 10\mu A, 1.0V >$, $< 20\mu A, 1.2V >$, $< 30\mu A, 1.4V >$ and $< 40\mu A, 1.6V >$ (Figure 3.4).

After every programming pulse, a verify pulse is applied to the device, with $V_{TE} = 0.2V$ and $V_G = 1.7V$. The verification process consists in reading the current during each read state, and if it exceeds the target current chosen for the level,

(a)

	I_{target}	G_{target}	V_G
HRS	5μA	25μS	2.7V
LRS1	10μA	50μS	1.0V
LRS2	20μA	100μS	1.2V
LRS3	30μA	150μS	1.4V
LRS4	40μA	200μS	1.6V

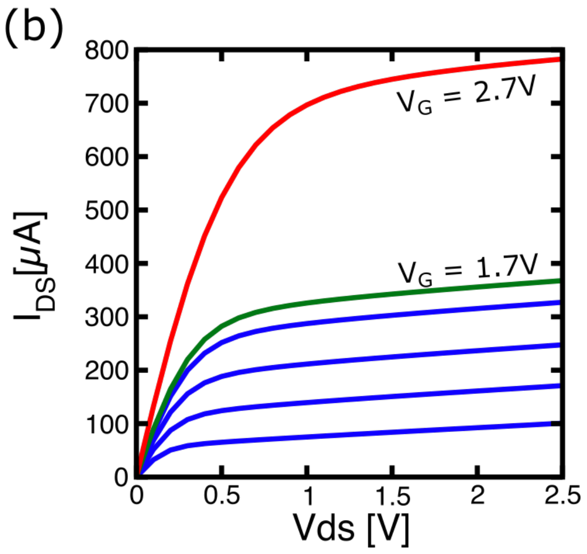

Figure 3.4: (a) Recap of different levels in ISPVA that are programmed by controlling the compliance current through the V_G voltage of the NMOS. (b) Some of the I_{DS} - V_{DS} characteristics of the NMOS in series to the RRAM cell. ISPVA LES levels (in blue), reset (in red), and read gate voltage (in green).

the algorithm stops, otherwise, the algorithm continues with applying the next program pulses.

Taking as an example a single device cycled 1000 times by alternating set and reset programming, for the highest level LRS4 corresponding to $V_G = 1.6V$ and $I_{target} = 40\mu A$ is shown in figure 3.5a. Each line in the figure corresponds to the current read during a single cycle after each set pulse. As one can see, from cycle to cycle the device presents a relatively high variability. This can be explained by the fact that each time a Conductive Filament (CF) is formed during the set and disrupted during the reset, the ions in the CF migrate and arrange in different positions.

One particularity of the ISPVA transition is that the CF growth is triggered only after a certain V_{TE} is applied across the device, and this voltage, namely V_{SET}, varies from device to device and from cycle to cycle. In figure 3.5b are highlighted certain cycles that have an early transition, for lower V_{SET} in red, and higher V_{SET} in blue. It can be noticed from the figure that when the device switches early, its conductive filament grows in a more controlled way and follows an upper bound given by the compliance current of the NMOS, that because of a

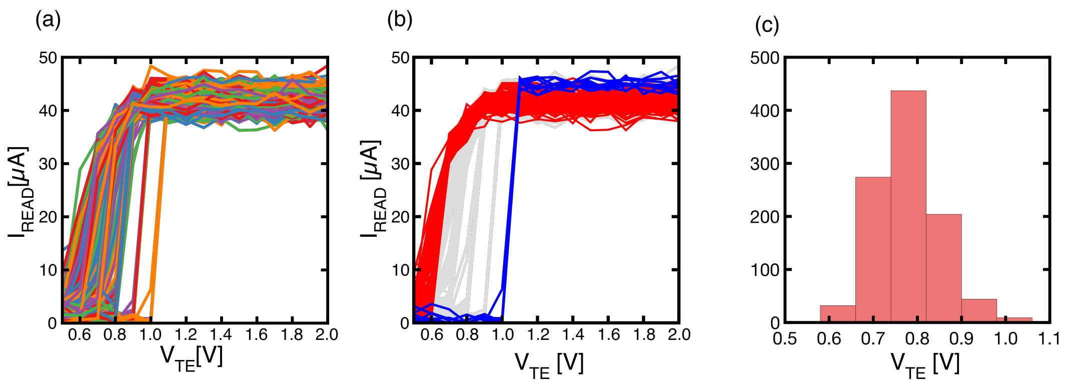

Figure 3.5: (a) Extracted I_{read} during set transitions of LRS4 during ISPVA100 programming ($I_{target} = 40\mu A$). Device #229, 1000 cycles; (b) Highlight of (read) early transitions and (blue) late transitions; (c) Histogram of the voltage V_{TE} distribution for which the cell switches from HRS to LRS.

low V_{TE} that consequently limits the V_{DS} falling across the transistor. On the other hand, when the device switches at a higher V_{TE}, this polarization limit does not influence anymore the growth of the CF, with the result of a much steeper increase in conductance. In general, the distribution of V_{SET} assumes an almost Gaussian distribution, as shown for the device under test in figure 3.5c.

3.2.2 *IGVVA SET algorithm*

Incremental Gate Voltage and Verify Algorithm (IGVVA) (Figure 3.3a) [58], instead, is performed by keeping the TE at a constant fixed voltage of $V_{TE} = 1.2V$, while increasing V_G from $0.5V$ to $1.7V$.

In particular, in this work two different IGVVA variants were analyzed, differing in the voltage steps made by the gate, $\Delta V_G = 100mV$ for IGVVA100 and $\Delta V_G = 10mV$ for IGVVA10. Both the algorithms have a pulse duration of $dt = 1\mu s$, and after each programming pulse a readout pulse of the same duration $dt = 1\mu s$ is applied ($V_{TE} = 0.2V$) and $V_G = 1.7V$).

This kind of algorithm allows for finer control of the final conductive state because at every next step the current is limited by the NMOS polarization instead of being limited only by the voltage. Differently from ISPVA, programming the cell with IGVVA results in a more gradual and controlled conductive increase. With

ISPVA, after reaching a threshold value of $V_{TE} = V_{SET}$ the cell's conductance grows abruptly until the compliance level is reached (Figure 3.5a), when programming with IGVVA the compliance current increases at every step and since the voltage at the top electrode is chosen to be higher than V_{SET}, the conductance of the RRAM grows just until the compliance of the step is reached. In the following step, the compliance increases letting the cell increase its conductance another bit.

Another visible difference between the two approaches is the point at which the conductance begins to grow. In ISPVA, this was directly connected to the V_{TE} overpassing the V_{SET} value, and it suffers from high variability, both cycle to cycle and device to device. In IGVVA, on the other hand, since the $V_{TE} > V_{SET}$ condition is always reached, this variability is considerably lower.

By reducing the voltage steps from $\Delta V_G = 100mV$ to $\Delta V_G = 10mV$, despite the increase in time needed to reach the desired state, the accuracy and the control of the programmed conductance value increase significantly.

The advantages over ISPVA are evident. First of all, by having smaller conductive increase steps it is possible to control more finely the value of the conductance and to interrupt the algorithm when a defined target value of current is read during the readout phase. Another benefit of using IGVVA over ISPVA is that the algorithm is completely identical for every level and does not differ as it happens in ISPVA, by changing the V_G accordingly.

3.2.3 *RESET algorithm*

As per the set transitions, also the reset transition is performed with a program and verify algorithm. The reset algorithm is similar to the ISPVA, but this time the stair voltage ramp has to be applied to the Bottom Electrode (BE), keeping the TE at ground, in fact, reversing the voltage direction (Figure 3.3c). V_{BE} is incremented from $0.5V$ to $2.0V$ with steps of $100mV$, and $V_G = 2.7V$. After every reset pulse, there is a verify pulse equal to the two set algorithms ($V_{TE} = 0.2V$, $V_{BE} = 0V$, and $V_G = 1.7V$). The target current that should be reached during the readout phase

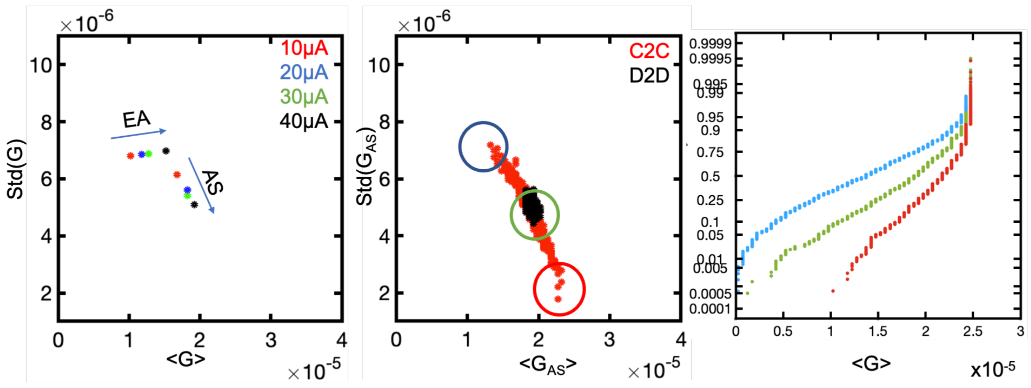

Figure 3.6: (a) Extracted reset After switching and End of algorithm values for four different levels; (b) C2C and D2D distributions of reset after switching values; (c) distribution of reset values for three devices. The red curve represents the cleanest device, the blue curve the dirtiest, while the green curve refers to the median device with the most common behavior.

of the algorithm is $I_{reset} = 5\mu A$, corresponding to a conductance level of $25\mu S$, and it is not dependent on the previously programmed LRS.

Analyzing the HRS value that is reached after a RESET, shows how it is partly dependent on the previous programmed LRS value (Figure 3.6a). This can be explained considering how the processes of growing and the breaking of the CF are performed. One can suppose that after a forming or a set process with a specific compliance current, the Conductive Filament (CF) has approximately a cylindrical form with a diameter ϕ_1. If then a reset transition occurs, it will create a gap in the CF with the height Δ_1 to reach the target read current of the reset transition ($5\mu A$ in our study). If instead, the CF was formed with a different compliance current, the filament will have a diameter ϕ_2 and with the following reset transition the gap will have a height of Δ_2. It is experimentally validated that this geometric difference brings to a slightly different After Switching (AS) reset value, as if the reset of a thinner CF (i. e. lower LRS) has a steeper decrease. This emphasizes how the RRAM cell has a sort of memory of its previous conductive state.

Analyzing the variability distribution of the After Switching (AS) value, it can be noted that some devices present a more thin distribution over the endurance cycles while some others have a large span of values (Figure 3.6b,c).

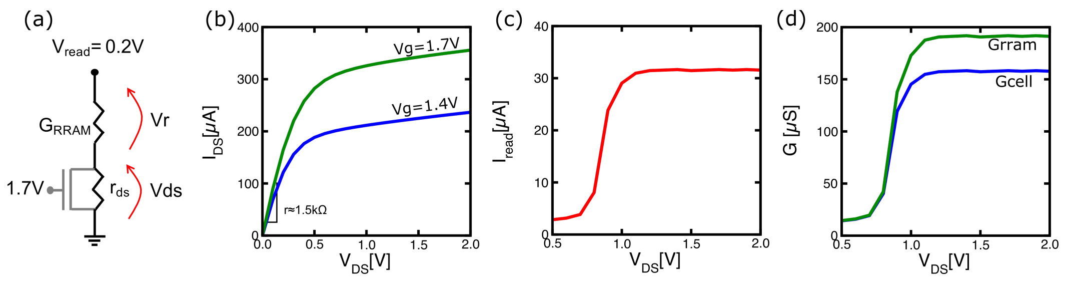

Figure 3.7: (a) representation of the 1T1R series during the verify (V) phase. The transistor is in the Ohmic regime and has a non-negligible resistance value. (b) I-V characteristics during the program (P) phase ($V_G = 1.4V$) of the $LRS3$ level and verify (V) phase ($V_G = 1.7V$). (c) $LRS3$ read current during the ISPVA. (d) comparison between the conductance value of the 1T1R cell and of just the G_{RRAM} value.

3.3 EXPERIMENTAL DATA

The goal of this work is to analyze and study the different program and verify algorithms that are introduced in the previous sections.

The measurements were performed on the 1T1R chip introduced in section 3.1, and they consist of the read current I_{read} that is sampled during the *verify* phase of the algorithms (Figure 3.7c).

Along with the read current for the different algorithms and the different levels, also the exact I-V characteristic for the NMOS transistor is available.

The actual compliance current is given during the programming (P) pulses and is much higher than the corresponding I_{target} of the defined levels (Figure 3.4b) that is extracted during the readout. Taking as an example ISPVA100 - LRS3, between each programming pulse (P) and each verify pulse (V) the polarization of the NMOS changes (Figure 3.7b), and a maximum current value of around $150\mu A$ is reached right after the conductance growth. Instead, during the next verify pulse, the maximum current that is reached is the desired target value of $I_{target} = 30\mu A$ (Figure 3.7c). In fact, by applying only $0.2V$ at the top electrode and $1.7V$ to the gate, the NMOS is in the Ohmic region and acts as a resistor of about $1.5k\Omega$. The $G_{CELL} = I_{read}/V_{read}$ that is considered in the algorithm is a series of the $r_{ds,on}$ of the NMOS and the resistance $R_{RRAM} = 1/G_{RRAM}$ (Figure 3.7a). By intersecting the

NMOS characteristic with the verify polarization, it is possible to extract the actual value of G_{RRAM} (Figure 3.7d) that allows the cell to reach the desired current.

3.3.1 *Datasets*

The available experimentally measured data represent the current that is read after each programming (set/reset) step. As stated in section 3.2, the readout of the cell is performed by applying a Top Electrode voltage $V_{TE} = 0.2V$ and a gate voltage of $V_G = 1.7V$. This polarizes the NMOS to have low *on-channel resistance* (about $r_{ds,on} = 1.5k\Omega$) and allows to read the current of the cell without the risk of perturbing its state.

Four main *conductance levels* were aimed during this study, corresponding to four different read currents used as targets during the verify phase of the algorithms. These levels, LRS1 to LRS4, correspond to read currents of $10\mu A$, $20\mu A$, $30\mu A$, and $40\mu A$, respectively.

These four levels were programmed onto a single 64x64 crossbar array, virtually separating it into four 16x64 different regions. Each region of 1024 devices is programmed with only one LRS level, meaning that all of the operations, from electroforming to set-reset cycling, are performed with the same compliance.

Each 1T1R device was first subjected to electroforming by using an ISPVA algorithm with TE voltage amplitude sweep from $V_{TE} = 2.0V$ to $V_{TE} = 5.0V$ and steps of $0.01V$.

Incremental Gate Voltage and Verify Algorithm (IGVVA) is performed by applying a fixed voltage $V_{TE} = 1.2V$ to the Top Electrode and sweeping the gate voltage from $0.5V$ to $1.7V$ with steps of $10mV$ for IGVVA10 (Figure 3.8a) and steps of $100mV$ for IGVVA100. The Top electrode voltage is chosen to be higher than $Vset$, while the gate terminal controls the compliance current of the device by modifying the polarization of the NMOS.

Incremental Step pulse Program and Verify Algorithm (ISPVA) instead is executed by applying a fixed gate voltage that limits the NMOS polarization, thus the compliance current for the desired level, respectively ($V_G = 1.0V$, $V_G = 1.2V$,

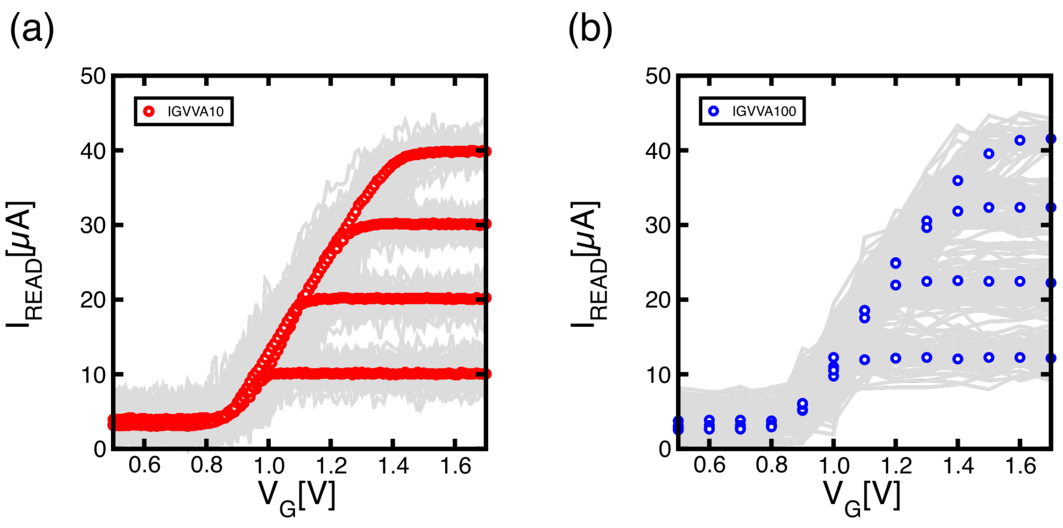

Figure 3.8: Four LRS levels measured adopting the IGVVA programming scheme. Each measurement is affected by noise and variability (in grey) and median behavior is extracted (colored). Two different implementations of IGVVA were analyzed: (a) IGVVA10 with gate voltage steps of $10mV$ and (b) IGVVA100 with gate voltage steps of $100mV$.

$V_G = 1.4V$, and $V_G = 1.6V$ for $I_C = 10\mu A$, $I_C = 20\mu A$, $I_C = 30\mu A$, and $I_C = 40\mu A$ respectively). The Top electrode sweeps with steps of $100mV$ from $0.5V$ to $2.0V$.

The RESET algorithm does not depend on which SET algorithm was previously used nor on the programmed conductive level. It is similar to ISPVA, in fact, the gate terminal is kept at a fixed voltage of $V_G = 2.7V$, while the voltage across the device is set by applying a ramp on Bottom Electrode stepping from $0.5V$ to $2.0V$, with the Top Electrode at ground.

Every pulse step of the set, reset, and read phases has a duration of $1\mu s$. Since every algorithm has a different number of steps, depending on the start and stop voltages and on the voltage increase, the number of measurements during each algorithm changes accordingly, so the available measured data are composed of:

- 16 points in RESET algorithm
- 16 points in SET - ISPVA100
- 13 points in SET - IGVVA100
- 121 points in SET - IGVVA10

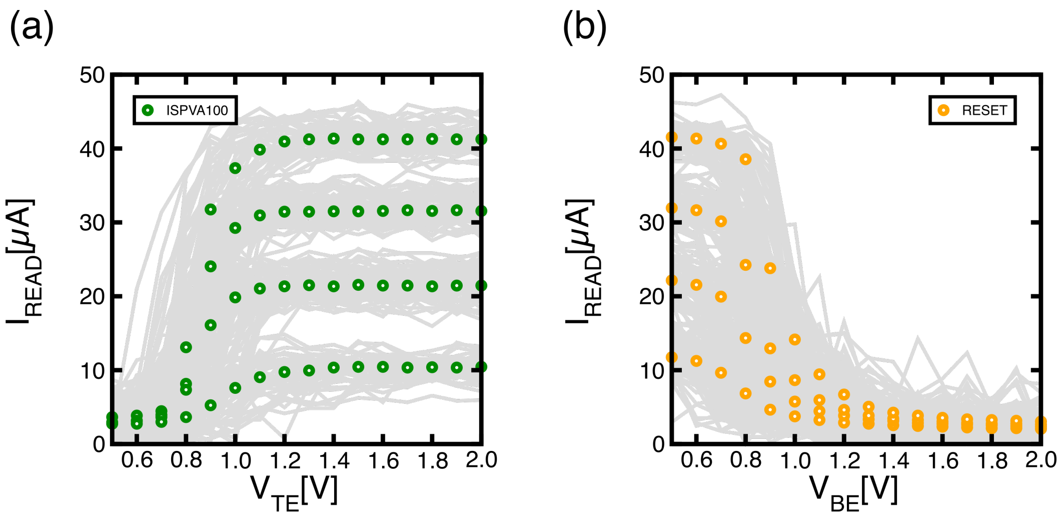

Figure 3.9: Four LRS levels measured adopting ISPVA programming scheme with V_{TE} steps of $100mV$ for (a) SET and for (b) RESET. Each measurement is affected by noise and variability (in grey) and median behavior is extracted (colored).

3.3.2 *C2C*

The main topic of the study is to understand how the different program and verify algorithms perform in endurance cycling, namely, how the programmed value of a certain level under a certain algorithm of the same device varies from cycle to cycle, and how this variability varies from a device to another.

For this scope, endurance cycling measurements were performed on the test chip for the two previously described algorithms: IGVVA and ISPVA. ISPVA features a step size of $100mV$, while IGVVA is analyzed with both step sizes of $100mV$ and $10mV$. After the initial forming process, each device was continuously subjected to reset - set algorithms, for a total of 1000 cycles (Figure 3.10).

The data of the IGVVA10 set algorithm, unfortunately, were unavailable for the whole transition, instead, only a selected part of interest consisting in the last part of the conductance increase is present.

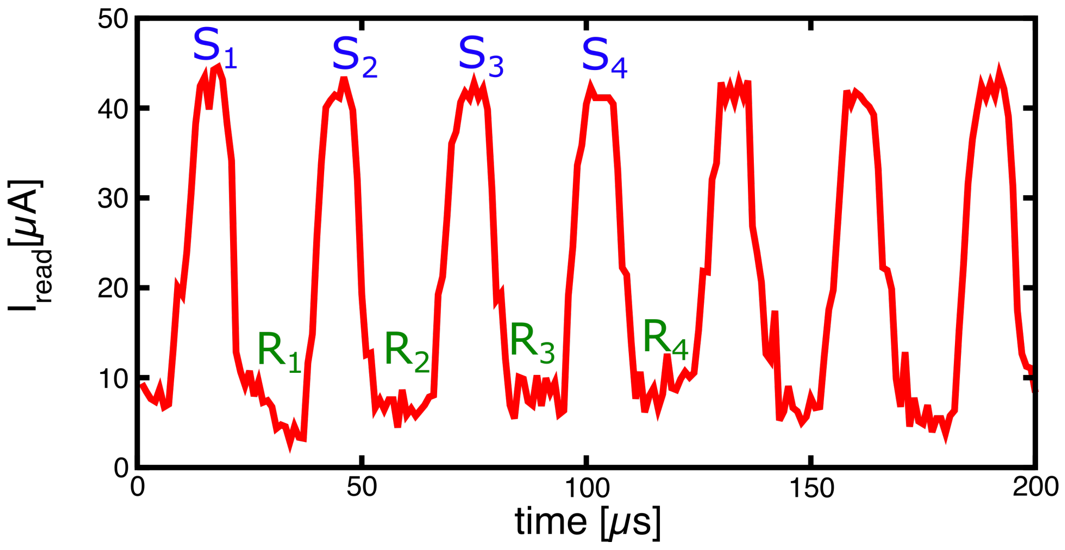

Figure 3.10: Single device cycling through continuous set-reset transitions. Set transition is performed with IGVVA100. The programming introduces C2C variability that affects the conductance value at every cycle.

3.3.3 *D2D*

A second dataset was also analyzed, containing just the single-cycle measurements of reset-set processes for the same three algorithms: IGVVA10, IGVVA100, and ISPVA100, but for 8 different conductive levels. In fact, beyond the original four levels ($I_C = 10\mu A$, $I_C = 20\mu A$, $I_C = 30\mu A$, and $I_C = 40\mu A$), this dataset features the measurements to other four intermediate levels: $I_C = 15\mu A$, $I_C = 25\mu A$, $I_C = 35\mu A$, and $I_C = 45\mu A$. These additional levels allow to increase the information density from $2bits$ to $3bits$ and this can be of great advantage for both data storage and neural network applications. (Figure 3.11a,b)

3.3.4 *Additional D2D*

An additional dataset was available, containing the analysis of the original four levels ($I_C = 10\mu A$, $I_C = 20\mu A$, $I_C = 30\mu A$ and $I_C = 40\mu A$) programmed with the two algorithms ISPVA and IGVVA. In this dataset, for both algorithms the analyzed step sizes ΔV are $10mV$, $20mV$, $50mV$ and $100mV$, in order to understand how the accuracy of the two programming schemes varies with ΔV.

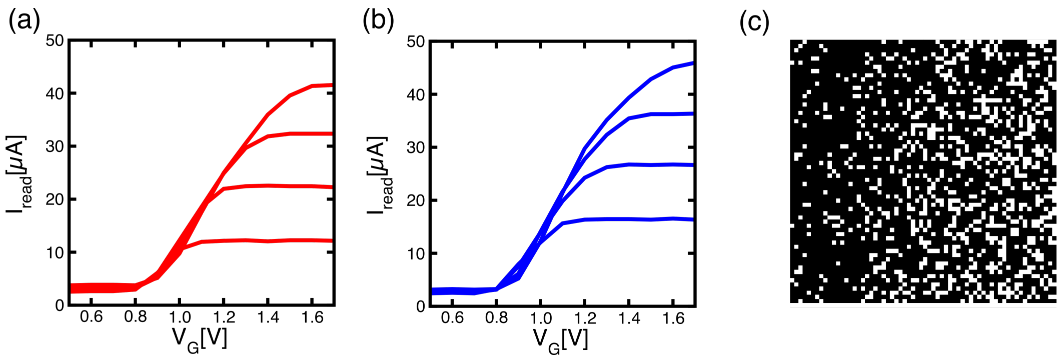

Figure 3.11: (a) Traditional 4 LRS levels and (b) additional 4 LRSs, both programmed through IGVVA100. The additional levels have the following current targets: $I_C = 15\mu A$, $I_C = 25\mu A$, $I_C = 35\mu A$ and $I_C = 45\mu A$. (c) representation of stuck cells (in white) in a 64x64 array. Circa 30% of the total cells are stuck at LRS or HRS position.

3.3.5 *Stuck cells*

Unfortunately, a relatively high number of 1T1R cells in the array was in the so-called *stuck state* and it happens for both HRS and LRS positions. Some other cells presented an incorrect transition starting always from the LRS state and growing to even a higher value, only to be "reset" back again to the LRS value after that, instead of the HRS. A handful of cells, instead, presented wrong behavior just during some of the cycles. The total distribution of the stuck cells within a $64x64$ array is depicted in figure 3.11c.

Depending on the performed analysis and on the values that were extracted, these cells were selectively excluded from the study.

3.4 AFTER SWITCHING - END OF ALGORITHM DISTRIBUTIONS

The first value of interest that was extracted and analyzed from the given experimental endurance dataset is the so-called After Switching (AS) value, i. e. the first point of the set transition that overcomes the desired target current value during the readout (Figure 3.12a).

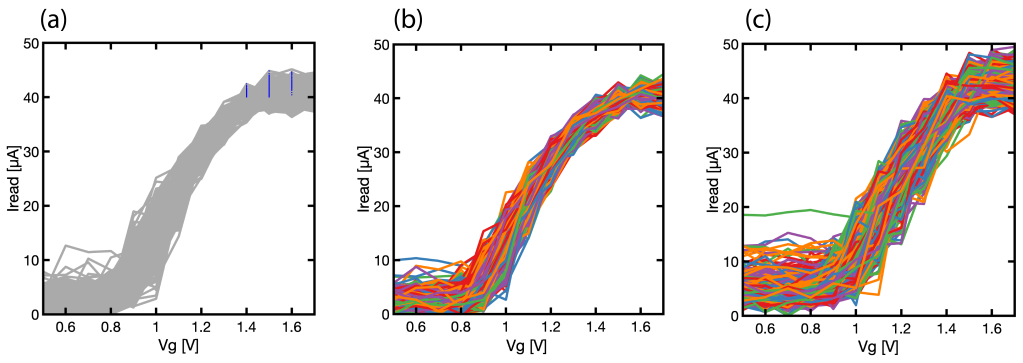

Figure 3.12: (a) Set transition of a single device under IGVVA100 algorithm. After switching values are highlighted in blue. Examples of devices with (b) low variability and (c) high variability over the 1000 cycles endurance measurements.

Figure 3.13 shows the cumulative distributions of the four programmed conductive levels right after the target crossing, namely, from left to right, $50\mu S$, $100\mu S$, $150\mu S$, and $200\mu S$. Each line in the figure represents one of the devices, and each of the points along every line is the *after switching* value at one of the 1000 cycles during the endurance measurements. Ideally, each level should have vertical and overlapping curves near to the leftmost value, but unfortunately, the RRAM exhibits strong C2C and D2D variabilities, easily noticeable in the experimental measurements.

From the same figure 3.13, it is evident how some devices tend to fall closer to the desired target and have a more vertical distribution. This translates in a device having both a relatively low median value of the after switching conductance $< G_{AS} >$, close to the target, and a small standard deviation $\sigma(G_{AS})$ of the desired programmed value. The full endurance cycling test shows that these devices follow roughly the same path every time during the programming transition, as shown in figure 3.12b.

On the contrary, looking at the devices that have a curvier distribution over the 1000 cycles of the endurance test, one can expect a higher standard deviation $\sigma(G_{AS})$ and also an increase in the median value in comparison to the desired target.

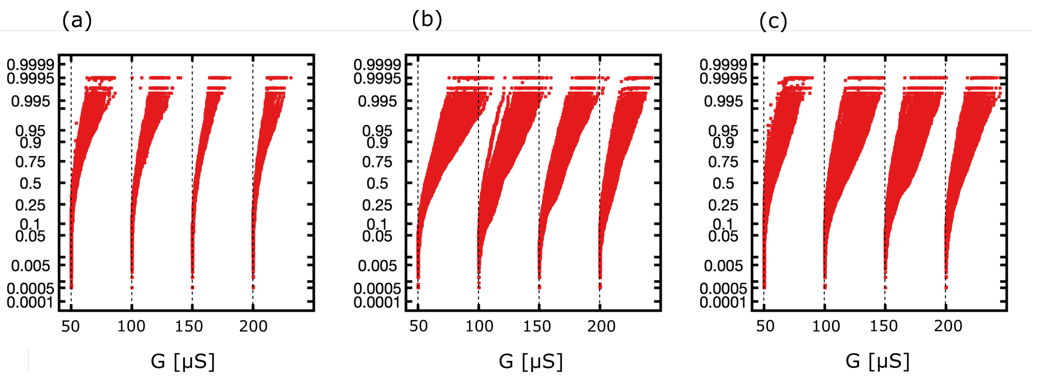

Figure 3.13: cycle-to-cycle (C2C) cumulative distributions of After Switching (AS) conductance values for the four LRS levels. Each curve denotes a device, and each point along every curve represents an AS value in one of the 1000 cycles. (a) IGVVA10, (b) IGVVA100, (c) ISPVA100.

The strong correlation between these two parameters, $\sigma(G_{AS})$ and $< G_{AS} >$, is expected because, since the distribution has a lower bound at I_{target}, a device that sometimes hits higher values will suffer an increase in both the median value and in the standard deviation of its conductance (figure 3.12c).

As one can see from figure 3.14, the majority of the devices find themselves in the middle of the distribution, while a relatively low number of devices has a particularly clean or particularly *dirty* behavior.

Comparing the G_{AS} distributions of each device with the corresponding slope during the set transition shows another interesting correlation: those devices that have a steeper conductive growth are more inclined to exceed the desired target value.

Looking instead at the last value of the SET curve, i. e. the End of Algorithm (EA), two separate effects can be noticed. The first one is the small fluctuation around the target value given by simple read-out noise in order of $\sigma_I = 0.85\mu A$. It is independent of the programmed level or the performed algorithm. The second effect that can be noticed is a small relaxation of the programmed value of the cell, which starts after the algorithm stops the programming pulses and is enhanced by time and temperature. It is a detrimental effect that needs further analysis and experimental measurements in order to better understand its magnitude and its implications.

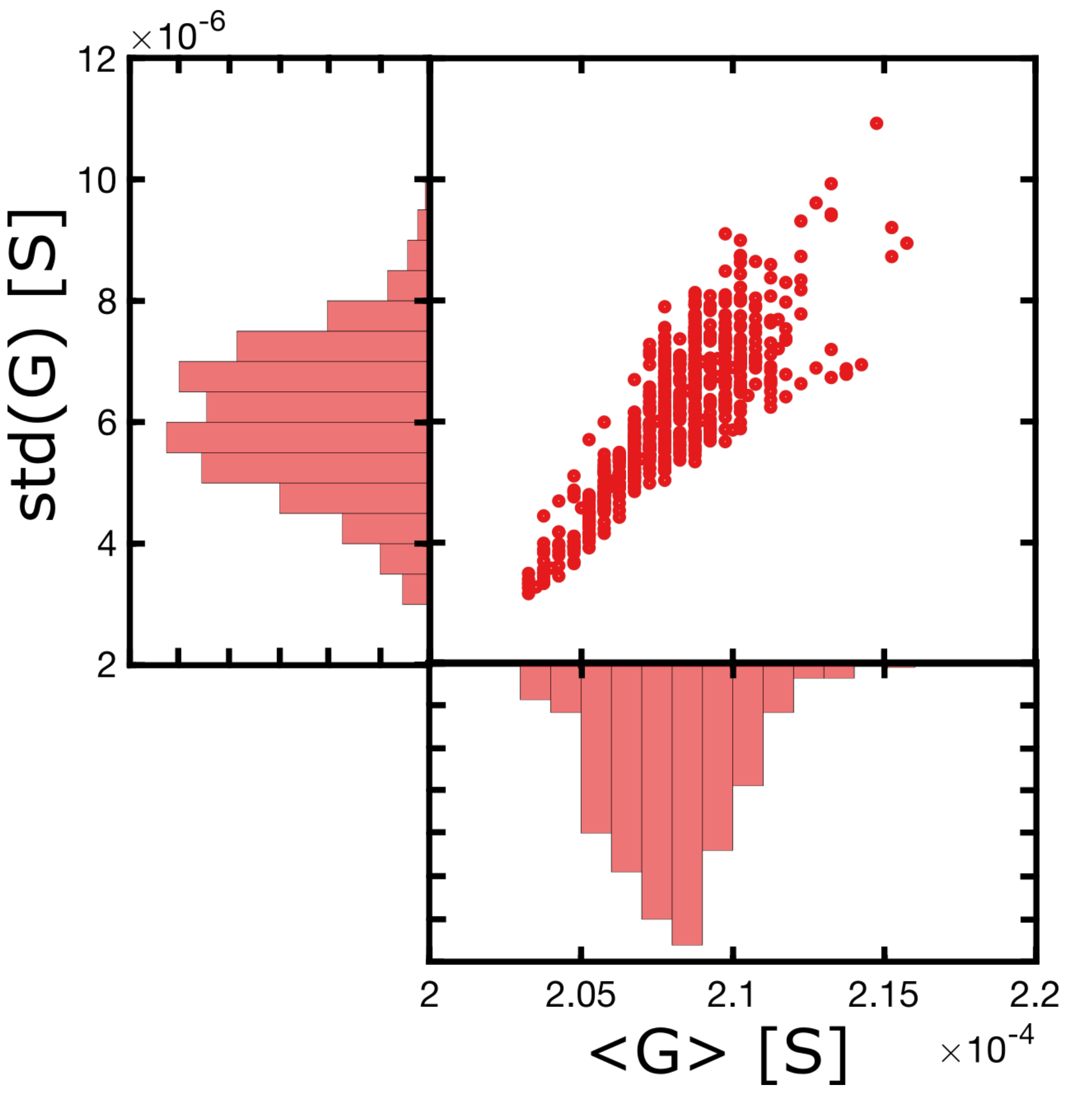

Figure 3.14: (a) cycle to cycle distribution of After switching conductance value, the median value of G_{AS} compared to the standard deviation $\sigma(G_{AS})$ for each device extracted during the 1000 cycles.

Comparing the variability for the three considered algorithms, shown in figure 3.15, highlights how the smallest step of IGVVA10 allows for finer and more precise control of the conductance value, with the median value of the AS conductance variability of circa $\sigma_G = 4\mu S$. For comparison, IGVVA100 has a $\sigma_G = 9\mu S$ and ISPVA100 around $\sigma_G = 6\mu S$.

Indeed, using an IGVVA scheme but with a greater step, such as in IGVVA100, results in worse control of the programmed conductance and increased variability of the after switching values.

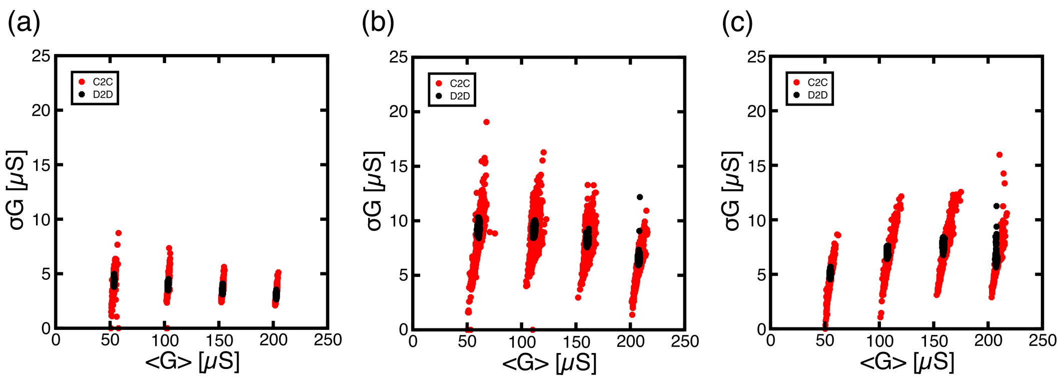

Figure 3.15: C2C (in red) and D2D (in black) variabilities for the four LRS levels for the following three algorithms (a) IGVVA10, (b) IGVVA100, and (c) ISPVA100. IGVVA10 results in the most precise programming algorithm and shows the lowest variability for every level.

ISPVA instead lies right in between the first two algorithms, and it is not influenced by the variation of the step size.

3.5 POSITION AND SLOPE OF THE TRANSIENT

Another parameter that caught the interest of this study is how many program pulses the cell needs to increase the value of its read current up to half of the desired target, represented in figure 3.16a. This figure of merit is called V_{MID} and represents the control voltage (V_G for IGVVA and V_{TE} for ISPVA) for which this read current value is reached. In figure 3.16b,c the cumulative distributions of D2D and C2C for the IGVVA100 LRS4 set transition are depicted, respectively. It can be seen that the C2C distribution is more widespread than D2D, meaning that some devices switch earlier while some devices switch later. Another behavior that can be understood from figure 3.16c is that devices that switch earlier have a more vertical distribution, while devices that tend to switch later are also the ones that show a less stable behavior from cycle to cycle.

On the other hand, the median slope for each set transition (Figure 3.17) shows a nice correlation with the programming accuracy, represented by the standard deviation of the after switching value of the 1T1R conductance. It is expected behavior since a device that grows faster tends to overcome the target value by a

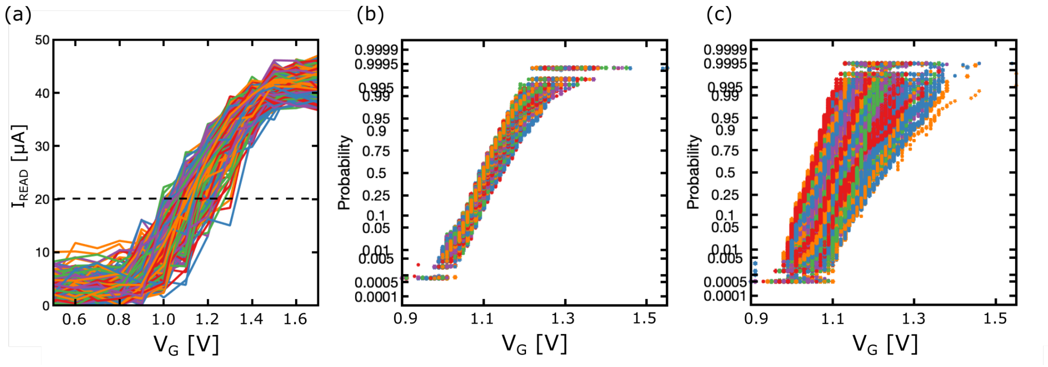

Figure 3.16: (a) Example of how V_{MID} is extracted. It is the voltage Vg for which the read current I_{read} reaches half of the target current. Cumulative distribution of V_{MID} for IGVVA100 programming scheme extracted from the experimental data. (b) On the left, each curve represents a cycle, and each dot on the curve represents a device. (c) On the right, instead, the C2C behavior is shown, meaning that each curve represents a device, and the inclination of the curve represents the variability of that device.

larger amount, compared to a device that increases its conductance more gradually. These results show that in order to have a more precise and accurate programmed value, it is important to be able to control the growth rate of the cell, and so choose an algorithm such as IGVVA10, that by performing smaller programming steps gives greater control over the whole transition.

3.6 NOISE

Each value of the dataset presents a fluctuation in amplitude, that can be treated as a noise happening during the read phase. Its impact is important because, in some finer algorithms, such as IGVVA10, the amplitude of this fluctuation is high enough to be the main cause of the variability in the distributions of the programmed G_{AS} values. Stuck cells resulted to be particularly useful to understand and quantify the noise affecting the device. In figure 3.18a, the set transition of the $LRS4$ level programmed through IGVVA10 from the device-to-device (D2D) dataset was used to study this phenomenon. This particular set was chosen because it shows the behavior of 1024 cells during a single set cycle and

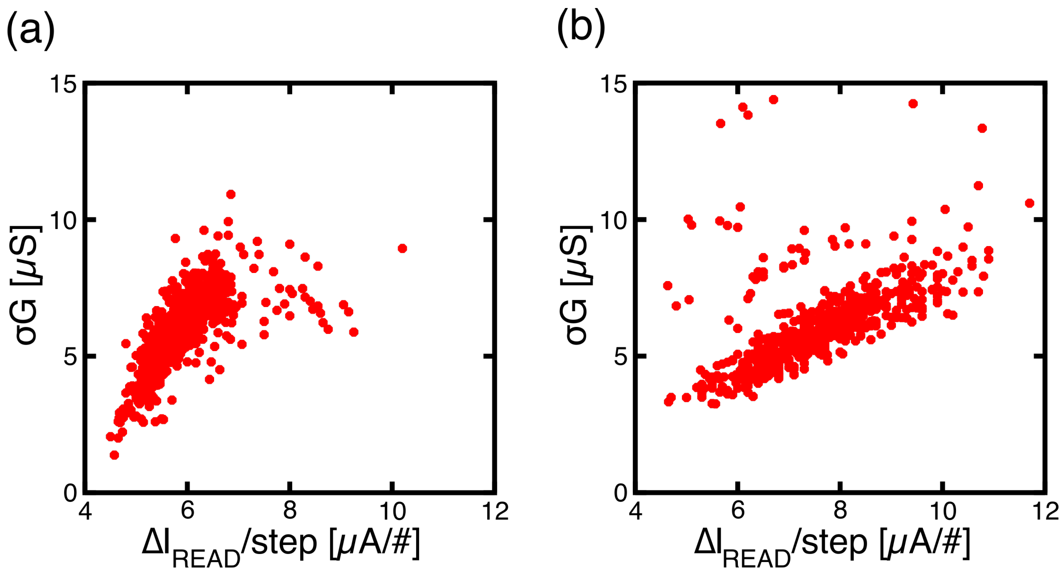

Figure 3.17: Correlation between the median slope of each device (C2C), i. e. how many μA the conductance increases for a single programming step, and the standard deviation of the programming accuracy. Each point represents a device. (a) shows IGVVA100 and (b) ISPVA100 programming schemes, both for LRS4 with target $I_{read} = 40\mu A$.

each has 121 points. Unfortunately, many of the cells in the array are affected by previous reset problems or are in a stuck status that can be both LRS or HRS. In the case highlighted in figure 3.18a, the cells that have the first value of the read current higher than the target current $I_{target} = 40\mu A$, are not affected by the programming algorithm but are just periodically read during the verify phase with $V_{TE} = 0.2V$ and $V_G = 1.7V$. This means that the conductance value of each cell ideally remains unperturbed during the whole process, and the "noise" that is present is just due to a statistical variability of the readout circuitry.

In particular, of the initial 1024 cells of the array, 85 cells are permanently stuck at higher current values, and calculating the standard deviation of each curve around their median value shows that some cells are affected by a slightly lower noise while some other by a slightly larger. The distribution of this noise from device to device has a Gaussian shape, as shown in figure 3.18b.

Instead, looking at the stuck LRS cells of the first dataset with cycling endurance data allows us to quantify how the noise impacts a single cell across a long-range of measurements. Turns out that the standard deviation has about the same

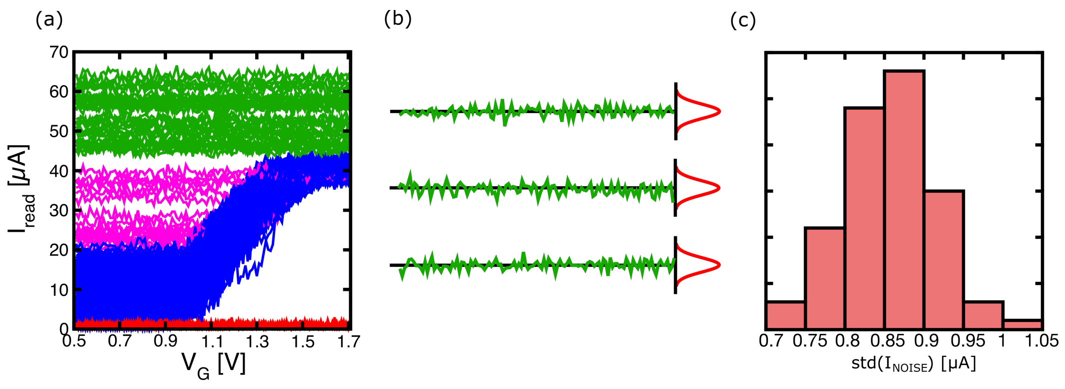

Figure 3.18: (a) set transition for IGVVA10 $LRS4$ (target $I_{read} = 40\mu A$) Some of the cells result in a stuck state Particularly stuck in LRS(in green) and stuck in HRS (in red). They are not affected by programming pulses. (b) Each stuck cell will show a fluctuation around its median value, and it differs from device to device and also from cycle to cycle. (c) The overall histogram of the standard deviations of each stuck cell expressed as $\sigma_{I,read}[\mu A]$.

amplitude, and that the behavior along the cycles is very similar to the behavior across the different cells, with no evident difference between C2C and D2D.

3.7 EFFECT OF PROGRAMMING STEPS WIDTH ON PROGRAMMING TIME AND VARIABILITY

Since the chip under test cannot be programmed in parallel, but only sequentially cell per cell, the time needed to fully reprogram the $4 - kbit$ array can be an important factor when choosing the algorithm. If one wants to set all the 4096 cells of an array to the highest LRS value and obtain the cleanest result, the choice would be to rely on IGVVA10, which requires 121 steps from $V_G = 0.5V$ to $V_G = 1.7V$ with an increase of 10mV between each step. The same amount of Verify steps is needed. All steps have a duration of $1\mu s$ and the delay between steps is considered negligible. This translates in a total time of $2 \times 121 \times 1\mu s = 242\mu s$ for each cell to be programmed. To set the whole array circa one second is needed, which is roughly a factor 10 increase compared to a less accurate algorithm such as IGVVA100 that takes a maximum of 13 steps per programming.

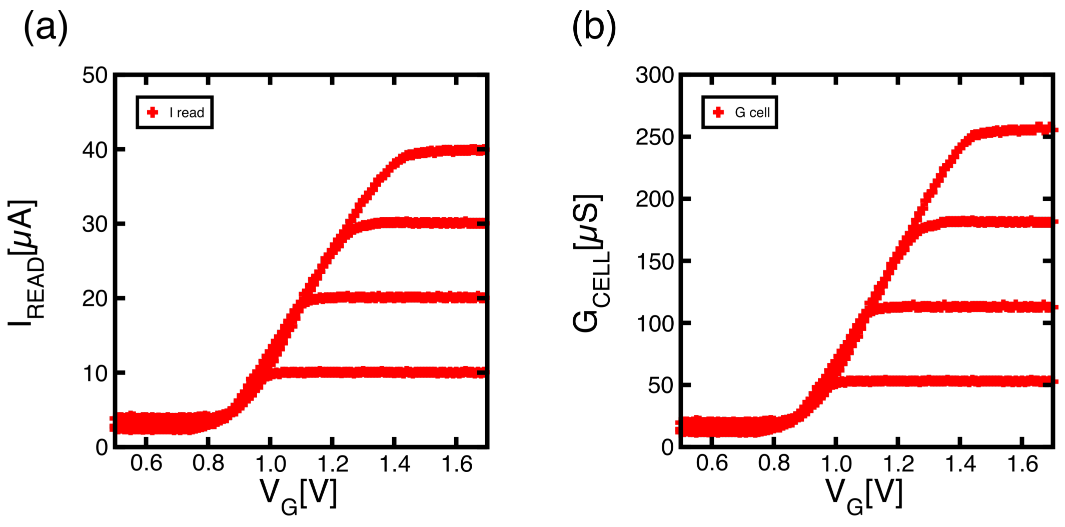

Figure 3.19: (a) Four *LRS* levels programmed through IGVVA10, namely $10\mu A$, $20\mu A$, $30\mu A$, and $40\mu A$. (b) The corresponding conductance of the RRAM element extracted from the measurements.

The effect of reducing the step size has different impacts whether it is applied to the ISPVA algorithm or IGVVA. On the former, the difference does not influence the programming variability, and the HRS to LRS transitions are always abrupt. On the other hand, the variability of the programmed levels with the IGVVA scheme is influenced by the size of the steps. The finer they become, the closer the final conductance value arrives at the desired target. This directly impacts the results in applications that require precise levels, such as the implementation of a hardware neural network.

3.8 RRAM TRANSITION VALUES EXTRACTION

The read current values and the I-V MOS characteristic enable the extraction of many useful quantities during the whole set and reset transitions. For example, with few assumptions, it is possible to extract the trend of the voltages around the RRAM and currents that flow during the programming pulses, and also how much each pulse impacts on the change of the conductance. The following analysis was initially performed on the median values of the curves, in order to abstract the

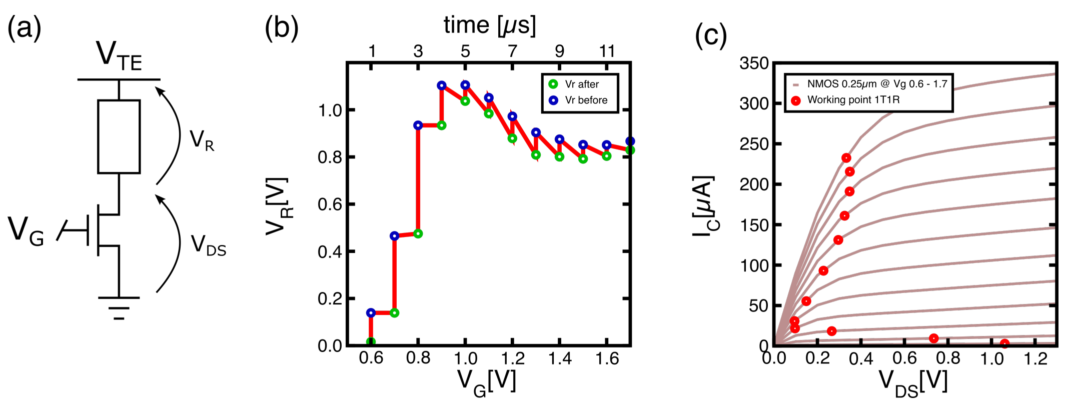

Figure 3.20: (a) Representation of the 1T1R cell during the programming phase. (b) V_R voltage across the RRAM before and after each applied programming pulse P_j. (c) Working point extracted through the intersection of the conductance of the cell and the characteristic of the transistor, for each programming pulse in the IGVVA100 programming scheme for level $LRS4$ corresponding to $I_{read} = 40\mu A$.

trends from the variability impacting each cell and each cycle, and from the noise affecting the data samples.

As already briefly introduced in section 3.3, from the available data of I_{read} it is possible to extract the conductance of the RRAM cell considering also the non-null channel resistance of the transistor during the readout verification phase of the algorithm.

For instance, let us consider a single device with a low variability under the IGVVA100 programming scheme.

After averaging the read current for all the cycles it is straightforward to extract the value of the G_{RRAM}, which is higher than the total equivalent conductance of the cell G_{CELL}, which comprehends also the NMOS resistance $r_{ds,on}$. By looking at figure 3.19, it is noticeable how the levels of cell conductance are not equally spatiated, and the more the level increases, the more the distance from the previous layer increases. This can be explained by the partition of G_{RRAM} and $r_{ds,on}$. Since the NMOS resistance is of fixed value, to reach a higher level the conductance of the cell has to increase more.

From the available data can be deduced that every value (verify step V_j) corresponds to the conductance right after the previous programming pulse P_j but also to the conductance at the beginning of the next programming pulse P_{j+1}. If between two steps j and $j+1$ the conductance grows from G_j to G_{j+1}, it means that the pulse P_{j+1} had an impact on the RRAM causing a variation of ΔG_{j+1}.

Intersecting each conductance value G_j with its previous and its next programming pulse allows reconstructing all currents and voltages that act on the cell during the whole transition. (Figure 3.20).

4 STATISTICAL ANALYTICAL MODEL OF 4KBIT RRAM ARRAY

This chapter presents the development of a statistical model with the purpose of predicting the programming variability of the HfAlO RRAM array that was described and analyzed in the previous chapters. Different program and verify algorithms are implemented on top of the proposed model and the simulated results are compared with the experimental data measurements highlighted in the previous chapter, in terms of median value, programming variability, and statistical spread. After the extraction of fitting parameters for all the analyzed programming schemes, a Monte Carlo simulation implementation will be discussed, with the aim of fitting the experimentally extracted variability distributions in terms of cycle-to-cycle and device-to-device variabilities. Then, a similar approach will be adopted to study and analyze a model for the reset transition, through a similar program and verify programming scheme. In the end, a brief introduction is made about the case study of neural network implementation, and about the structure of the model's algorithm that will be used in the following chapter to perform large-scale inference simulations.

4.1 INTRODUCTION

Through this section will be proposed the development of a stochastic analytical model for the switching transition of an HfO_x-based RRAM. Inserting the aforementioned model in a real case scenario of one Transistor - one Resistor (1T1R) structure will allow to simulate and analyze, with a unique set of parameters, the behavior of the device under different applied program and verify algorithms, starting from the IGVVA and ISPVA discussed in the previous chapter (Section 3.2). In the first place, the implemented formulae will be discussed along with the simplifications applied to maintain the compactness of the model. Then, a set of parameters will be extracted to fit the median conductance transition for each of the proposed algorithms. After this, a statistical variability will be added to shape the distribution of the parameters in a Gaussian form, and a Monte Carlo method analysis will be performed on the model to reproduce all experimental results in terms of After Switching (AS) and End of Algorithm (EA) values considering the cycle-to-cycle (C2C) and device-to-device (D2D) variability distributions. This work will open the possibility to study and research for new and different programming schemes to further reduce the variability and enhance the precision of multilevel programming, considering also the time and the power consumption for the whole operation.

A similar approach will be applied to deduce and analyze an equation for the reset transition's model, for the same RRAM device.

The models derived over this chapter will be implemented as a case study in a simulation of a hardware neural network, with particular emphasis on the effect of the programming variability on class accuracy reachable during the inference phase of the network.

4.2 MODEL DESCRIPTION

The goal for this model is to accurately describe the increase of the conductance value of an RRAM device under the given initial conditions of conductance G_0, the voltage across the device V_R, and the flowing current I_R. The model should be able to reproduce deterministic and statistical nonidealities, such as both C2C and D2D distributions, and the read current noise. It should be compact enough to enable the simulation of a high number of devices (in the order of 1000 or 10000), even with a home computer requiring a small amount of time in the order of few minutes.

4.2.1 *Differential Equation*

The starting point for the model is the rate equation for the filament growth of the RRAM device described in section 2.2. It is straightforward to translate the filament growth into conductance growth using the following steps:

$$\frac{d\phi}{dt} = Ae^{\frac{-E_A}{kT}}$$

$$G = \frac{1}{\rho}\frac{\pi(\frac{\phi}{2})^2}{L} = \beta\phi^2$$

$$\phi = \sqrt{\frac{G}{\beta}}$$

$$\frac{dG}{dt} = 2\beta\phi\frac{d\phi}{dt}$$

$$\frac{dG}{dt} = A^*\sqrt{G}e^{\frac{-E_A}{kT}} \tag{4.1}$$

The resulting equation (4.1) expresses the rate of the conductance increase over time. A^* is a constant factor regrouping the geometrical parameters β and the pre-exponential factor A of the rate equation of ϕ, G is the actual value of conductance, E_A is the activation energy, k the Boltzmann's constant and T the temperature of the device expressed in Kelvin. When integrated, the equation (4.1) gives as the output the conductance of the RRAM cell.

The activation energy E_A is affected by the energy barrier lowering effect due to the applied voltage V_R across the device, $E_A = E_{A0} - \alpha q V_R$ (Section 2.2). The temperature T, instead, is subjected to a rise due to the Joule's effect, $T = T_0 + R_{th}P$, being $P = V_R I_R$ the power flowing across the device at any time and R_{th} the thermal resistance of the conductive filament. For the sake of simplicity and the compactness of the proposed model, the thermal resistance will be assumed to be constant over the whole conductive transition. To summarize, the final rate equation of the model is expressed as follows:

$$\frac{dG(t)}{dt} = A^*\sqrt{G(t)}e^{-\frac{E_{A0}-\alpha q V_R(t)}{kT_0\left(1+\frac{R_{th}P(t)}{T_0}\right)}} \tag{4.2}$$

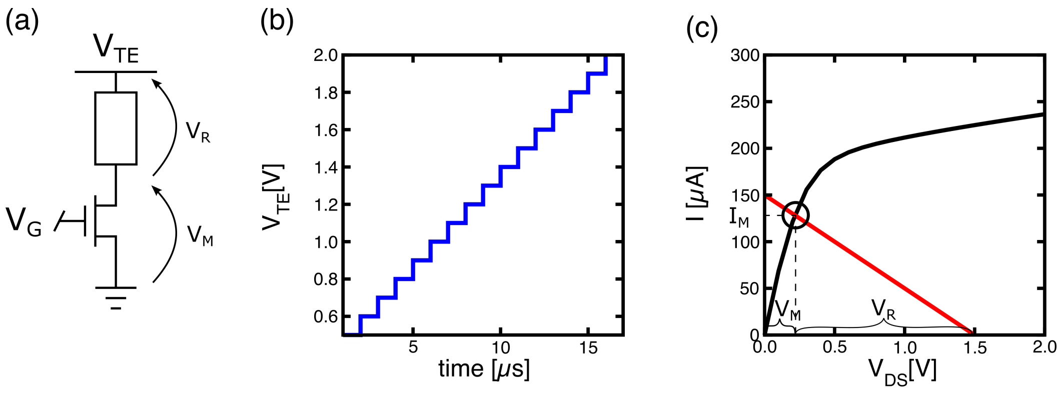

Figure 4.1: (a) Schematic representation of a 1T1R structure. V_R is the voltage across the device that has to be calculated from the working point, V_M is the voltage across the NMOS; (b) Waveform applied to the top electrode V_{TE} during ISPVA programming scheme; (c) Computation of V_R and of the current flowing through the series by finding the working point, i. e. the intersection between the NMOS characteristic for a particular V_G (in black) and the load curve of the RRAM device (in blue).

4.2.2 *Integration*

The rate equation (4.2) is an *Ordinary Differential Equation* and it contains both the integral function $G(t)$ and the parameters that are dependent on time, such as $V_R(t)$ and $I_R(t)$. Thus, the integration of equation (4.2) is not straightforward, and it cannot be resolved analytically.

By knowing the initial conditions $G^{(0)}$, $V_R^{(0)}$, $I_R^{(0)}$, an iterative method can be adopted to solve the equation.

Since the purpose of this work is to simulate the switching behavior of a full 1T1R cell, henceforth for *device* will be intended a *series* of an RRAM device and an NMOS transistor as depicted in figure 4.1a, with the bottom electrode of the resistive memory connected to the drain terminal of the transistor.

The 1T1R structure complicates the implementation of the model because, to calculate the voltage V_R and the current I_R of the RRAM, a series of iterative steps must be performed. Indeed, by knowing the overall applied voltage V_{TE} across the device (Figure 4.1b), the RRAM conductive value G, and the I-V characteristic curve of the NMOS for the various V_{DS} and V_G, it is possible to find the working point of the system and extract the desired values of the RRAM cell needed by the model (Figure 4.1c). Then, it is possible to proceed with the integration of the rate equation (4.2).

To summarize, the following steps have to be executed to perform one integration step.

- Compute the device load curve $I_R^{(k)} = G^{(k-1)} V_R = G^{(k-1)}(V_{TE} - V_{DS}^{(k-1)})$ using the values of the previous step and intersect it with the NMOS I-V characteristic to extract the new value of $V_{DS}^{(k)}$.
- Update the new value $V_R^{(k)} = V_{TE} - V_{DS}^{(k)}$.
- Compute the new current value $I_R^{(k)} = I_{MOS}(V_G^{(k)}, V_{DS}^{(k)})$. Since the RRAM and the NMOS are in series, the intersection between the two curves represents the working point of the system.
- With the updated $V_R^{(k)}$ and $I_R^{(k)}$, compute the conductance update $\Delta G^{(k)}$ using the formula (4.2).
- Compute the conductance value $G^{(k)} = G^{(k-1)} + \Delta G^{(k)} \Delta t$

Particular attention must be paid to the last step. The integration method here applied is called *Forward Euler*'s method, and it is the most basic explicit method for numerical integration of ordinary differential equations, given the initial values. But, especially for stiff equations such as the equation (4.2) implemented in this work, this method can produce unstable output, i. e. the numerical solution grows very large while the exact solution should not, with the consequent divergence of the algorithm.

This divergence problem can be avoided most of the time by decreasing the integration timestep Δt. Considering that all the programming algorithms in this work have a step duration of $1\mu s$, Δt should not be higher than that. In most cases, $\Delta t = 1ns$ is enough to guarantee convergence. This means that, for example, between two consecutive increases of the V_{TE} voltage during ISPVA programming, a total of $N_{\#} = \frac{1\mu s}{1ns} = 1000$ integration steps will be performed. Further decreasing Δt directly impacts the performance of the model.

	IGVVA10	IGVVA100	ISPVA100
Forward Euler	0.14s	1.41s	1.32s
Heun	0.14s	0.16s	0.95s
ODE23	0.09s	0.12s	0.12s

Table 4.1: Execution time comparison table between the three proposed ODE solvers, for the three analyzed algorithms. Forward Euler and Heun's methods are computed by using the maximum Δt possible for convergence. ODE23 solver is the fastest overall, and while it gives just a minor speedup for IGVVA10, it results necessary for IGVVA100 and ISPVA100, enabling a high-speed simulation of a large number of cells.

A more advanced and improved version of Euler's method, *Heun's method*, can be implemented. It is a second-order method that uses a two-step procedure to increase the approximation accuracy of the result. First, an intermediate step is computed: $G^*_{i+1} = G_i + \Delta t f(t_i, G_i)$, in which $f()$ is the right hand term of the equation (4.2) and G^* is an intermediate value of conductance at the next step, called *predictor*, calculated exactly as with Forward's Euler method. Then, the effective conductance value is computed with more accuracy: $G_{i+1} = G_i + \frac{\Delta t}{2}\left[f_n + f(t_{i+1}, G^*_{i+1})\right]$

By using Heun's integration method, it is possible to increase the value of the timestep Δt even of factor 10 without causing instability, and consequently decrease of the same factor the computation time.

To further decrease the computational time of the algorithm, a more complex algorithm will be used to solve the ordinary differential function, *ode23* embedded in the MATLAB suite. Ode23 algorithm features two single-step formulas, one of the second-order and one of the third-order. It is faster to compute than the previously discussed solvers because it features also dynamic timesteps, that are adjusted at every algorithm's step, in order to guarantee convergence. Since equation (4.2) has a steep change at the edge but after that follows a relatively flat behavior, overall the ode23 solver is the fastest between the analyzed algorithms.

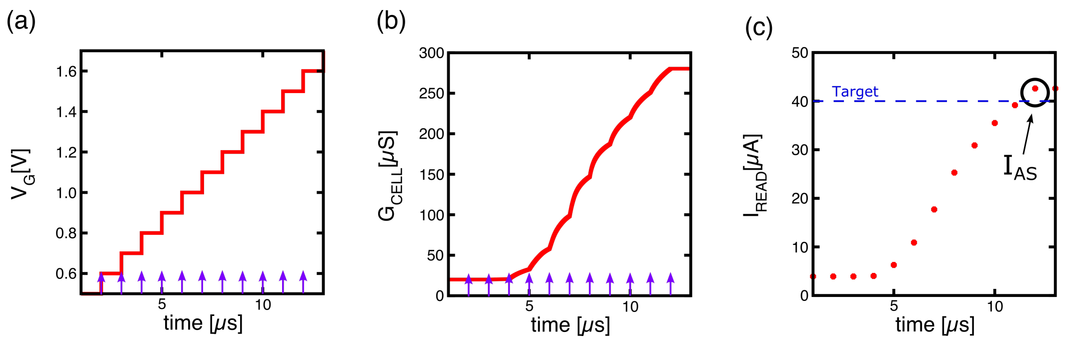

Figure 4.2: IGVVA100 - $40\mu A$ target. (a) Waveform applied to the gate terminal (program pulses P, in red) and verify pulses (in violet); Evolution in time of the conductance value of the RRAM cell (in red); (c) Read current of the 1T1R structure (in red). The first value that overcomes the target (blue dashed line) represents the After Switching (AS) current value.

A comparison of the execution times with the three solvers and for the three algorithms can be seen in table 4.1.

4.2.3 *Program and verify algorithms*

All of the program and verify algorithms explored in chapter 3 consist in two separate phases constantly alternating one to each other. In the first phase, the *program* (P) phase, the cell's conductive value is programmed by applying a proper combination of voltages to the device; the following phase, the *verify* (V) phase, is used to read and monitor the conductive value reached during the previous P phase, by reading the device's current with a defined applied voltage. If during the V phase the current overcomes a previously defined threshold value, the cell is considered correctly programmed and the algorithm stops. On real hardware, both of these steps must be accurately defined and executed, but in this simulation only the program P part will be implemented by applying the correct voltages with correct timings to the model, leaving the verify phase to be easily simulated via software.

An example programming of the level LRS4 with IGVVA100 is depicted in figure 4.2. With every programming pulse of the duration $1\mu s$, the conductance of the cell grows and stabilizes at a certain level limited by the compliance current of

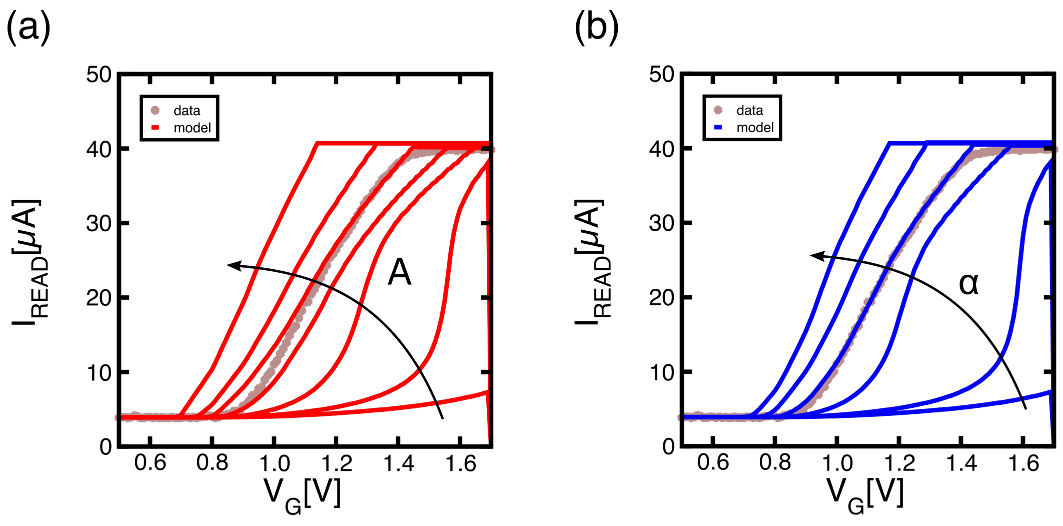

Figure 4.3: IGVVA10 - 40μA target. Read current calculated by (a) keeping α at a fixed value and by increasing A, the curve begins to rise earlier and with a higher slope; (b) the same can be seen by fixing A and increasing the value of α.

the current programming step (Figure 4.2b). After the programming pulse, the verify phase is applied (represented as violet pulses in the figure), and the final conductive value is sampled as I_{read}, depicted in figure 4.2c. If the read current overcomes the target of the desired LRS level, the algorithm ends.

For both the two analyzed algorithms, ISPVA and IGVVA, the structure of the model's implementation remains the same. In fact, during ISPVA's P phase, a constant V_G and a ramp of V_{TE} are supplied to the device's terminals, while during IGVVA's P phase the opposite: a constant V_{TE} and a ramp of increasing V_G. Both programming schemes feature the same V phase with $V_G = 1.7V$ and $V_{TE} = 0.2V$.

4.3 parameters' tuning

It is now the time to fit the experimental data, and the first step to do so is to simulate the behavior of a nominal cell, chosen to be represented by the median curve of each algorithm and each level. Note should be taken that the median curve does not represent a real device and it itself might slightly vary between different levels or between two different datasets, but it is a great starting point to find an acceptable range for the free parameters of the model described by equation

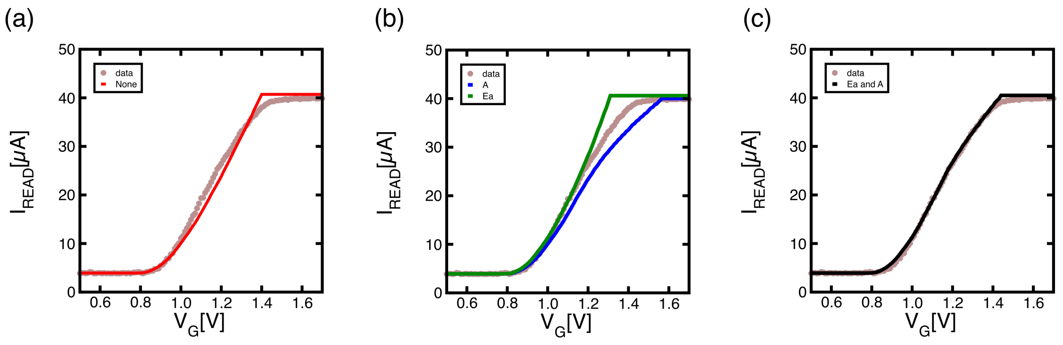

Figure 4.4: IGVVA10 - $40\mu A$ target. Effect of the modulation of Ea and A on the read current. (a) Parameters without modulation; (b) Only A (in blue) or E_A (in green) are modulated; (c) Both parameters are modulated.

(4.2), namely A, E_{A0}, α and R_{th}. Instead, $k = 1.38 \times 10^{-23} \left[\frac{m^2 kg}{s^2 K}\right]$, $T_0 = 300[K]$ and $q = 1.6 \times 10^{-19}[C]$ are constans.

The activation energy E_{A0}, as described in the literature, should be around $1eV$ The barrier lowering coefficient should be lower than 1. The thermal resistance R_{th} instead depends on the geometrical size of the conductive filament, on the material, and the density of oxygen vacancies inside the oxide layer. In this discussion, an accepted value for the thermal resistance is around $R_{th} = 0.1 - 1 \left[\frac{K}{\mu W}\right]$. The preexponential factor A will be chosen accordingly to adjust the amplitude of the rate equation.

By trying different combinations of A and α, it was noticed that these two parameters have a correlated effect on the algorithm: by increasing A the conductance grows faster, and the same happens by increasing α, as shown in figure 4.3.

4.3.1 *Ea and A modulations*

To better fit the experimental curves, two effects are added to the model, with the following hypothesis.

As the filament grows and the conductance of the cell increases, the energy needed for a further increase of the filament size slightly decreases. The magnitude of the whole variation is considered to be around $0.1eV$, and this effect is added

	$A\left[\frac{\sqrt{S}}{s}\right]$	$\alpha[V^{-1}]$	$E_A[eV]$	$R_{th}\left[\frac{K}{\mu W}\right]$
IGVVA10	1×10^{13}	0.5	1.2	0.8
IGVVA100	2×10^{10}	0.5	1.0	0.8
ISPVA100	3×10^{13}	0.5	1.2	0.1

Table 4.2: Selection of parameters that best fit each algorithm.

to the formula as $E_A = E_{A0} - 400\frac{eV}{S} \cdot G$. G is the conductance of the cell and E_A linearly decreases from circa $1.2eV$ to $1.1eV$ during the transition.

On the other hand, as the filament increases in size, the number of ions that can be moved from the edges to the filament, and thus contribute to the filament expansion, is constantly decreasing, highly slowing down the process over a certain threshold. This second behavior is modeled by modulating the preexponential factor with a rapidly decreasing function: $A = A_0 \frac{1}{1+\left(\frac{G}{150\mu S}\right)^8}$. These two effects are summarized in figure 4.4.

4.4 FITTING OF MEDIAN CURVES

After a coarse decision of the parameters, it is possible to finely tune them and fit the median curves of each algorithm.

Figure 4.5(a-i) shows the model in action with the fourth LRS level ($40\mu A$) for all three algorithms. The goal of this part is to fit precisely the experimentally measured data, and these fittings were obtained by using the parameters listed in table 4.2. As one can notice, the values slightly change from an algorithm to another to obtain a complete overlap between median curves and the model. The left column represents the read current given as the output during the verify phase after each programming step of the algorithms. The column in the middle shows the voltage V_R across the RRAM device during the programming, while the rightmost column shows the current during the P phases. Note that when the algorithm reaches the desired read current target, it stops the P phases and only the verify phases continue. This means that the last part of the transient after

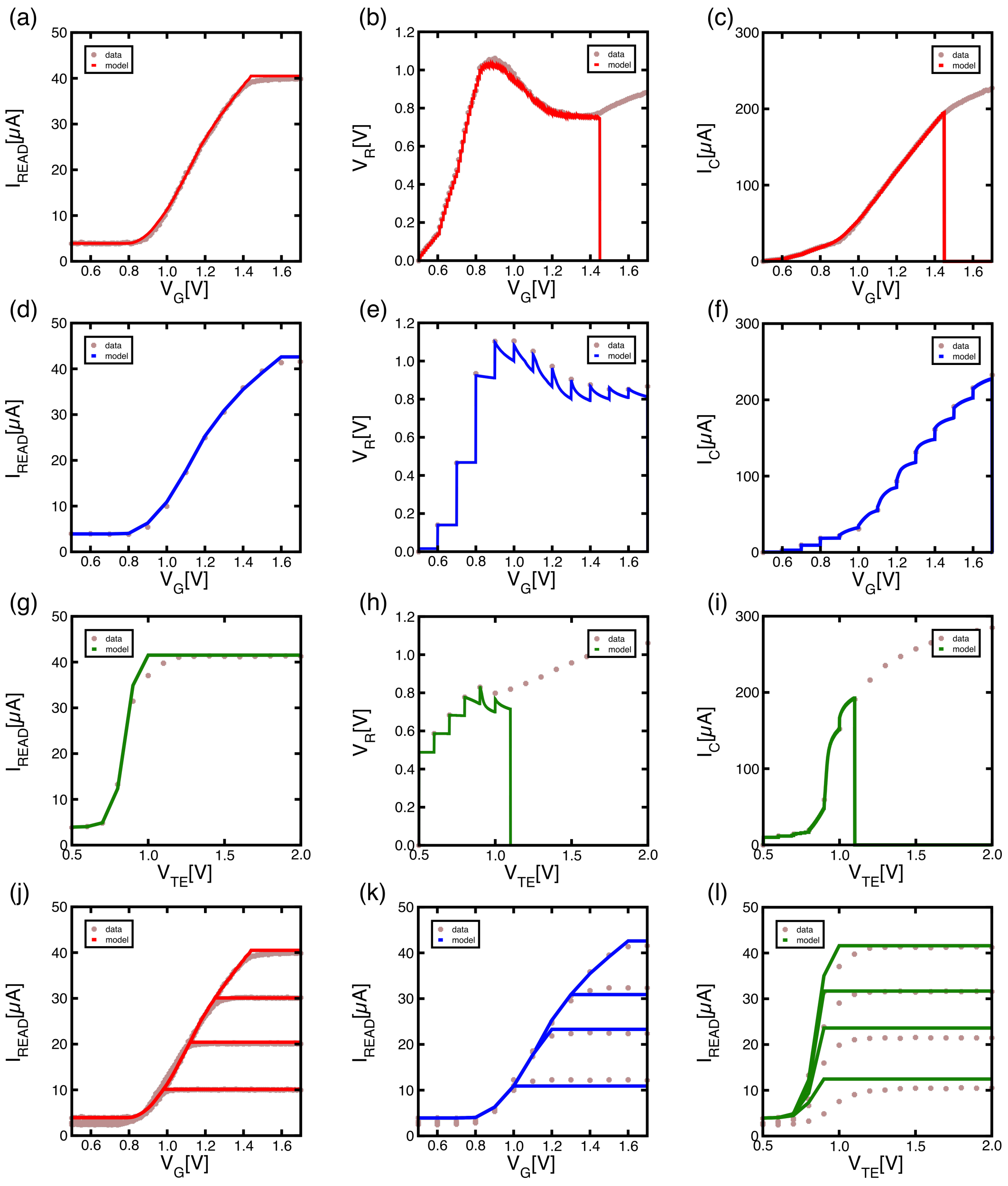

Figure 4.5: Fitting the median curves with parameters shown in table 4.2. (a-c) IGVVA10, (d-f) IGVVA100, (g-i) ISPVA100. (j-l) instead show the fitting of all the 4 LRS levels.

the AS shown in the graphs actually does not exist, but it is just the output of the data extraction process. The last row of the figure, 4.5(j-l), depicts the read current calculated for all four LRS levels.

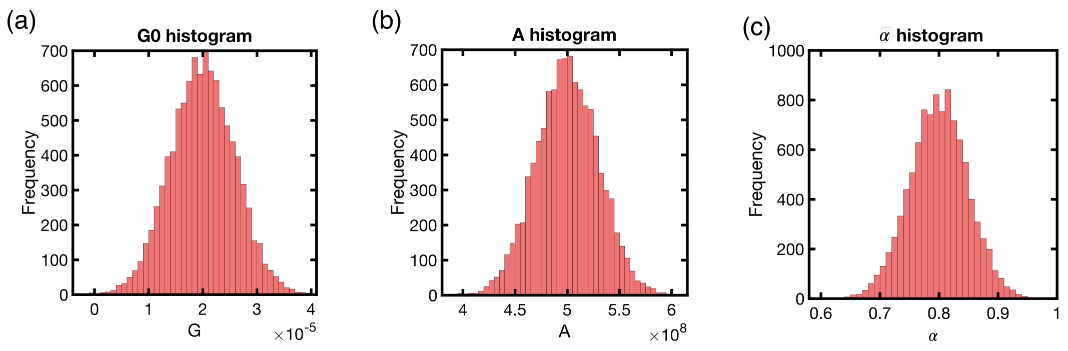

Figure 4.6: Histograms of the variability added to the devices' parameters, (a) the initial conductance value G_0, (b) the pre-exponential factor A and (c) the parameter α. For every device, a value having these probability distributions is selected, and it introduces variability in the model.

4.5 DISTRIBUTION FITTING THROUGH MONTE CARLO SIMULATIONS

Since the model is able to reproduce with high precision the set transition of a single curve, the next step is to extend its functioning to the simulation of a full array of 1T1R devices by using the previously described equation and to fit the statistical distributions extracted in chapter 3.

To do so, the parameters will be treated as statistical distributions having a mean value, corresponding to the previously tuned parameters, and a standard deviation. For simplicity, the distributions will have a Gaussian shape.

The generic parameter X of a device d can be written as $X_d = \mu_X + \sigma_{Xd}$, with μ_X being the median value, or the nominal value of the parameter found in the previous sections, and σ_{Xd} the standard deviation of that parameter (Figure 4.6). By doing so, each device will have its unique set of parameters, and the median value will remain constant.

By tuning the amplitude of σ_{Xd} it is possible to recreate accurately the experimentally measured distributions.

On top of that, a second fluctuation can be added around the just defined X_d, so that at every programming cycle the device suffers from a variability around its median value, that differs from the fluctuations of other devices. $X_{d,c} = X_d + \sigma_{Xc}$. σ_{Xc} is sampled once per device, and it is the standard deviation responsible for a

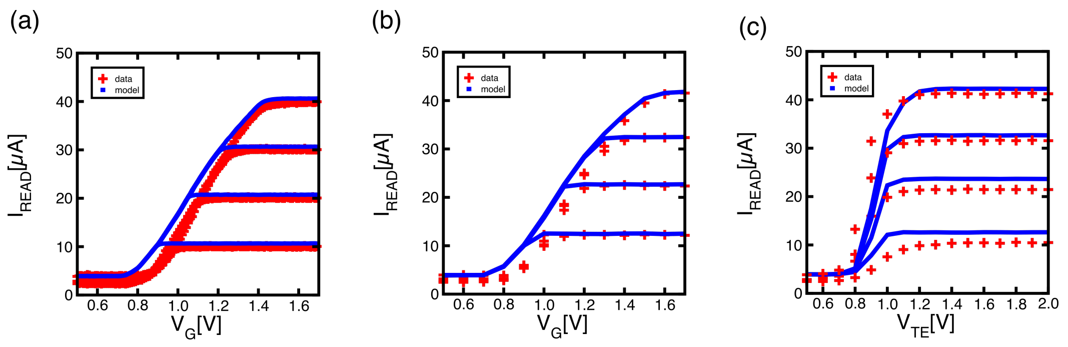

Figure 4.7: Simulation of the read current featuring four LRS levels, namely $10\mu A$, $20\mu A$, $30\mu A$, and $40\mu A$, for the three algorithms (a) IGVVA10, (b) IGVVA100 and (C) ISPVA100.

variation during the different cycles. If the value of σ_{Xc} is small, the device will follow always a similar path, varying very little between different cycles. Instead, if σ_{Xc} is big, that particular device will suffer from high C2C variability. This behavior is observed and described during the analysis of the experimental data in the previous chapter.

The electronic noise seen during the current read-out is also modeled into the simulation, and it can be expressed as a $\mu_{\sigma I} = 0.85\mu A$, i. e. the median value of the Gaussian distribution of the electronic noise. The standard deviation $\sigma_{\sigma I}$ is chosen to be $0.2\mu A$. Since there was no evident difference between C2C and D2D behavior in terms of noise, as discussed in section 3.6, the value will be sampled every time a new *verify phase* - V of the algorithm is simulated.

This is the typical Monte Carlo method approach, and it consists in solving the equation (4.2) hundreds of times with the introduction of random noise and random fluctuations around parameters' median values.

4.6 UNIFIED MODEL DESCRIPTION

Since the goal is to have a model that is agnostic to the applied program and verify algorithm, and thus able to fit every algorithm with a single set of parameters obtaining the same results as the experimental data even for the statistical distribu-

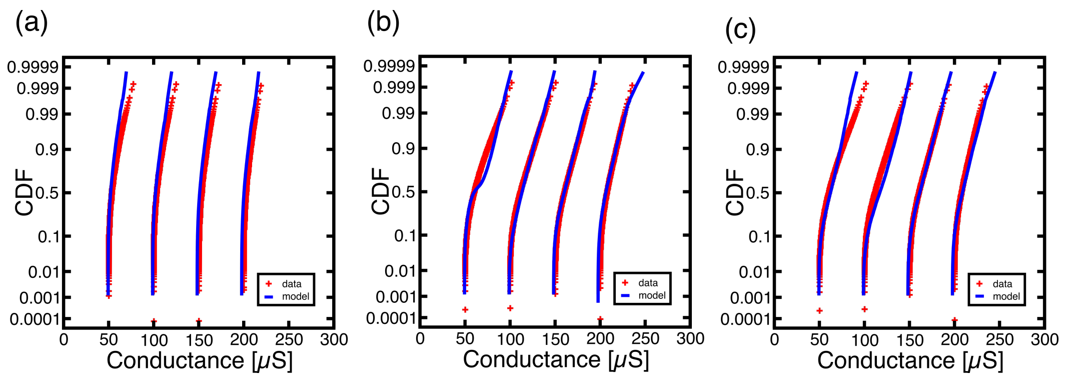

Figure 4.8: Cumulative distributions of the After Switching conductance value for the three simulated algorithms, (a) IGVVA10, (b) IGVVA100 and (C) ISPVA100. The model (in blue) follows with high accuracy the distribution of the experimental data (in red). IGVVA10 results in a steeper curve, meaning that the standard deviation of the final value is lower compared with the other two algorithms.

tions, the model shall be finely tuned in the median value of the parameters and both the C2C and D2D variabilities.

To be able to describe all three algorithms with a single set of parameters and variability values, some fitting compromises should be made. It is an acceptable trade-off considering that experimental data naturally present non-negligible variability even between two adjacent levels or between two different measurement sets. The closest fitting to every algorithm is reached by choosing the parameters as follows: $\alpha = 0.8V^{-1}$, $A = 5 \times 10^8 \frac{\sqrt{S}}{s}$, $E_A = 1.0eV$, and $R_{th} = 1e5\frac{K}{W}$.

The values for the variabilities used in the simulations are the following: $\sigma_{A,d} = 3e7\sqrt{S}/s$, $\sigma_{A,c} = 0.8e7\sqrt{S}/s$, $\sigma_{\alpha,d} = 0.05V^{-1}$, $\sigma_{\alpha,c} = 0.01V^{-1}$, while noise is expressed as $\mu_{\sigma I} = 0.85\mu A$ and $\sigma_{\sigma I} = 0.2\mu A$.

The median values for the current of the four main levels of the three analyzed algorithms are depicted in figure 4.7. As one can notice, the final value is reached with discrete accuracy, and most importantly, as shown in figure 4.8, the distribution of the *After Switching* values of conductance follow accurately the experimental data, confirming once again that IGVVA10 is the most accurate programming scheme.

Comparing the C2C and D2D distributions shown in figure 4.9, it is possible to notice how the model is able to accurately predict the statistical distributions of the

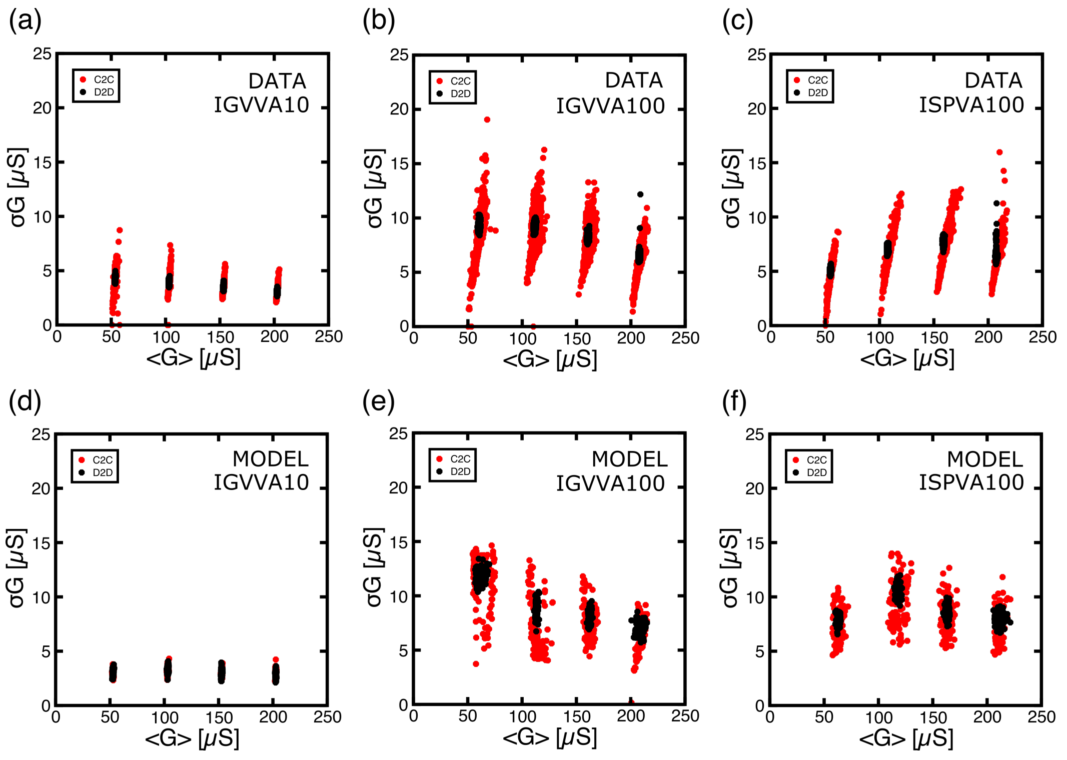

Figure 4.9: C2C vs D2D variability of the experimental data (top row) compared to the developed model (bottom row) expressed as σ_G as a function of the median value μ_G of the after switching conductance value. From left to right the three algorithms are simulated: (a,d) IGVVA10, (b,e) IGVVA100, and (c,f) ISPVA100.

devices, for all three algorithms and all the tested LRS levels. Particular attention must be paid to C2C distribution (represented in red). The elongated shape of the distribution means that by statistically varying the parameters it is possible to recreate the presence of different kinds of devices, namely the cleanest ones, i. e. with a lower variability between cycles, and the *dirtiest* ones, with higher values of standard deviation.

Looking instead at the same results from another point of view, in figure 4.10 the cumulative distribution of V_{MID} is represented, showing how early a device is transitioning from HRS to LRS state. V_{MID}, as described in the previous chapter, is the control voltage (V_G in case of IGVVA), for which the read current reaches half of the LRS value, in this case $I_{read} = 20\mu A$. On the bottom row 4.8c,d the results from the simulations of the IGVVA100 programming scheme are shown. Differently

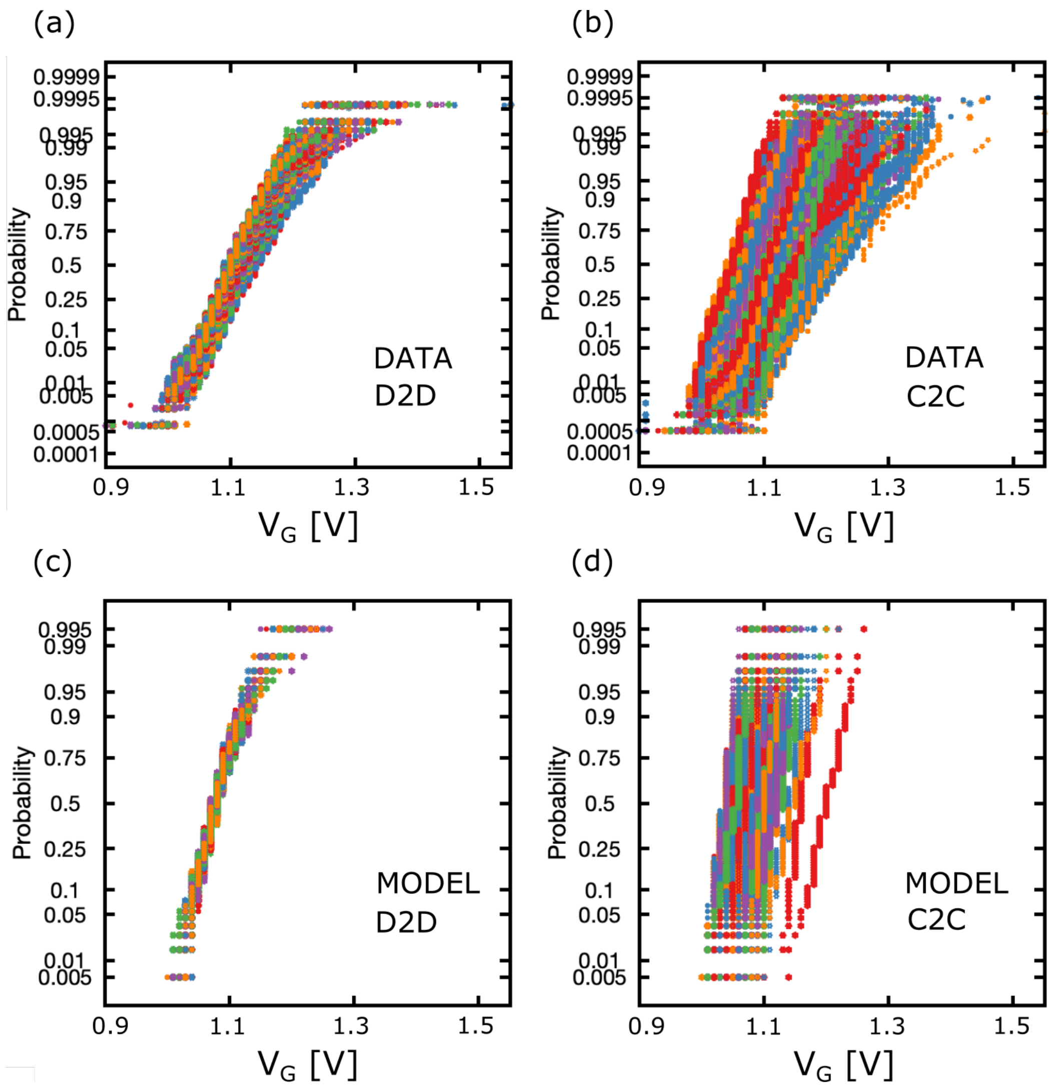

Figure 4.10: Cumulative distribution of V_{MID}, comparison between experimental data (top row) and model simulation (bottom row) for IGVVA100 programming scheme. On the left, each curve represents a cycle, and each dot on the curve represents a device. On the right, instead, the C2C behavior is shown, meaning that each curve represents a device, and the inclination of the curve represents the variability of that device.

from the experimental data, which represents 1000 cycles of 1000 devices, the model simulated only 100 devices, each of them for 100 cycles.

Moreover, the experimental data considers all the cells that reach a correct set value, while our model is targetting particularly the median behavior of the devices. This explains the thinner and more regular distributions of the model analysis compared to the experimental data. Since the vast majority of the experimentally

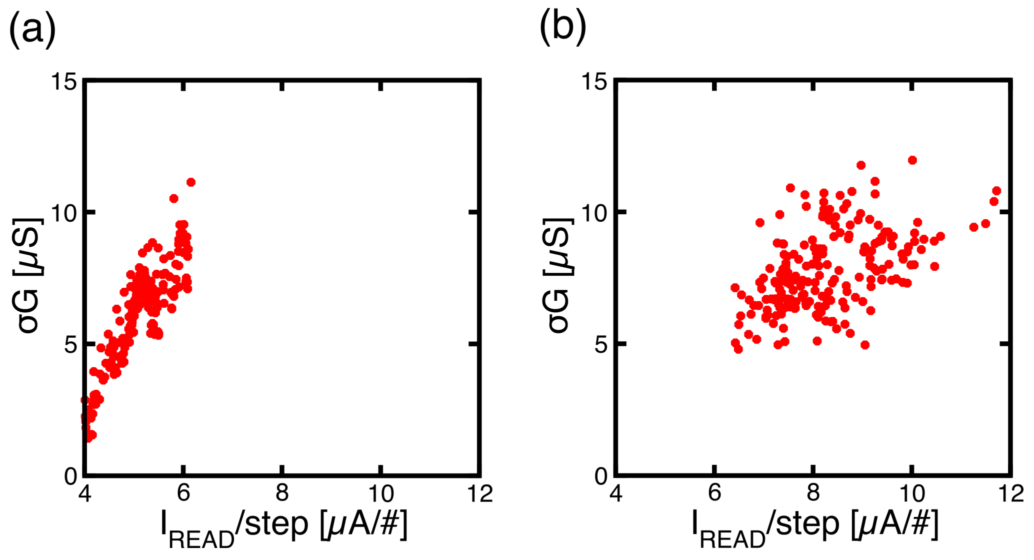

Figure 4.11: Correlation between the median slope of each device (C2C), i. e. how many μA the conductance increases for a single programming step, and the standard deviation of the programming accuracy. Each point represents a device. (a) shows IGVVA100 and (b) ISPVA100 programming schemes simulations, both for LRS4 with target $I_{read} = 40\mu A$.

measured cells follow the median behavior, and only a few of them fall on the two extremes, this analysis demonstrates that the model can predict the statistic behavior of the array with good approximation.

Comparing instead, the slope of the set transition with the reached accuracy in the same way it was done in section 3.5, shows a similar correlation between the chosen figure of merits. The model, as it is expected, can easily simulate the desired behavior intrinsically confirming the relevance of the model accuracy.

4.7 RESET TRANSITION MODELING

Through this work, the reset transition performed with an ISPVA algorithm was also briefly analyzed in section 3.2.3. At this scope, a model similar to the one discussed above is proposed, following the same developing path that starts from the median values and goes to the variability analysis, exactly as done before with the development of the set model.

4.7.1 *Reset model rate equation*

During the set transition, from a geometrical standpoint, the width of conductive filament changes, increasing with every step until the compliance current is reached. Instead, during the reset transition as already discussed in section 2.2, the conductance changes due to a gap that forms in the filament and increases in size.

For this reason, the rate equation of the conductance during the reset transition is modeled through a different equation than the set model.

$$\frac{dG(t)}{dt} = -Ae^{-\frac{E_{A0}-\alpha q V_R(t)}{kT_0\left(1+\frac{R_{th}P(t)}{T_0}\right)}} \tag{4.3}$$

The differences with the set model are the *minus* sign in front of the equation, to express a decrease in the conductance during the process, and the absence of the square root of G in the right term of the equation. This is justified by the different relation that connects the geometric and the conductance variations.

4.7.2 *Reset model algorithm*

The Program and Verify algorithm used to reset the conductive value of the 1T1R cell is a modified version of the ISPVA programming scheme used during the set transition.

In fact, during the P phase, the gate terminal is polarized at a fixed voltage $V_G = 2.7V$, while the voltage across the device is applied with opposite polarity, i. e. keeping V_{TE} at ground and applying a voltage ramp to V_{BE}. Like during ISPVA100 programming for the set transition, the voltage ramp starts at $0.5V$ and stops at $2.0V$ increasing with $100mV$ steps occurring every $1\mu s$. The V phase instead is performed exactly in the same way as for the set algorithms, with $V_G = 1.7V$, $V_{TE} = 0.2V$, and V_{BE} at ground.

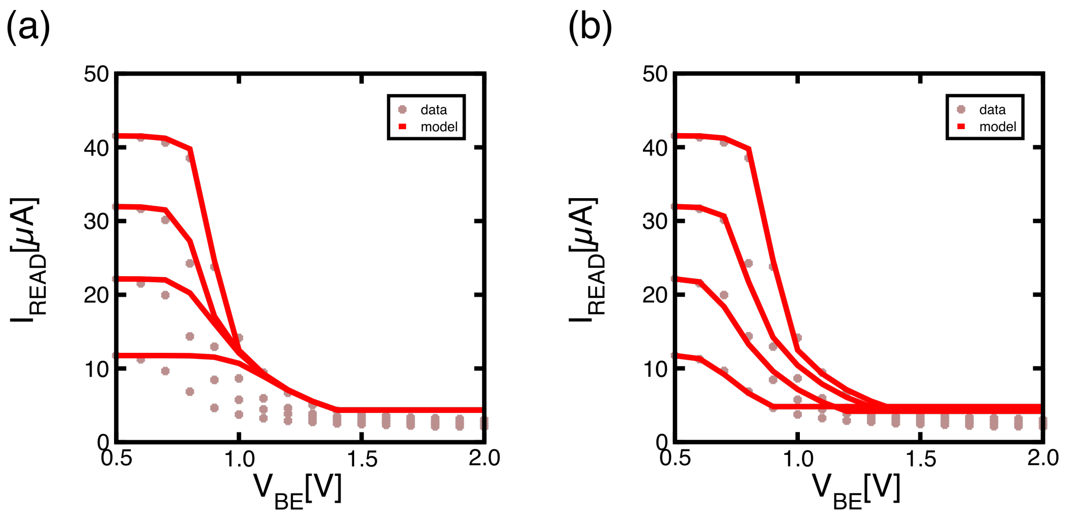

Figure 4.12: Reset transitions for levels LRS 1-4 simulated (a) using the same parameters for all levels and (b) adjusting thermal resistance for each level.

4.7.3 *Reset model fitting*

To fit the median device curves, a distinction has to be made for the different programmed levels. The chip analyzed in chapter 3 consists of 4096 devices divided into four different regions for the four different LRS levels. Each device was initially formed with the compliance current of that particular level, so that the width of the filament changes between LRS1, LRS2, LRS3, and LRS4. On an actual chip, every device will be electroformed with an equal compliance current.

This difference in compliance translates into a difference in the parameters for each level. To simplify, only the thermal resistance R_{th} is directly affected in this model, increasing the value as the filament width decreases.

The parameters chosen for this model are the following: $\alpha = 0.2V^{-1}$, $A = 3 \times 10^6 \frac{S}{s}$, $E_A = 1.0eV$, and the thermal resistance for levels from LRS 1 to LRS 4 respectively is $R_{th} = 20e6, 11e6, 8e6, 7e6 \frac{K}{W}$.

As depicted in figure 4.12b, the median values of the four levels are followed accurately by the model. Instead, by keeping the same parameters' values for all the levels, the result changes slightly, and it can be seen in figure 4.12a.

By adding variability on top of the parameters, in the same way it is done for the set transition model, the result is the following, showing an accurate representation

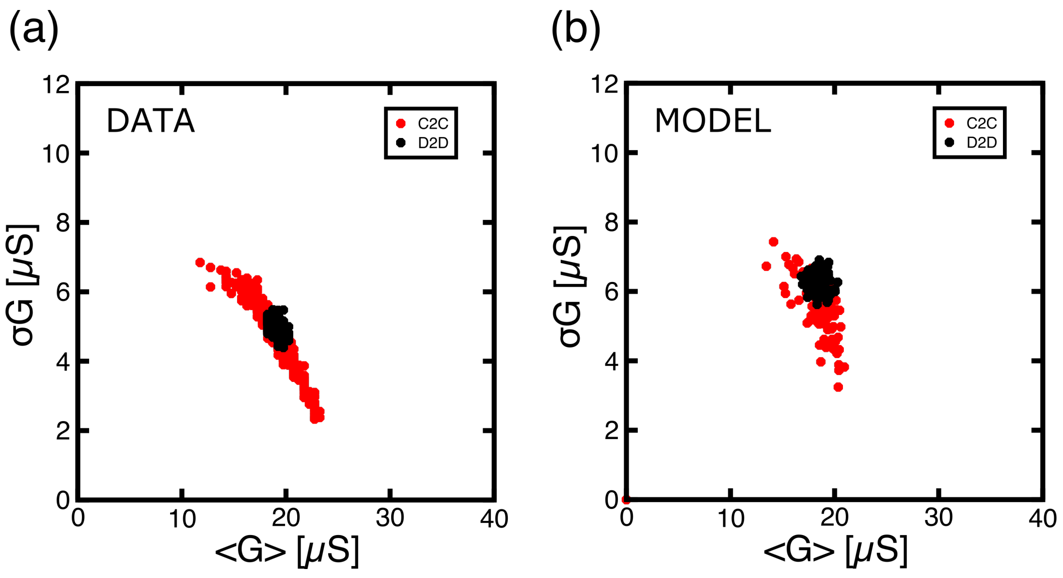

Figure 4.13: C2C (in red) variability and D2D (in black) extracted from experimental data (a) and the model simulation (b).

of the cycle to cycle and device to device variabilities in figure 4.13. This model is able to accurately simulate the variability distribution occurring during the reset transition. Similarly to what has been discussed for the set model, the reset model predicts the behavior of different types of devices, namely the ones with a clean transition featuring a lower AS variability, and in the same way the more abrupt that cause a higher C2C variation of the programmed conductance value.

4.8 SIMULATION CONTAINER FOR NEURAL NETWORK IMPLEMENTATION

Now that the models for both set and reset transitions implementing different program and verify schemes are validated over experimental measurements, it is possible to integrate everything inside a simulation of a hardware neural network implementation for image classification.

In the next chapter a deeper analysis will be performed, comparing results of the neural network's inference using variability values extracted from experimental data in chapter 3 and using a fully simulated network by including the model developed in this chapter.

5

FULLY-CONNECTED NEURAL NETWORK IMPLEMENTATION USING 1T1R HAFNIUM-OXIDE-BASED RRAM ARRAYS

This chapter presents the implementation of a fully connected multilayer neural network with RRAM arrays discussed in the previous chapter. Particular attention will be paid to the impact of the programming variability affecting the hardware device. Simulations will highlight the different trade-offs that have to be considered, demonstrating the need for an accurate statistical model that aims to the optimization of the neural network design in a real-case scenario.

5.1 INTRODUCTION

Recent works [59, 60, 61] have investigated the ability to implement neural networks in hardware by the use of HfO-based RRAM arrays. In this chapter, such an application will be proposed, implementing a multilayer Fully connected Neural Network (FC-NN) for image recognition, using an array of Hafnium Oxide RRAM devices in a 1T1R configuration. As already introduced in chapter 1, the multilevel operation is essential to achieve a high inference accuracy with such neural networks able to reach the performance close to a software implementation of the same network. In fact, by increasing the number of programmable levels, a finer synaptic weight closer to analog programming can be achieved. A universal problem with multilevel programming is the presence of variability and noise that can be detrimental to the accuracy of the final level, making a further increase of the level's number useless. Thus, a common goal is to refine the multilevel operation of such devices, increasing the number of levels and reducing the programming variability. In previous chapters different program and verify schemes are analyzed and in chapter 4 a stochastic model able to predict such programming

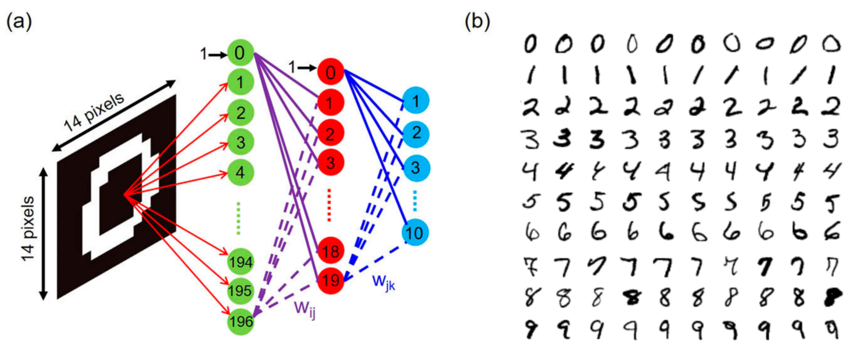

Figure 5.1: (a) Schematic representation of a two-layer fully connected neural network, adapted with permission from [62]; (b) Example of some handwritten digits contained in the MNIST dataset.

variability is developed. A stochastic model is a very powerful tool to predict the variability with different programming schemes, and if implemented into a neural network simulation, can be exploited to optimize the network aiming to increase the inference accuracy.

Through this chapter, different network configurations will be studied with particular attention to the trade-offs that guide the design choices of the neural network in order to increase the inference accuracy, while maintaining low power consumption and compact network size. The programming variability affecting the physical devices will be taken into consideration and the different program and verify techniques will be analyzed and exploited to mitigate the programming variabilities. Some array nonidealities such as IR drop will be also taken into consideration. After that, the statistical model developed in chapter 4 will be integrated into the neural network simulation, in order to compare the achieved results and fully simulate the inference process on a pre-trained network.

5.2 FULLY CONNECTED NEURAL NETWORK

As already briefly introduced in chapter 1, a fully-connected Neural Network can be implemented in a smart way employing emerging memory devices disposed in crossbar structures, taking advantage of the natural predisposition to perform Matrix-Vector Multiplication (MVM) directly in-situ, thus mitigating the

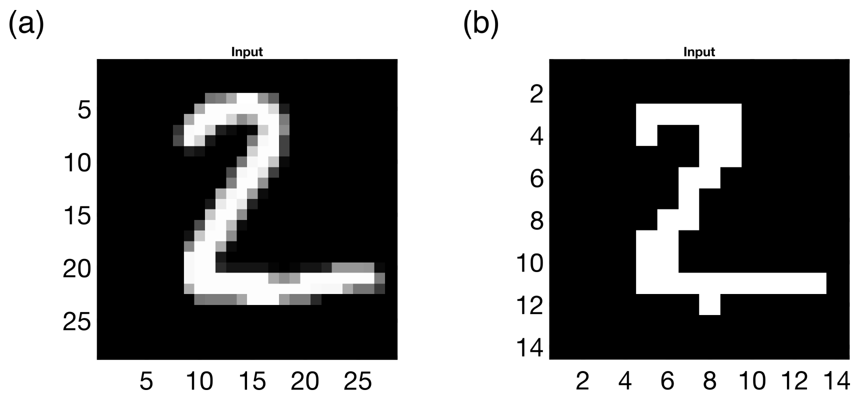

Figure 5.2: (a) Example of a 28x28 pixels greyscale image from MNIST test dataset; (b) The same image but resized to 14x14 pixels and with only black/white color depth.

Von Neumann memory bottleneck by avoiding millions of memory accesses and performing vectorial operations directly.

In this chapter, a particular implementation of a fully connected deep neural network specialized in classifying handwritten digits will be proposed and developed (Figure 5.1A). The implementation consists first in a *training* phase, during which the synaptic weights of the network are trained, i. e. optimized against a high number of example images. The second phase of the implementation, namely the *inference*, consists in classifying other images that the network has never seen before. Both the training and the inference datasets used in this work are from the Modified National Institute of Standards and Technologies database (MNIST) dataset (Figure 5.1b).

Each image of the MNIST dataset is an 8bit grayscale square (28x28 pixels) picture, representing a handwritten digit from 0 to 9 (Figure 5.2a). Each image comes with an attached label representing its value.

The network will be virtually implemented on the $4-kbit$ 1T1R array from *IHP Microelectronics*, already deeply analyzed in chapter 3, however, different configurations will be considered exploiting the combinations of more than one array chips in order to increase the complexity and the size of the network to increase the final accuracy.

Since the array consists of 64x64 cells, it is convenient to reduce the number of the requested input neurons. To do so, the original dataset was simplified, reducing the size of the images to just 14x14 pixels, and converting the greyscale color depth into black and white values, represented by "0" and "1" instead of a floating-point value (Figure 5.2b).

This allows us to reduce the number of the required cells of the input layer to just 196. An additional cell is used as bias. The bias is a constant term and it has the effect of adjusting the output to the activation function [63].

The output layer will be composed of just 10 neurons, each one for each output class.

In this work, only two-layer neural networks are explored (Figure 5.1a), thus containing only one hidden layer of neurons between the input and the output layers. The number of hidden neurons was chosen accordingly to study the influence of the network size on the final classification accuracy.

The simplest network as described above can be implemented on a 64x64 array by concatenating 4 different arrays as shown in figure 5.3. The three arrays on the left act as the first synaptic layer that connects the input of the network to the input of the second layer. The input signals, entering as voltages from the left of the network are propagated into the second layer (right array) after the MVM are performed between the signals and the conductance's synaptic weights. The output of the first layer, given as a current, passes through a nonlinear function such as a sigmoid, here entirely simulated via software, and enters the second synaptic layer that after an equivalent process generates the output of the system, firing the neuron that represents the output class, i. e. the label representing the number given to the network as the input image.

5.3 INCREMENTAL NETWORK QUANTIZATION ALGORITHM

As shown in chapter 3, the total number of separate conductive levels that can be successfully programmed on the 1T1R array chip in our study consists of 8 LRS levels and one HRS level (Figure 5.4c).

The experimental data show that the HRS value, despite being targeted to $25\mu S$, suffers a sort of relaxation and the median value of the final conductance decreases

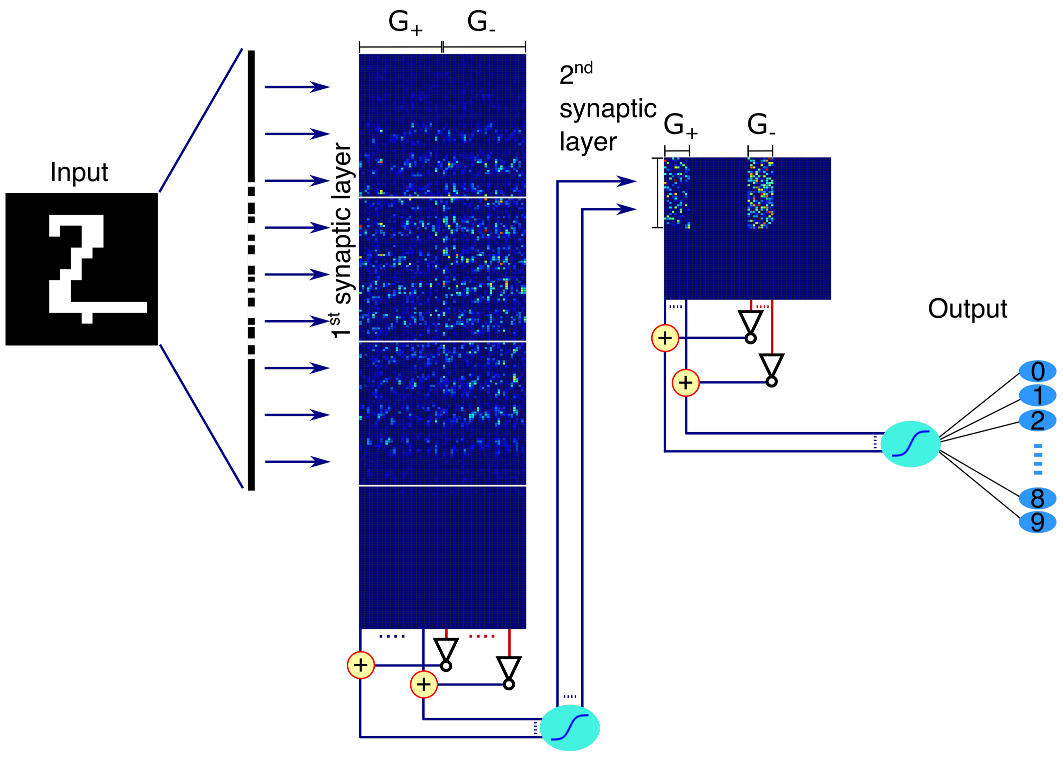

Figure 5.3: Implementation of a two-layer fully connected neural network on real-size crossbar arrays, consisting of 64x64 matrices of 1T1R devices analyzed during this work. To fully implement the network, four arrays are needed for the first synaptic layer, and one array for the second layer.

down to about $8\mu S$ just after few read cycles. This gave the space to virtually add another LRS level that can be programmed targetting $25\mu S$. This solution increases the total number of levels to nine LRS and one HRS (Figure 5.4c).

On the other hand, a software implementation of a neural network typically employs a virtually continuous distribution of the synaptic weight values, by representing each weight with a precision of 32 or even 64bit in a floating-point format. This is one of the main differences between software and hardware implementations of such networks. Reducing the precision of the synaptic weights decreases the overall accuracy of the network, but this effect can be traded with the overall speed, size, and power consumption of the network. Finally, by increasing the size of the network it is possible to increase the overall accuracy compared to smaller software implementation, while still having a smaller power impact.

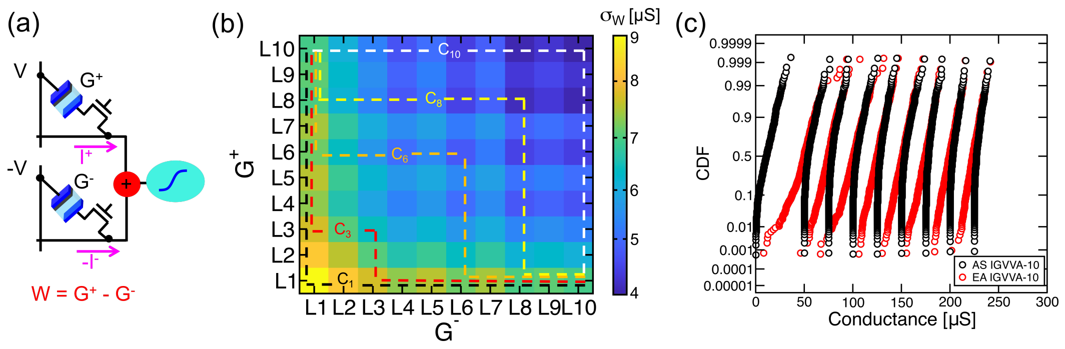

Figure 5.4: (a) Schematic representation of the differential configuration of conductance values used in this work; (b) Example of different weight combinations that can be selected to form the 19 differential levels. The colors in the color map represent the standard deviation of the single levels using the IGVVA10 programming scheme; (c) experimentally extracted 8 LRS levels and one HRS level using the IGVVA10 programming scheme. The figure highlights the difference between the After Switching (AS) value and the End of the Algorithm (EA). Adapted with permission from [55]. Copyright 2021, IEEE.

Another way to increase the accuracy of a neural network while using a discrete number of levels is to act on the quantization method. Instead of simply rounding all the trained weights to their closest discrete value, an iterative incremental quantization algorithm can be applied to the network.

For this purpose, an Incremental Network Quantization (INQ) algorithm was developed by Zhou et al. [64] and it is implemented in this work.

The INQ algorithm is applied during the training of the network and consists of several steps, as depicted in figure 5.5.

- First, the network is trained with floating-point values.
- Then, 50% of the weights is randomly chosen and get quantized, i. e. rounded to the next closest value.
- The network is then re-trained, but keeping the previously quantized weights fixed and updating only the remaining ones, that are free to adapt to the modified situation.
- Another 50% of the remaining floating point weights is quantized and fixed.

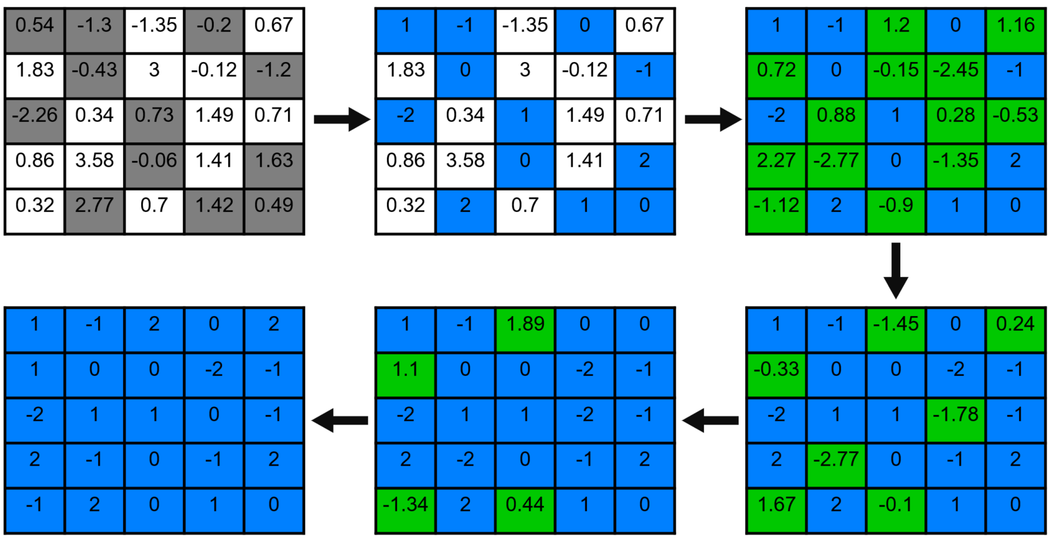

Figure 5.5: Schematic representation of the Incremental Network Quantization algorithm on a 5x5 matrix. (Top row) Half of the floating-point weights in a pre-trained network are randomly selected and (center) quantized. (right) The remaining floating point weights of the network are then retrained. The bottom row shows the next iterations of the algorithm until all the weights are quantized (bottom left).

- The procedure is repeated until all the cells are quantized.

In this work, the chosen percentages of INQ are: 50%, 75%, 87.5%, 100%. The number of finite quantized levels is chosen accordingly to the used device. In the analyzed case study, each cell can be programmed with up to 9 different LRS levels and one HRS level, but since the cells are used in a differential configuration to exploit both positive and negative levels (Figure 5.4a), a total number of 19 discrete levels can be programmed.

5.4 NEURAL NETWORK ACCURACY

A neural network with 196 input neurons, 20 hidden neurons, and 10 output neurons, trained with a backpropagation algorithm employing floating-point values for the synaptic weights, can reach a class accuracy of 93.27% during the inference when supplied with the test dataset from the MNIST database. By simply quantizing the weights to their closest discrete value, the accuracy drops down to

	20h	50h	100h
Training FP64	93.27%	95.78%	96.77%
INQ 19L	92.16%	95.41%	96.67%
INQ 17L	91.84%	95.33%	96.58%
INQ 15L	91.56%	95.29%	96.40%
INQ 13L	90.97%	94.98%	96.45%
INQ 11L	89.69%	94.64%	96.44%
INQ 9L	88.72%	94.05%	96.00%
INQ 7L	85.57%	92.86%	95.20%
INQ 5L	70.27%	89.92%	93.14%

Table 5.1: Accuracy of the real network inference after quantizing the synaptic weights to a different number of levels. The first row indicates the size of the hidden layer.

82.7%, and by applying the INQ training algorithm discussed in section 5.3, the accuracy increases up back to 92.16%. This means that additional 1000 images of the 10000 images test dataset are correctly classified in comparison with a network that has a simple rounding quantization.

Increasing the total number of neural synapses by increasing the size of the hidden layer from 20 to 50 neurons, can increase the inference accuracy of a floating-point network up to 95.78%, and if the number of hidden neurons is brought up to 100, the maximum class accuracy of the inference reached is 96.77%. These results are summarized in table 5.1.

In figure 5.6, it can be noticed how the decrease of quantization levels affects the ability of the network to correctly classify the images. This explains the need for studying and enhancing multilevel programming schemes.

By further decreasing the number of levels, for example, to 7 quantized levels, it is possible to have a distance ΔW between two consecutive weights of $\Delta W = 25\mu S$, or to use more distant levels and have a $\Delta W = 50\mu S$ or even $\Delta W = 75\mu S$.

One interesting result that is highlighted in figure 5.6b, is that when ΔW increases also the accuracy of the network increases.

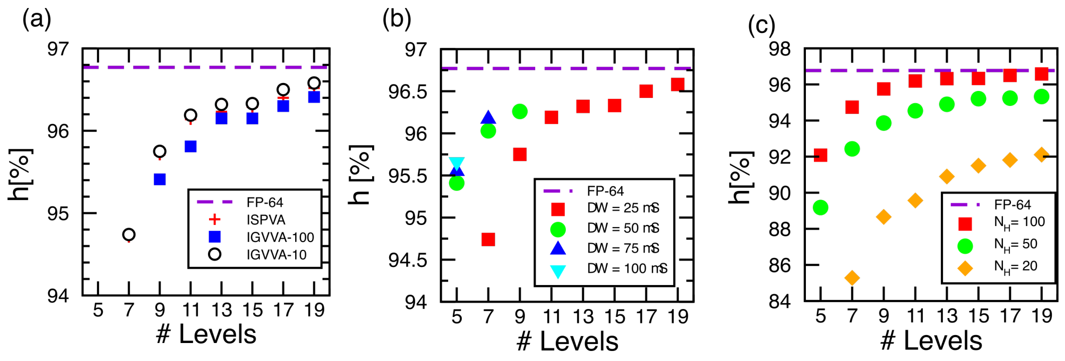

Figure 5.6: Calculated inference accuracy of the 2-layer FC-NN with $N_H = 100$ as a function of the number of synaptic weight levels (a) by IGVVA-100, ISPVA, and IGVVA-10 CDFs. The higher number of levels combined with IGVVA-10 programming leads to an accuracy η close to FP-64. for (b) various steps ΔW in weight mapping, and (c) increasing size of hidden layer N_H. Adapted with permission from [55]. Copyright 2021, IEEE.

5.5 VARIABILITY, NONIDEALITIES, AND IMPACT ON INFERENCE ACCURACY

Having the ability to simulate the training and the inference of a neural network allows us to understand many of the trade-offs weighting on the design choices. Unfortunately, when the network is transplanted into a real hardware device, the presence of electronic noise, programming variability, and other types of nonidealities can influence the performance and the accuracy of the system in a catastrophic way. For this reason, it is essential to be able to consider all these kinds of nonidealities at a simulation level.

We know from chapter 3 and chapter 2 that the conductance values programmed into the RRAM array present some strong variabilities. In this specific application, it means that every synaptic weight will not be precisely the discrete value given by the training optimal for the network, but it will likely suffer from variability and will have a Gaussian shape distribution, as shown in figure 5.7. Another nonideality that affects the neural network when implemented on a real crossbar array, is the presence of some stuck cells that can be both at HRS or LRS positions. Unfortunately, as was already discussed by looking at the experimental data in chapter 3, the number of stuck cells inside a $4 - kbit$ array can be in the order of

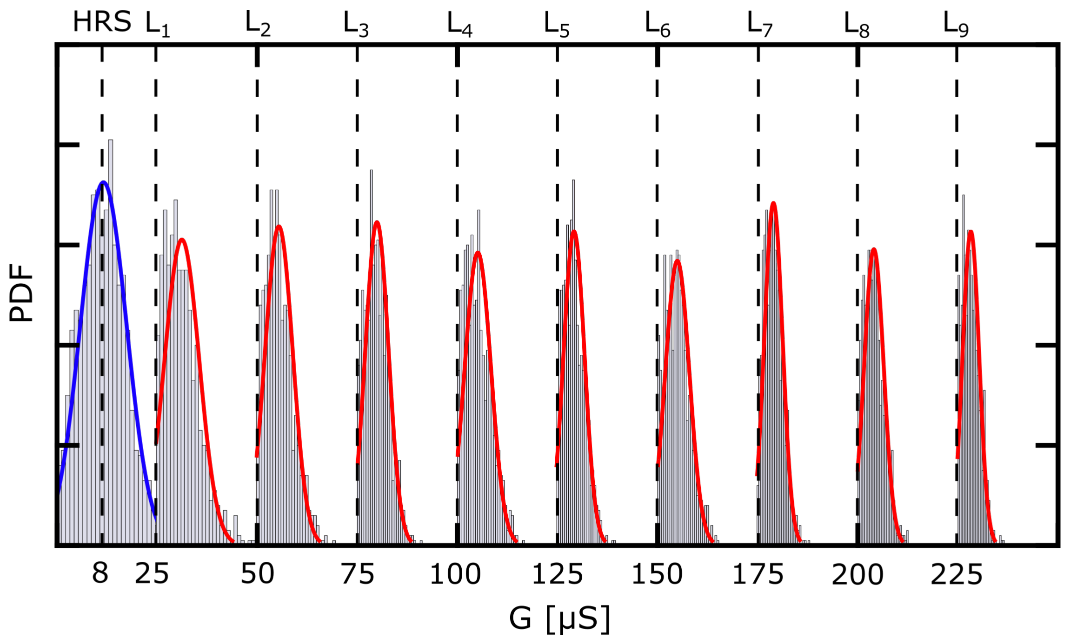

Figure 5.7: PDF of the after switching conductance G_{CELL} for the programmed levels through IGVVA10 programming algorithm.

20-30%. If the network is trained without considering these non-working cells, the accuracy of the inference performed on a simulated crossbar array and considering a sparse distribution of these stuck cells will suffer an inevitable reduction.

Fortunately, since the position of the stuck cells is known a priori, it is possible to exploit this knowledge and train the network considering the presence of some non-working cells. The network will adapt and this will have a positive impact on the final value of inference accuracy.

Another detrimental effect on the accuracy of the network is the Ohmic drop (IR drop) caused by the wire resistances of the array [55]. It depends on different contributions, namely on the ratio between the cell resistance R_{LRS} and the wire resistance r_{wire}, on the size of the array, and the voltages applied to the network during the inference phase.

5.5.1 *Program and verify algorithms variability*

The program and verify algorithms help to decrease the programming variability of the set transition in the RRAM cells. In particular, this work focuses on two

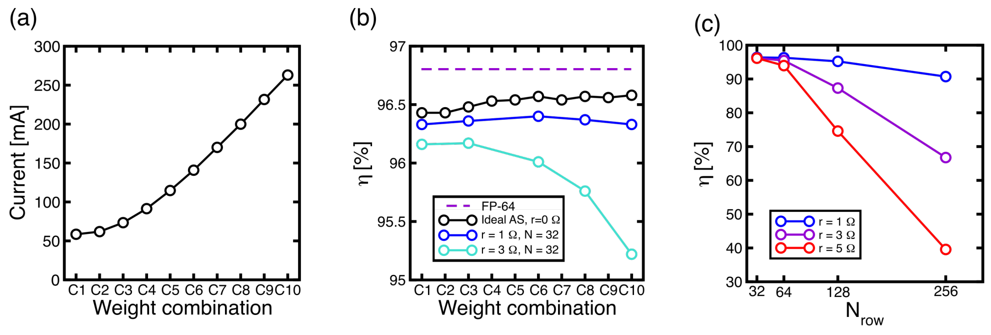

Figure 5.8: (a) Current consumption as a function of the weight mapping combination; (b) Impact of the IR drop on the inference accuracy of the calculated FC-NN based on crossbar arrays of 32x32 RRAM, as a function of the weight combination for increasing line resistance r; (c) Impact of the IR drop on the calculated inference accuracy as a function of the size of the crossbar array for increasing wire resistance r. Adapted with permission from [55]. Copyright 2021, IEEE.

different programming algorithms, already introduced and discussed in previous chapters.

ISPVA is performed by choosing a particular polarization for the NMOS transistor, and by applying a pulsed voltage ramp to the top electrode of the cell. Each programming pulse is interleaved with a read pulse with a low top electrode voltage, and this way by measuring the current it is possible to extract the conductance value. At a certain $V_{TE} = V_{set}$ the RRAM switches from a High Resistive State (HRS) to the Low Resistive State (LRS) defined by the compliance.

IGVVA, on the other hand, is performed by applying a pulsed voltage ramp on the gate of the transistor, while keeping the top electrode at a voltage higher than V_{set}. This way, the cell increments its conductance at every pulse and it is limited gradually by the increasing compliance.

The step of the voltage stair in ISPVA was $100mV$, while the IGVVA scheme was stepped with both $100mV$ and $10mV$. There is a direct proportionality between the precision of the programmed levels and the resulting neural network inference accuracy (Figure 5.6a).

5.5.2 *Current consumption*

From figure 5.4b, it is evident how the 19 discrete levels can be mapped in different ways. The same weight W can be obtained by the subtraction of different G^+ and G^- conductances. From a variability point of view, it is easy to say that for the IGVVA10 algorithm the best choice would be *combination C10*, i. e. the combination of G^+ and G^- for which one of the two conductances always assumes the LRS9 value, which provides the cleanest distribution.

From a current consumption point of view, on the other hand, the best solution would be to use *combination C1* in which one of the two conductances is always in the HRS position, thus has the lowest overall current flowing. This last combination of course has a detrimental effect on the network accuracy, since the total variability of the HRS is higher than the variability of the other levels, and this accuracy loss can be evaluated in figure 5.8b. As one can see, by using a precise algorithm such as IGVVA10, even if the lower conductive levels present some more variability compared to the higher levels, the loss in terms of network accuracy is limited to a range of 0.2%, and the current consumption, with the addition of dissipated power, can create an interesting trade-off between the two figure of merits (Figure 5.8a).

This trade-off is further complicated if the IR drop is considered. Since the current amplitude directly impacts the ohmic drop along the wires, the advantages of using lower conductance values win over the higher level accuracy of cleaner combinations.

5.5.3 *Size of the network*

The size of the network, i. e. the number of synaptic neurons directly impacts the accuracy that can be reached during the inference. In fact, increasing the number of neurons causes an increase also in the number of synaptic connections. In this work, several configurations were analyzed in order to better understand how the size impacts the accuracy. The input layer and the output layer have a fixed number of neurons because each neuron of the input corresponds to a pixel of the image (14x14 pixels), and each output neuron corresponds to an output class (0 to 9). The topology of the network is a fully connected neural network with only one layer of hidden neurons. The number of neurons in the hidden layer can vary, and in this work networks with 20, 50, and 100 hidden neurons were analyzed. The impact in terms of the inference accuracy is important, increasing by increasing the size, as can be seen in figure 5.6 and table 5.1.

Since the size of the input layer is higher (14 x 14 + 1 = 197 neurons) than any of the sides of the array under test (64 x 64), at least four of them must be employed to create the first layer of synaptic weights that connect the input to the second layer. An example of the design is shown in figure 5.3.

6 CONCLUSION

In this thesis work, a new statistical model of a 4kbit HfAlO RRAM array was developed, able to predict the programming variability of set and reset transitions during the different program and verify algorithms.

First, the physical characteristics of the RRAM device responsible for the conductance increase and the variability were analyzed. The switching mechanism in OxRAM memories is due to formation and disruption for set and reset transitions respectively, of a conductive filament made of oxygen vacancies that during an applied voltage to the top and the bottom electrodes, migrate increasing or decreasing the conductance of the RRAM cell. Since the number of these vacancies is discrete, the resistive switching is strongly impacted by stochastic fluctuations, which results in the conductance variability of the programmed value.

To reduce such variability and to exploit the multilevel-cell programming, some program and verify algorithms were analyzed and compared, measuring their performances in terms of programming accuracy for several Low Resistive State (LRS) levels.

After that, a statistical model for the RRAM cell was developed starting from the geometrical variations of the conductive filament during the programming transitions, and including the I-V characteristic of the NMOS device in series with the resistive cell that together form a one Transistor - one Resistor (1T1R) structure.

The model was tuned to fit the medians of the transitions measured experimentally for each of the previously described algorithms, achieving up to 8 different LRSs. Then, the model was enhanced to include stochastic variability on top of some parameters and tuned to statistically fit the experimentally measured distributions.

As the last step, implementation of a Fully connected Neural Network (FC-NN) was studied addressing several trade-offs such as the network size and the accuracy of the programmed synaptic weights. The model was included inside the neural network simulation, predicting the inference accuracy drop due to programming variability.

To conclude, the statistical model developed during this thesis is a useful tool to predict the programming variability of an RRAM array, enabling the simulation of large-scale neural network implementations and helping the study of new programming algorithms aiming at further increasing the accuracy.

BIBLIOGRAPHY

[1] G.E. Moore. “Cramming more components onto integrated circuits.” In: Electronics, vol.38 (1965), pp. 114–117 (cit. on p. 1).

[2] 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.” In: *IEEE Journal of Solid-State Circuits* 9.5 (1974), pp. 256–268. DOI: 10.1109/JSSC.1974.1050511 (cit. on p. 2).

[3] Mark T. Bohr and Ian A. Young. “CMOS Scaling Trends and Beyond.” In: *IEEE Micro* 37.6 (2017), pp. 20–29. DOI: 10.1109/MM.2017.4241347 (cit. on p. 2).

[4] Sayeef Salahuddin, Kai Ni, and Suman Datta. “The era of hyper-scaling in electronics.” In: *Nature Electronics* 1.8 (2018), pp. 442–450. DOI: 10.1038/s41928-018-0117-x (cit. on pp. 2, 7, 13).

[5] M. Horowitz. “1.1 Computing’s energy problem (and what we can do about it).” In: (2014), pp. 10–14. DOI: 10.1109/ISSCC.2014.6757323 (cit. on p. 2).

[6] M. M. Waldrop. “The chips are down for Moore’s law.” In: Nature, vol. 530.7589 (2016), pp. 144–147 (cit. on p. 2).

[7] K. Kupp. *Microprocessor Trend Data [Online].* URL: https://github.com/karlrupp/microprocessor-trend-data. (accessed: 12.06.2021) (cit. on p. 3).

[8] Wm. A. Wulf and Sally A. McKee. “Hitting the Memory Wall: Implications of the Obvious.” In: *SIGARCH Comput. Archit. News* 23.1 (Mar. 1995), 20–24. ISSN: 0163-5964. DOI: 10.1145/216585.216588 (cit. on p. 2).

[9] J. von Neumann. “First draft of a report on the EDVAC.” In: *IEEE Annals of the History of Computing* 15.4 (1993), pp. 27–75. DOI: 10.1109/85.238389 (cit. on p. 2).

[10] H. S. Philip Wong and Sayeef Salahuddin. “Memory leads the way to better computing.” In: *Nature Nanotechnology* 10.3 (2015), pp. 191–194. DOI: 10.1038/nnano.2015.29 (cit. on p. 3).

[11] Peter J. Denning and Stuart C. Schwartz. “Properties of the Working-Set Model.” In: 15.3 (1972). ISSN: 0001-0782. DOI: 10.1145/361268.361281 (cit. on p. 3).

[12] Intel. *Intel Optane Technology [Online].* URL: https://www.intel.com/content/www/us/en/architecture-and-technology/intel-optane-technology.html. (accessed: 12.06.2021) (cit. on p. 4).

[13] Shimeng Yu and Pai-Yu Chen. “Emerging Memory Technologies: Recent Trends and Prospects.” In: *IEEE Solid-State Circuits Magazine* 8.2 (2016), pp. 43–56. DOI: 10.1109/MSSC.2016.2546199 (cit. on p. 6).

[14] Daniele Ielmini. “Resistive-Switching Memory.” In: (2014), pp. 1–32. DOI: https://doi.org/10.1002/047134608X.W8222. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/047134608X.W8222 (cit. on pp. 7, 8, 27).

[15] Daniele Ielmini. “Resistive switching memories based on metal oxides: mechanisms, reliability and scaling.” In: *Semiconductor Science and Technology* 31.6 (May 2016), p. 063002. DOI: 10.1088/0268-1242/31/6/063002 (cit. on pp. 7, 25–27).

[16] Daniele Ielmini, Rainer Bruchhaus, and Rainer Waser. “Thermochemical resistive switching: materials, mechanisms, and scaling projections.” In: *Phase Transitions* 84.7 (2011), pp. 570–602. DOI: 10.1080/01411594.2011.561478. eprint: https://doi.org/10.1080/01411594.2011.561478 (cit. on p. 7).

[17] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. “Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups.” In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97. DOI: 10.1109/MSP.2012.2205597 (cit. on p. 7).

[18] Stefano Larentis, Federico Nardi, Simone Balatti, David C. Gilmer, and Daniele Ielmini. "Resistive Switching by Voltage-Driven Ion Migration in Bipolar RRAM—Part II: Modeling." In: *IEEE Transactions on Electron Devices* 59.9 (2012), pp. 2468–2475. DOI: 10.1109/TED.2012.2202320 (cit. on pp. 7, 29).

[19] Attilio Belmonte, Woosik Kim, (BT) Boon Chan, Nancy Heylen, Andrea Fantini, Michel Houssa, M. Jurczak, and L. Goux. "A thermally stable and high-performance 90-nm Al2O3 Cu-based 1T1R CBRAM cell." In: *IEEE Transactions on Electron Devices* 60 (Sept. 2013), pp. 3690–3695. DOI: 10.1109/TED.2013.2282000 (cit. on p. 8).

[20] H. Y. Lee, P. S. Chen, T. Y. Wu, Y. S. Chen, C. C. Wang, P. J. Tzeng, C. H. Lin, F. Chen, C. H. Lien, and M.-J. Tsai. "Low power and high speed bipolar switching with a thin reactive Ti buffer layer in robust HfO2 based RRAM." In: (2008), pp. 1–4. DOI: 10.1109/IEDM.2008.4796677 (cit. on p. 8).

[21] Stefano Ambrogio, Simone Balatti, Antonio Cubeta, Alessandro Calderoni, Nirmal Ramaswamy, and Daniele Ielmini. "Statistical Fluctuations in HfOx Resistive-Switching Memory: Part II—Random Telegraph Noise." In: *IEEE Transactions on Electron Devices* 61.8 (2014), pp. 2920–2927. DOI: 10.1109/TED.2014.2330202 (cit. on p. 8).

[22] H.-S. Philip Wong, Simone Raoux, Sangbum Kim, Jiale Liang, John Reifenberg, Bipin Rajendran, Mehdi Asheghi, and Kenneth Goodson. "Phase Change Memory." In: *Proceedings of the IEEE* 98 (Dec. 2010). DOI: 10.1109/JPROC.2010.2070050 (cit. on p. 9).

[23] Geoffrey W. Burr, Matthew J. Brightsky, Abu Sebastian, Huai-Yu Cheng, Jau-Yi Wu, Sangbum Kim, Norma E. Sosa, Nikolaos Papandreou, Hsiang-Lan Lung, Haralampos Pozidis, Evangelos Eleftheriou, and Chung H. Lam. "Recent Progress in Phase-Change Memory Technology." In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 6.2 (2016), pp. 146–162. DOI: 10.1109/JETCAS.2016.2547718 (cit. on p. 10).

[24] D. Ielmini, S. Lavizzari, D. Sharma, and A. L. Lacaita. "Physical interpretation, modeling and impact on phase change memory (PCM) reliability of resistance drift due to chalcogenide structural relaxation." In: (2007), pp. 939–942. DOI: 10.1109/IEDM.2007.4419107 (cit. on p. 10).

[25] Daniele Ielmini and Stefano Ambrogio. "Emerging neuromorphic devices." In: *Nanotechnology* 31.9 (Dec. 2019), p. 092001. DOI: 10.1088/1361-6528/ab554b (cit. on pp. 10, 18–20, 22, 23).

[26] T. S. Böscke, J. Müller, D. Bräuhaus, U. Schröder, and U. Böttger. "Ferroelectricity in hafnium oxide thin films." In: *Applied Physics Letters* 99.10 (2011), p. 102903. DOI: 10.1063/1.3634052. eprint: https://doi.org/10.1063/1.3634052 (cit. on p. 11).

[27] T. Mikolajick, C. Dehm, W. Hartner, I. Kasko, M.J. Kastner, N. Nagel, M. Moert, and C. Mazure. "FeRAM technology for high density applications." In: *Microelectronics Reliability* 41.7 (2001), pp. 947–950. ISSN: 0026-2714. DOI: https://doi.org/10.1016/S0026-2714(01)00049-X (cit. on p. 11).

[28] Yoshihiro Arimoto and Hiroshi Ishiwara. "Current Status of Ferroelectric Random-Access Memory." In: *MRS Bulletin* 29.11 (2004), 823–828. DOI: 10.1557/mrs2004.235 (cit. on p. 11).

[29] M. Trentzsch, S. Flachowsky, R. Richter, J. Paul, B. Reimer, D. Utess, S. Jansen, H. Mulaosmanovic, S. Müller, S. Slesazeck, J. Ocker, M. Noack, J. Müller, P. Polakowski, J. Schreiter, S. Beyer, T. Mikolajick, and B. Rice. "A 28nm HKMG super low power embedded NVM technology based on ferroelectric FETs." In: (2016), pp. 11.5.1–11.5.4. DOI: 10.1109/IEDM.2016.7838397 (cit. on p. 11).

[30] Daniele Ielmini and H. S. Philip Wong. "In-memory computing with resistive switching devices." In: *Nature Electronics* 1.6 (2018), pp. 333–343. DOI: 10.1038/s41928-018-0092-2 (cit. on p. 11).

[31] T. Kawahara, K. Ito, R. Takemura, and H. Ohno. "Spin-transfer torque RAM technology: Review and prospect." In: *Microelectronics Reliability* 52.4 (2012).

Advances in non-volatile memory technology, pp. 613–627. ISSN: 0026-2714. DOI: https://doi.org/10.1016/j.microrel.2011.09.028 (cit. on p. 12).

[32] Roberto Carboni, Elena Vernocchi, Manzar Siddik, Jon Harms, Andy Lyle, Gurtej Sandhu, and Daniele Ielmini. "A Physics-Based Compact Model of Stochastic Switching in Spin-Transfer Torque Magnetic Memory." In: *IEEE Transactions on Electron Devices* 66.10 (2019), pp. 4176–4182. DOI: 10.1109/TED.2019.2933315 (cit. on p. 12).

[33] Andrew D. Kent and Daniel C. Worledge. "A new spin on magnetic memories." In: *Nature Nanotechnology* 10.3 (2015), pp. 187–191. DOI: 10.1038/nnano.2015.24 (cit. on p. 13).

[34] Geoffrey W. Burr, Rohit S. Shenoy, Kumar Virwani, Pritish Narayanan, Alvaro Padilla, Bülent Kurdi, and Hyunsang Hwang. "Access devices for 3D crosspoint memory." In: *Journal of Vacuum Science & Technology B* 32.4 (2014), p. 040802. DOI: 10.1116/1.4889999. eprint: https://doi.org/10.1116/1.4889999 (cit. on pp. 13, 15, 24).

[35] Qiangfei Xia and J. Joshua Yang. "Memristive crossbar arrays for brain-inspired computing." In: *Nature Materials* 18.4 (2019), pp. 309–323. DOI: 10.1038/s41563-019-0291-x (cit. on pp. 13, 14).

[36] Daniele Ielmini and Giacomo Pedretti. "Device and Circuit Architectures for In-Memory Computing." In: *Advanced Intelligent Systems* 2.7 (2020), p. 2000040. DOI: https://doi.org/10.1002/aisy.202000040. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/aisy.202000040 (cit. on pp. 14, 16, 23).

[37] Zhong Sun, Giacomo Pedretti, Elia Ambrosi, Alessandro Bricalli, Wei Wang, and D. Ielmini. "Solving matrix equations in one step with cross-point resistive arrays." In: *Proceedings of the National Academy of Sciences* 116 (Feb. 2019). DOI: 10.1073/pnas.1815682116 (cit. on p. 16).

[38] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." In: *Nature* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539 (cit. on p. 17).

[39] Duygu Kuzum, Shimeng Yu, and H-S Philip Wong. “Synaptic electronics: materials, devices and applications.” In: *Nanotechnology* 24.38 (2013), p. 382001. DOI: 10.1088/0957-4484/24/38/382001 (cit. on p. 17).

[40] Wolfgang Maass. “Networks of spiking neurons: The third generation of neural network models.” In: *Neural Networks* 10.9 (1997), pp. 1659–1671. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(97)00011-7 (cit. on p. 18).

[41] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel Emer. *Efficient Processing of Deep Neural Networks: A Tutorial and Survey*. 2017. arXiv: 1703.09039 [cs.CV] (cit. on p. 18).

[42] Guo-qiang Bi and Mu-ming Poo. “Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type.” In: *Journal of Neuroscience* 18.24 (1998), pp. 10464–10472. DOI: 10.1523/JNEUROSCI.18-24-10464.1998. eprint: https://www.jneurosci.org/content/18/24/10464.full.pdf (cit. on p. 18).

[43] Per Jesper Sjöström, Gina G Turrigiano, and Sacha B Nelson. “Rate, Timing, and Cooperativity Jointly Determine Cortical Synaptic Plasticity.” In: *Neuron* 32.6 (2001), pp. 1149–1164. ISSN: 0896-6273. DOI: https://doi.org/10.1016/S0896-6273(01)00542-6 (cit. on p. 18).

[44] Hsinyu Tsai, Stefano Ambrogio, Pritish Narayanan, Robert M Shelby, and Geoffrey W Burr. “Recent progress in analog memory-based accelerators for deep learning.” In: *Journal of Physics D: Applied Physics* 51.28 (2018), p. 283001. DOI: 10.1088/1361-6463/aac8a5 (cit. on p. 19).

[45] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. “Gradient-based learning applied to document recognition.” In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791 (cit. on p. 19).

[46] Eduardo Pérez, Cristian Zambelli, Mamathamba Kalishettyhalli Mahadevaiah, Piero Olivo, and Christian Wenger. “Toward Reliable Multi-Level Operation in RRAM Arrays: Improving Post-Algorithm Stability and As-

sessing Endurance/Data Retention." In: *IEEE Journal of the Electron Devices Society* 7 (2019), pp. 740–747. DOI: 10.1109/JEDS.2019.2931769 (cit. on p. 24).

[47] Fabien Alibart, Ligang Gao, Brian D Hoskins, and Dmitri B Strukov. "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm." In: *Nanotechnology* 23.7 (2012), p. 075201. DOI: 10.1088/0957-4484/23/7/075201 (cit. on p. 24).

[48] S. Balatti, S. Larentis, D. C. Gilmer, and D. Ielmini. "Multiple Memory States in Resistive Switching Devices Through Controlled Size and Orientation of the Conductive Filament." In: *Advanced Materials* 25.10 (2013), pp. 1474–1478. DOI: https://doi.org/10.1002/adma.201204097. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/adma.201204097 (cit. on p. 27).

[49] Daniele Ielmini. "Modeling the Universal Set/Reset Characteristics of Bipolar RRAM by Field- and Temperature-Driven Filament Growth." In: *IEEE Transactions on Electron Devices* 58.12 (2011), pp. 4309–4317. DOI: 10.1109/TED.2011.2167513 (cit. on pp. 28, 33).

[50] Stefano Ambrogio, Simone Balatti, David C. Gilmer, and Daniele Ielmini. "Analytical Modeling of Oxide-Based Bipolar Resistive Memories and Complementary Resistive Switches." In: *IEEE Transactions on Electron Devices* 61.7 (2014), pp. 2378–2386. DOI: 10.1109/TED.2014.2325531 (cit. on pp. 28, 30, 33).

[51] A. Fantini, L. Goux, R. Degraeve, D.J. Wouters, N. Raghavan, G. Kar, A. Belmonte, Y.-Y. Chen, B. Govoreanu, and M. Jurczak. "Intrinsic switching variability in HfO2 RRAM." In: *2013 5th IEEE International Memory Workshop*. 2013, pp. 30–33. DOI: 10.1109/IMW.2013.6582090 (cit. on p. 31).

[52] Stefano Ambrogio, Simone Balatti, Antonio Cubeta, Alessandro Calderoni, Nirmal Ramaswamy, and Daniele Ielmini. "Statistical Fluctuations in HfOx Resistive-Switching Memory: Part I - Set/Reset Variability." In: *IEEE Transactions on Electron Devices* 61.8 (2014), pp. 2912–2919. DOI: 10.1109/TED.2014.2330200 (cit. on p. 31).

[53] Milo V. Ielmini D. "Physics-based modeling approaches of resistive switching devices for memory and in-memory computing applications." In: *Journal of Computational Electronics* 16.4 (2017), pp. 1121–1143. DOI: 10.1007/s10825-017-1101-9 (cit. on p. 32).

[54] Shimeng Yu, Ximeng Guan, and H.-S. Philip Wong. "On the stochastic nature of resistive switching in metal oxide RRAM: Physical modeling, monte carlo simulation, and experimental characterization." In: (2011), pp. 17.3.1–17.3.4. DOI: 10.1109/IEDM.2011.6131572 (cit. on p. 33).

[55] Valerio Milo, Artem Glukhov, Eduardo Pérez, Cristian Zambelli, Nicola Lepri, Mamathamba Kalishettyhalli Mahadevaiah, Emilio Pérez-Bosch Quesada, Piero Olivo, Christian Wenger, and Daniele Ielmini. "Accurate Program/Verify Schemes of Resistive Switching Memory (RRAM) for In-Memory Neural Network Circuits." In: *IEEE Transactions on Electron Devices* 68.8 (2021), pp. 3832–3837. DOI: 10.1109/TED.2021.3089995 (cit. on pp. 35, 83, 86–88).

[56] Óscar G. Ossorio, Eduardo Pérez, Salvador Dueñas, Helena Castán, Héctor García, and Christian Wenger. "Effective Reduction of the Programing Pulse Width in Al: HfO2-based RRAM Arrays." In: (2019), pp. 1–4. DOI: 10.1109/EUROSOI-ULIS45800.2019.9041880 (cit. on p. 36).

[57] E. Pérez, A. Grossi, C. Zambelli, P. Olivo, R. Roelofs, and Ch. Wenger. "Reduction of the Cell-to-Cell Variability in Hf1-xAlxOy Based RRAM Arrays by Using Program Algorithms." In: *IEEE Electron Device Letters* 38.2 (2017), pp. 175–178. DOI: 10.1109/LED.2016.2646758 (cit. on pp. 36, 37).

[58] Valerio Milo, Francesco Anzalone, Cristian Zambelli, Eduardo Pérez, Mamathamba K. Mahadevaiah, Óscar G. Ossorio, Piero Olivo, Christian Wenger, and Daniele Ielmini. "Optimized programming algorithms for multilevel RRAM in hardware neural networks." In: (2021), pp. 1–6. DOI: 10.1109/IRPS46558.2021.9405119 (cit. on p. 39).

[59] M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov. "Training and operation of an integrated neuromor-

phic network based on metal-oxide memristors.” In: *Nature* 521.7550 (2015), pp. 61–64. DOI: 10.1038/nature14441 (cit. on p. 78).

[60] Peng Yao, Huaqiang Wu, Bin Gao, Sukru Burc Eryilmaz, Xueyao Huang, Wenqiang Zhang, Qingtian Zhang, Ning Deng, Luping Shi, H. S. Philip Wong, and He Qian. “Face classification using electronic synapses.” In: *Nature Communications* 8.1 (2017), p. 15199. DOI: 10.1038/ncomms15199 (cit. on p. 78).

[61] Can Li, Daniel Belkin, Yunning Li, Peng Yan, Miao Hu, Ning Ge, Hao Jiang, Eric Montgomery, Peng Lin, Zhongrui Wang, Wenhao Song, John Paul Strachan, Mark Barnell, Qing Wu, R. Stanley Williams, J. Joshua Yang, and Qiangfei Xia. “Efficient and self-adaptive in-situ learning in multilayer memristor neural networks.” In: *Nature Communications* 9.1 (2018), p. 2385. DOI: 10.1038/s41467-018-04484-2 (cit. on p. 78).

[62] V. Milo, C. Zambelli, P. Olivo, E. Pérez, M. K. Mahadevaiah, O. G. Ossorio, Ch. Wenger, and D. Ielmini. “Multilevel HfO2-based RRAM devices for low-power neuromorphic networks.” In: *APL Materials* 7.8 (2019), p. 081120. DOI: 10.1063/1.5108650. eprint: https://doi.org/10.1063/1.5108650 (cit. on p. 79).

[63] Simon Haykin. *Neural networks and learning machines, 3/E*. Pearson Education India, 2010 (cit. on p. 81).

[64] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. “Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights.” In: *CoRR* abs/1702.03044 (2017). arXiv: 1702.03044 (cit. on p. 83).