Efficient and Reliable In-Memory Computing Systems Using Emerging Technologies: From Device to Algorithm

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Masoud Zabihi

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Sachin S. Sapatnekar, Advisor

November, 2021

Acknowledgements

I would like to thank my academic advisor, Prof. Sachin S. Sapatnekar, for giving me the chance to work on various projects and learn from him. I appreciate his elaborated feedback on manuscripts. I have experienced academic research with him for over five years, and I am extremely grateful that I had such a rewarding experience.

This thesis is an interdisciplinary effort. I was fortunate to have the guidance and feedback of Prof. Ulya Karpuzcu, Prof. Jian-Ping Wang, and Prof. Chris Kim. Their comments stemming from their respected field have enriched my research works. Salonik Resch, Zamshed Chowdhury, Husrev Cilasun, Zhengyang Zhao, DC Mahendra, Thomas Peterson, Yang Lv, Arvind Sharma, Meghna Mankalale, Ibrahim Ahmed, Vidya Chhabria, and Kishor Kunal were my colleagues and collaborators. I am genuinely grateful for having all those technical discussions and weekly meetings with them.

I would like to thank the graduate school at UMN for the doctorate dissertation fellowship for 2020-2021. I am very grateful to the NSF, which was the major funding source of my works. I would like to acknowledge the funding support from SRC and DARPA on the CSPIN project.

I would like to thank all my family members who have always been there for me and have encouraged me throughout my academic endeavors.

Finally, I am grateful for having wonderful friends in my life who have supported me on various occasions during my Ph.D.

Dedication

To my ”Kherza Ha”:

Ava and Nila

Abstract

Big data applications are memory-intensive, and the cost of bringing data from the memory to the processor involves large overheads in energy and processing time. This has driven the push towards specialized accelerator units that can perform computations close to where the data is stored. Two approaches have been proposed in the past: (1) near-memory computing places computational units at the periphery of memory for fast data access, and (2) true in-memory computing uses the memory array to perform computations through simple reconfigurations. Although there has been a great deal of recent interest in the area of in-memory computing, most solutions that are purported to fall into this class are really near-memory processors that perform computation near the edge of memory arrays/subarrays rather than inside it.

This thesis discusses several years of effort in developing various true in-memory computation platforms. These computational paradigms are designed using different emerging non-volatile memory technologies such as spin transfer torque (STT) magnetic tunnel junction (MTJ), spin Hall effect (SHE) MTJ, and phase change memory (PCM) device. The proposed platforms in this thesis effectively eliminate the energy and delay overhead associated with data communication. Our approach is digital, unlike prior analog-like in-memory/near-memory solutions, which provides more robustness to process variations, particularly in immature technologies than analog schemes.

The thesis covers alternatives at the technology level, followed by a description of how the in-memory computing array is designed, using the basic non-volatile unit (such as MTJ) and some switches, to function both as a memory and a computational unit. This array is then used to build gates and arithmetic units by appropriately interconnecting memory cells, allowing high degrees of parallelism. Next, we show how complex arithmetic operations can be performed through appropriate scheduling (for adders, multipliers, dot products) and data placement of the operands. We demonstrate how this approach can be used to implement sample applications, such as neuromorphic inference engine and a 2D convolution, presenting results that benchmark the performance of these CRAMs against near-memory computation platforms. The performance gains can be attributed to (a) highly efficient local processing within the memory, and (b) high levels of parallelism in rows of the memory. For our in-memory computing platforms, wire resistances and variations are a substantial source of non-ideality that must be taken into account during the implementations. To ensure the electric correctness of implementations, we have developed different frameworks to analyze the parasitic effects of wires based on actual layout considerations. We have demonstrated that interconnect parasitics have a significant effect on the performance of the in-memory computing system and have developed a comprehensive model for analyzing this impact. Using this methodology, we have developed guidelines for the physical parameters such as array size and numbers of rows and columns.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Today's computational engines are inadequately equipped to handle the demands of future big-data applications. With the size of data sets growing exponentially with time [14], the computational demands for data analytics applications are becoming even more forbidding, and the mismatch between application requirements and evolutionary improvements in hardware is projected to become more extreme. Current hardware paradigms have moved towards greater specialization to handle this challenge, and specialized units that enable memory-centric computing are a vital ingredient of any future solution.

Table 1.1: Communication vs. computation energy, adapted from [1].

Technology Node	40nm	10nm HP	10nm LP
Energy of 64-bit data communication versus computation	1.55×	5.75×	5.77×

However, key bottlenecks for large-scale data analytics applications, even in state-of-the-art technology solutions, are the memory capacity, communication bandwidth, and performance requirements within a tight power budget. Technology scaling to date has improved the efficiency of logic more than communication, and communication energy dominates computation energy [15, 16]. Table 1.1 compares the cost of computation (a double-precision fused multiply add) with communication (a 64-bit read from an on-chip SRAM). The ratio of communication energy to computation energy increases from 1.55× at 40nm to approximately 6× at 10nm, for both the high performance (HP) and low power (LP) variants. Even worse, transferring the same quantity of data off-chip, to main memory, requires more than 50× computation energy even at 40nm [1]. Such off-chip accesses become increasingly necessary as data sets grow larger, and even the cleverest latency-hiding techniques cannot conceal their overhead. At the same time, the inevitable trend of higher degrees of parallel processing hurts data locality and results in increased execution time, power, and energy for data communication [15].

Moreover, general-purpose processors are often inefficient in computing for emerging applications: this has motivated a trend towards specialized accelerator units, tailored to specific classes of applications that they can efficiently execute. The trend of increasing data set sizes, coupled with the large cost (in terms of both energy and latency) of transporting data to the processor, have prompted the need for a significant departure from the traditional model of CPU-centric computing. An effective way to overcome the memory bottleneck and maintain the locality of computing operations is to embed compute capability into the main memory. In recent years, two classes of approaches have been proposed (see Fig. 1.1):

- *near-memory computing* places computational units at the periphery of memory for fast data access.
- *true in-memory computing* uses the memory array to perform computations through simple reconfigurations.

Although there has been a great deal of recent interest in the area of in-memory computing, most solutions that are purported to fall into this class are really near-memory processors that perform computation near the edge of memory arrays/subarrays rather than inside it. In this thesis, we discuss the design of true in-memory computation platforms using different emerging non volatile device technologies such as spin transfer torque (STT) magnetic tunnel junction (MTJ), spin Hall effect (SHE) MTJ, and phase change memory (PCM). In addition, the feasibility of implementations and electric correctness of our designs are studied in this thesis.

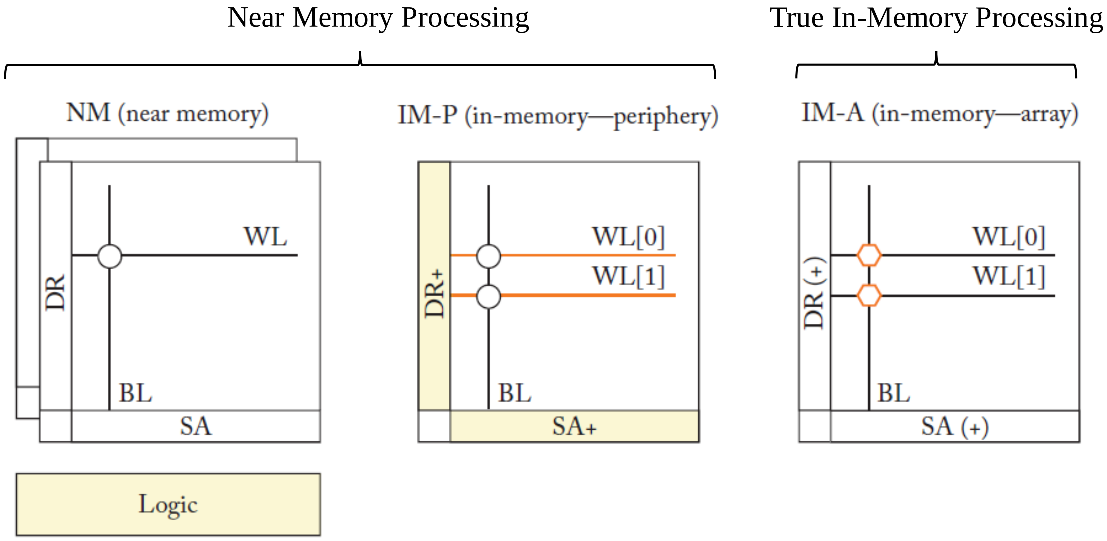


Figure 1.1: Taxonomy of in- and near-memory computing. Computation happens in the modules colored in yellow [11].

1.2 Thesis contributions

This thesis develops an approach that is a true in-memory computational platform. The underlying spintronics technology used in most of the propose solutions eliminates the problems of other substrates that have been proposed for true in-memory computation as it is more robust (high-endurance, easier to manufacture), accurate, and can be used to build noise-resilient circuits. The approach is based on the concept of the Computational Random Access Memory (CRAM) [17], which proposed an outline of an in-memory computation model. However, many issues have had to be tackled to move this idea from concept to practice. Our works shows that CRAM approach has significant potential of benefiting variety of applications [2, 18–31]. This thesis answers questions such as: (a) How can the practical issues involved in integrating this system in a silicon chip be resolved? (b) How can the primitive computational structures (logic gates) in [17] be extended to build larger structures (adders, multipliers, signal processing units, neural networks), which must perform the function of thousands to millions of gates? (c) How can operations be scheduled in the CRAM to achieve high levels of parallelism? (d) Given that the switching speed of the core switching device of the CRAM can be slower than today's CMOS technology, does the CRAM really deliver energy and power improvements over today's technologies? (e) What are the effects wire variations and non-idealities on the proposed CRAM implementations (f) How can other spintronic-based three terminal devices be integrated to build a CRAM-like architecture? (g) How can other non-spintronics-based technologies (e.g., PCM) perform true in-memory computation similar to CRAM?

Traditional solutions of in-memory computing have been circumscribed by disciplinary boundaries, primarily driven by computer architecture researchers [32,33]. Finding effective answers to the questions requires an interdisciplinary effort that breaks down such boundaries. This thesis research provides a strong bridge from novel materials/devices that can be used to build the CRAM, to circuit design methods that can build larger building blocks such as adders, multipliers, and neurons on the CRAM, to system-level and computer-architecture-level issues such as CRAM operation scheduling and the design of peripheral circuitry for the CRAM array. The thesis research is conducted by considering state-of-the-art device properties which working in close collaboration with experts in physics and materials science and understanding the available capabilities and translating them into usable circuit structures. At the circuit level, many innovations have been present to pave the way for further system-level investigations. In addition, a clear strategy for the design of core CRAM functionalities for smaller computation blocks was required to ensure the viability of CRAM at the application levels. This work revolves around building this robust interdisciplinary bridge to make a practical implementation of the CRAM possible. The specific contributions of this thesis are:

- *Developing STT MTJ based CRAM (Chapter 2):* The overall architecture of STT CRAM is obtained by modification of the STT MRAM architecture and including additional access transistors and lines to enable within array computation. Scheduling tables and algorithms for the implementations for various computational blocks such as multiplier and adders are developed. Using these computational blocks, methods for the implementations of more complex applications such as neuromorphic digit recognition and 2D convolution are proposed. It is demonstrated that STT CRAM can outperform a near memory processing unit in terms of energy and execution time of orders of magnitude. A bottom-up approach is followed where the proposed designs include considerations from material/device level, to the circuit level, to architecture and application levels.

- *Constructing a framework for analyzing parasitic effects of wires (Chapter 3):* The parasitic effects of wires are considered on the proposed STT CRAM implementations. In in-memory computing systems, wire non-idealities are a major cause of the failure of the implementations. In Chapter 3, a framework to study the effect of the parasitics on CRAM performance is developed. The framework is constructed based on realistic layout considerations based a 7nm technology node that is suitable for the STT MTJ integration. Electrical correctness of the proposed implementations is ensured by determining optimal layout specifications (e.g., FinFET sizes and aspect ratio) as well as CRAM subarray sizes.

- *Developing SHE MTJ based CRAM (Chapter 4):* A redesigned CRAM around a new MTJ based on the spin-Hall effect (SHE) is proposed, providing greatly improved energy efficiency. Similar to the development of STT CRAM, scheduling tables and implementation methods for various computational blocks are proposed, and the energy and execution time for the implementation of applications on SHE CRAM are compared with those of a near-memory processing system.

- *3D XPoint as a true in-memory computation platform (Chapter 5):* Beyond spintronics-based devices, we study how utilization of other emerging memory technologies (such as PCM) can help in performing true CRAM-like in-memory processing. 3D XPoint is a new class of memory technology that fills a unique place in the memory hierarchy [34]. The storage element in 3D XPoint is based on PCM non-volatile device technology. We demonstrate the feasibility of true in-memory processing computation on 3D XPoint array. We implement a neuromorphic engine on the 3D XPoint subarray and perform a threshold matrix-vector multiplication (TMVM) operation. Analysis considers effect of the variation and parasitics effects to ensure the electric correctness of our implementations.

Chapter 2

In-Memory Processing on the Spintronic CRAM: From Hardware Design to Application Mapping

2.1 Introduction

In this chapter, an approach is presented based on the spintronics-based computational random access memory (CRAM) paradigm. The CRAM concept [17] uses a small modification to the MTJ-based memory cell that enhances its versatility and enables logic operations through reconfiguration that enables the use of current-steered logic. Unlike many other approaches, this method fits the description of *true in-memory computing*, where computations are performed natively within the memory array and massive parallelism is possible, e.g., with each row of the memory performing an independent computation. The CRAM-based approach is digital, unlike prior analog-like in-memory/near-memory solutions [35, 36], which provides more robustness to variations due to process drifts, particularly in immature technologies than analog schemes. This sensitivity implies that digital implementations can achieve superior recognition accuracy over analog solutions when the full impact of errors and variations are factored in.

Our solution is based on spintronics technology, which is attractive because of its robustness, high endurance, and its trajectory towards fast improvement [6, 37]. The outline of the CRAM approach was first proposed in [17], operating primarily at the technology level with some expositions at the circuit level. The work was developed further to show system-level applications and performance estimations in [38]. In this work, we bridge the two to provide an explicit link between CRAM technology, circuit implementations, and operation scheduling. We present technology alternatives and methods for building gates and arithmetic units, study scheduling and data placement issues, and show how this approach can be used to implement a sample application, which is chosen to be a neuromorphic inference engine for digit recognition.

The rest of the chapter is organized as follows. Section 2.2 discusses CRAM architecture. In Section 2.3, we present an approach for designing arithmetic function at the device, gate, and functional levels. Given this design, a more detailed elaboration on scheduling CRAM operations is discussed in Section 2.4. We elaborate on two example applications, corresponding to implementations of 2D convolution and a neural inference engine, in Section 2.5. We discuss the evaluation and results in Section 2.6, related work in Section 2.7, and then conclude the chapter in Section 5.7.

2.2 CRAM Architecture

2.2.1 MTJ Devices

The unit storage cell used in a typical STT-MRAM is an MTJ, which is composed of two ferromagnetic layers – a fixed polarizing layer and a free layer – separated by an ultrathin nonconductive MgO barrier [39]. We consider perpendicular MTJ (PMTJ) technology, where both the free layer and the fixed layer are magnetized perpendicular to the plane of the junction. When the magnetization orientations of the two layers are parallel to each other (referred to as the P state), applying a voltage across the MTJ causes electrons to tunnel through the ultrathin nonconductive layer without being strongly scattered, as a result of which we have high current flow and relatively low resistance, R_P [40]; when the magnetization orientations of two layers are anti-parallel to each other (referred to as the AP state), the MTJ has a higher resistance, R_{AP}. In this way, an MTJ can store logic 1 and 0 depending on its resistance state, and we define logic 1 and 0 for the AP and P states, respectively [41]. A critical attribute of an MTJ is the tunneling magnetoresistance ratio (TMR), defined as

$$\mathrm{TMR} = \frac{R_{AP} - R_P}{R_P} \tag{2.1}$$

With an electrical current flowing through the MTJ, the magnetization direction of the free layer can be reversed due to the spin-transfer-torque (STT) effect, and thus the MTJ can be switched between P state and AP state. To flip the magnetization direction of the free layer, the current density should be larger than a threshold switching current density, J_c, which is technology-dependent.

2.2.2 The CRAM Array

The general structure of the spintronic CRAM is illustrated in Fig. 2.1. The overall configuration of the CRAM array is very similar to the standard 1-transistor 1-MTJ

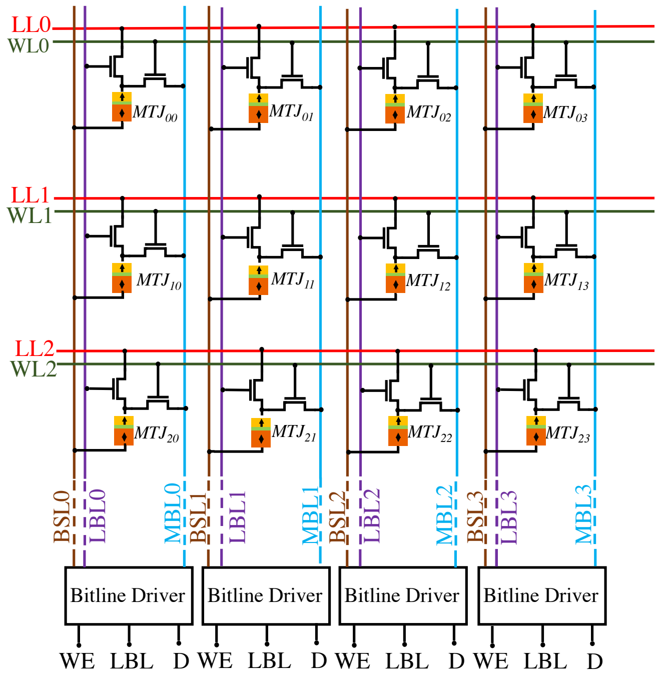


Figure 2.1: Overall structure of the CRAM.

(1T1MTJ) STT-MRAM, except that the CRAM uses a 2T1MTJ bit-cell, with one additional transistor. Like the standard STT-MRAM memory array, the MTJ in each bit-cell is addressed using the memory word line (WL). The second transistor in the bit-cell, which enables logic operations, is enabled by selecting the logic bit line (LBL) for the transistor while turning off WL. The array can operate in two modes:

Memory mode: When the WL is high, it turns on an access transistor in each column and enables data to be read from or written into the MTJ through the memory bit line (MBL). The second transistor is turned off during this mode by holding down LBL, and the configuration is effectively identical to a bit cell in a memory array.

Logic mode: Turning on the LBL allows the MTJ to be connected to a logic line (LL) in each row. In the logic mode, several MTJs in a row are connected to the logic line. To realize a logic function with multiple inputs and one output, an appropriate voltage is applied to the bitlines. Since the states of the input MTJs are expressed in terms of their resistance, the current through the output MTJ depends on the input MTJ resistances, and if it exceeds the critical switching current, I_c, the output MTJ state is altered.

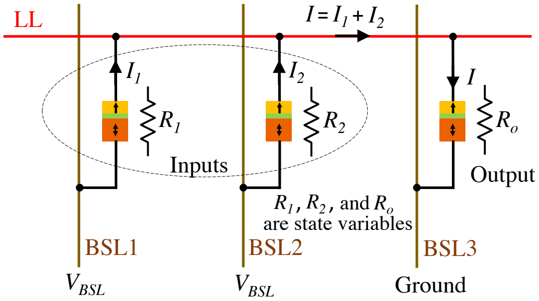


Figure 2.2: Performing a logic operation in a row of the CRAM array.

To understand the logic mode more clearly, consider the scenario where three MTJ devices are connected to the logic line, as shown in Fig. 2.2. The state variables are expressed in terms of the MTJ resistance, where resistances R_1 and R_2 correspond to the states of the two inputs, and R_o is the output MTJ resistance. Before the computation starts, the output MTJ state is set to a preset value. The bit select lines (BSLs) of the input MTJs are connected to a pulse voltage, while that of the output MTJ is grounded. This configuration corresponds to the application of a voltage V_{BSL} across a resistance of $(R_1 \mid\mid R_2)$ in series with R_o. As a result, a current I flows through the logic line:

$$I = V_{BSL} \Bigg/ \left[\left(\frac{R_1 R_2}{R_1 + R_2} \right) + R_o \right] \tag{2.2}$$

If $I > I_c$, where I_c is the critical threshold current required to switch the output MTJ from it preset value, the output state changes; otherwise it remains the same.

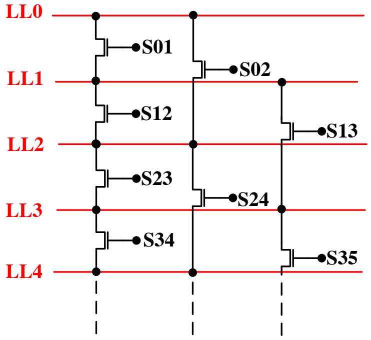


Figure 2.3: Switches between rows for inter-row transfers.

2.2.3 Performing Logic Operations Across Rows

The scheme in Fig. 2.2 shows how logic operations can be carried out between MTJs in the same row. However, it is important at times to perform logic operations that traverse multiple rows. To enable inter-row communication, we augment the array of Fig. 2.2 by inserting switches between logic lines which, when turned on, allow MTJs in different rows to communicate. It is unrealistic to allow every row to communicate with every other row, and nor is this required in typical computations.

To maintain the simplicity of the architecture, an LL in row i is connected through switches to the LLs in its two nearest adjacent rows, i.e., as illustrated in Fig. 2.3, the LL in row $i-2$, $i-1$, $i+1$, and $i+2$, if they exist. In performing in-memory computations, it is important to ensure that an internal data traffic bottleneck is not introduced: in fact, the best solutions will perform very local data movement. This is the reason why our CRAM architecture only allows direct movement to the two nearest adjacent rows. Data movement to more distant rows is not prohibited, but must proceed in sequential hops of one or two rows. In principle, it is possible to connect every row to every other row. However, for n rows, such a scheme would add $C(n,2) = O(n^2)$ transistors, and would also introduce significant routing overheads. In contrast, our scheme adds $2n$ transistors, and these local connections can be made quite easily. To illustrate the use of this structure, let us consider a very common operation where the output of an operation in row N must be moved to row M for the next operation. Each such operation requires the implementation of a BUFFER gate, which copies a value from one row to another. To move from row N to row M, the data can "jump" two rows at a time to reach its destination, except when the destination is one row away, where it "jumps" one row. For example, to copy a value from row 0 to row 7, for example, one could move first to row 2, then row 4, then row 6, and then finally to row 7. It is easy to see that in general the number of steps required to transfer one bit from row M to row N is $\left\lceil \frac{|M-N|}{2} \right\rceil$.

Other interconnection schemes may also be possible and could reduce the communication overhead, depending on application characteristics. The scheme described above is built on the assumption that energy considerations dictate that most inter-row communication must be local.

2.2.4 Peripheral Circuitry

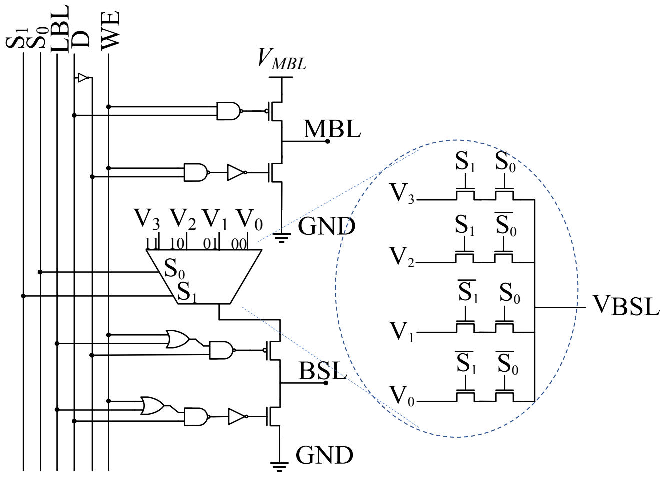


Figure 2.4: Bitline driver circuitry.

The voltages on BSL, LBL, and MBL are set by the bitline drivers. While LBL takes on normal V_{dd} values, required to turn on the access transistor, the chosen voltage on BSL depends on the logic function being implemented. As we will show in Section 2.3.2, this voltage is on the order of 100s of mV in today's technologies and 10s of mV in advanced technologies. The bitline drivers are illustrated in Fig. 2.4: the inputs are WE, D, and LBL, and the outputs BSL and MBL are generated using the circuitry shown in Fig. 2.4.

The generation of the MBL and BSL signals in Fig. 2.2 is illustrated in Fig. 2.4. In *memory mode*, LBL is grounded and line D is used to control the direction of the current. In case of a write operation, WE is set to V_{dd}, and if D is also at V_{dd}, then current is injected from MBL to BSL; otherwise, if D is grounded, the current direction is reversed. For a read operation, WE is grounded and both drivers are off; in this case, MBL is separately driven and connected to the sense amplifier. In *logic mode*, WL and WE are grounded, and LBL is at V_{dd}. If D is also at V_{dd}, then the driver connects BSL to ground, while if D is grounded, then BSL is connected to V_{BSL}.

2.3 Designing Arithmetic Functions

2.3.1 Device-level Models

In evaluating the performance of the CRAM, we consider two sets of device-level models whose parameters are listed in Table B.1: (i) Today's MTJ technology, corresponding to a mainstream process today, and (ii) Advanced MTJ technology, corresponding to a realistic future process. The value of the latter point is that due to the rapid evolution of MTJ technology, using only today's node is likely to be pessimistic. Moreover, by using technology projections, the evaluation on an advanced MTJ technology provides a clear picture of design issues and bottlenecks for this method. For each technology, the table provides specifics of the MTJ materials and dimensions, the TMR, the resistance-area

(RA) product, the critical switching current density, J_c, the critical switching current, I_c, the write time, t_{wr}, as well as the MTJ resistance in each state.

Table 2.1: MTJ specifications

Parameters	Today's MTJ	Advanced MTJ
MTJ type	Interfacial PMTJ	Interfacial PMTJ
Material system	CoFeB/MgO/ CoFeB	CoFeB (SAF)/MgO/ CoFeB
MTJ diameter	45nm	10nm
TMR	133% [42]	500%
RA product	$5\Omega\mu\text{m}^2$	$1\Omega\mu\text{m}^2$ [43]
J_c	$3.1 \times 10^6\text{A/cm}^2$	10^6A/cm^2
I_c	$50\mu\text{A}$	$0.79\mu\text{A}$
t_{wr}	3ns [44]	1ns [42]
R_P	$3.15\text{K}\Omega$	$12.73\text{K}\Omega$
R_{AP}	$7.34\text{K}\Omega$	$76.39\text{K}\Omega$

In general, for the CRAM application, a higher TMR is beneficial since this helps differentiate between the 0 and 1 states more effectively. To select the parameters for the Advanced MTJ technology, we consider various roadmaps and projections, as well as consultations with technology experts. Today, the largest demonstrated TMR is 604% at room temperature for an MTJ built using a material stack of CoFeB/MgO/CoFeB [6,45]. However, this MTJ uses a thick MgO layer, which results in a large RA product; moreover, it uses in-plane magnetization, which requires larger area due to its need for shape anisotropy in the plane, and is less preferred over perpendicular MTJ magnetization. While the best TMR for perpendicular MTJs in the lab today is about 208% [46], there are pathways to much higher TMRs. Accordingly, we set the TMR for the Advanced MTJ to 500%. This is further supported by predictions that show that a TMR of 1000% at room temperature will be attainable by 2024 [6]. The RA product can be tuned using new tunneling barrier materials (e.g., MgAlO), or reducing the MgO thickness while maintaining crystallinity.

2.3.2 Gate-level Design

When MTJs are connected together in logic mode, the type of gate functionality that results from the connection can be controlled by two factors: (i) the voltage applied on the BSL lines, which appears across the connected MTJ devices, and (ii) the logic value to which the output MTJ is preset. The corresponding bias voltage range and the preset value to implement each gate are summarized in Table 3.1. The output preset for each gate is unique and depends on the gate type rather than on MTJ technology parameters.

Consider the case where the configuration in Fig. 2.2 is used to implement a NAND gate. Since R_{AP} = (TMR+1)R_P, and a logic 0 corresponds to R_P, from Eq. (4.1), we have:

$$\begin{aligned} I_{00} &= V_{BSL} \Big/ \left(\frac{R_P}{2} + R_o\right) \\ I_{01} = I_{10} &= V_{BSL} \Big/ \left(\left(\frac{\text{TMR}+1}{\text{TMR}+2}\right) R_P + R_o\right) \\ I_{11} &= V_{BSL} \Big/ \left(\frac{(\text{TMR}+1)R_P}{2} + R_o\right) \end{aligned} \tag{2.3}$$

The requirements for the NAND gate is that the first two cases should result in logic 1 at the output MTJ, but the last case should keep the output value at logic 0. Using the fact that TMR > 0, it is easy to verify that the current monotonically decreases as $I_{00} > I_{01} = I_{10} > I_{11}$. Therefore, if the output is preset to logic 0, then by an appropriate choice of V_{BSL}, the first two cases can result in a current that exceeds I_c, thus switching the output while the last can result in a current below I_c, keeping the output at logic 0. A similar argument can be made to show that if the output is preset to 1, the gate will not function correctly because it requires the first two cases (higher currents) not to induce switching, while the last case (lowest current) must induce switching. The same arguments can be used to argue that an AND implementation should be preset to logic 1, allowing switching in the 00 and 01/10 cases, but not the 11 case.

It can further be seen that an XOR cannot be naturally implemented in a single gate under this scheme: depending on the preset value, it requires switching for the 00 and 11 cases but not 01/10, or vice versa. Neither case follows the trends by which the current I increases. Therefore, like CMOS, an XOR must be implemented using multiple stages of logic.

For the NAND gate, for a preset output value of 0, $R_o = R_P$. Therefore the results for the three cases are:

$$\begin{aligned}
I_{00} &= \frac{V_{BSL}}{R_P}\left(\frac{2}{3}\right)\\
I_{10} = I_{01} &= \frac{V_{BSL}}{R_P}\left(\frac{\text{TMR}+2}{2\text{TMR}+3}\right)\\
I_{11} &= \frac{V_{BSL}}{R_P}\left(\frac{2}{\text{TMR}+3}\right)
\end{aligned} \tag{2.4}$$

The requirements for the NAND gate is that the first two cases should induce switching to logic 1, but the last case should keep the output value at logic 0. Therefore,

$$\begin{aligned}
&\left(\frac{\text{TMR}+2}{2\text{TMR}+3}\right)\frac{V_{BSL}}{R_P} > I_c > \left(\frac{2}{2\text{TMR}+3}\right)\frac{V_{BSL}}{R_P},\\
\text{i.e., } &\left(\frac{2\text{TMR}+3}{\text{TMR}+2}\right) I_c R_p < V_{BSL} < \left(\frac{2\text{TMR}+3}{\text{TMR}+2}\right) I_c R_P.
\end{aligned} \tag{2.5}$$

From the values of R_P, TMR, and I_c provided in Table B.1 and the requirement that the 00 and 10/11 cases should switch, while the 11 case should not, we can obtain the values in Table 3.1. For NAND gate, 270.0mV $<V_{BSL}<$ 354.5mV for today's MTJs, and 18.6mV $<V_{BSL}<$ 40.2mV for advanced MTJs. Similar methods are used for other gates. From the table, it can be seen that the voltage V_{BSL} required to implement each gate type using today's MTJ technology is higher than that for the advanced MTJ technology.

Table 2.2: Bias voltage ranges and output preset values

Gate	Bias voltage range		Output
	Today's MTJ	Advanced MTJ	preset
NOT	315.0 – 551.5mV	20.1 – 70.4mV	0
BUFFER	551.5 – 788.0mV	70.4 – 120.6mV	1
AND	506.5 – 591.0mV	68.9 – 90.5mV	1
NAND	270.0 – 354.5mV	18.6 – 40.2mV	0
OR	472.7 – 506.5mV	65.3 – 68.9mV	1
NOR	236.2 – 270.0mV	15.0 – 18.6mV	0
MAJ3	459.6 – 481.5mV	64.9 – 67.8mV	1
$\overline{\text{MAJ3}}$	223.1 – 245.0mV	14.6 – 17.5mV	0
MAJ5	435.4 – 443.2mV	63.3 – 64.3mV	1
$\overline{\text{MAJ5}}$	198.9 – 206.7mV	13.0 – 14.0mV	0

For a NAND gate, the lower end and upper of the bias voltage (V_{BSL}) is shown in Eq. (2.5). For each gate type, if we denote the lower and upper ends of the bias voltage range as V_{min} and V_{max}, respectively, then we can define the noise margin, NM, as:

$$NM = (V_{max} - V_{min})/V_{mid} \tag{2.6}$$

$$\text{where } V_{mid} = (V_{max} + V_{min})/2 \tag{2.7}$$

This metric provides a measure of the range of the voltage swings, normalized to the mean.

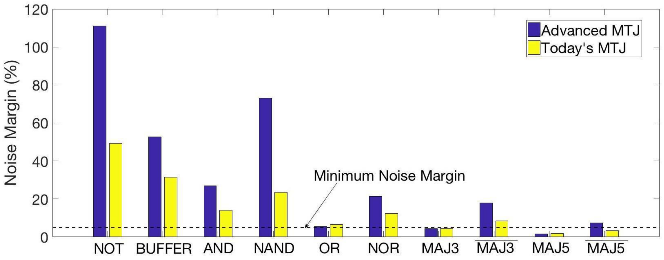


Figure 2.5: A comparison of the noise margin for various gate types, implemented in a CRAM today's MTJs and advanced MTJs, as defined in Table B.1.

Fig. 2.5 shows the noise margin for various gate types. It can be seen that for Advanced MTJs, the noise margin in most cases is about 2X larger than those of today's

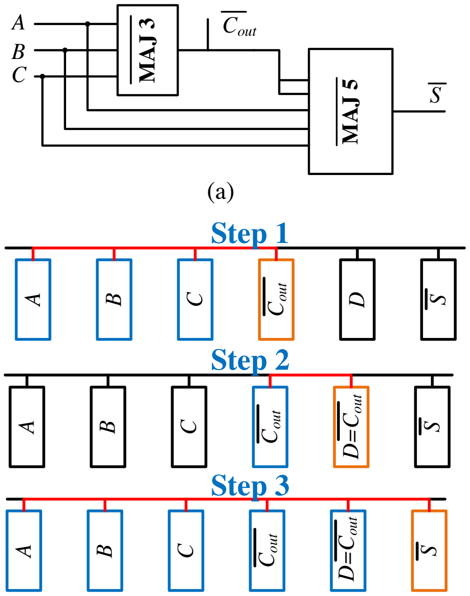
MTJ. This noise margin is intended to capture the level of resilience of each gate type to shifts in parameter values due to process variations, supply voltage variations, thermal noise, etc. While it is difficult to know the level of such drifts, since these technologies are still evolving and are in a high state of flux. In this work, we choose a threshold for the minimum acceptable noise margin in this work as $NM = 5\%$. From the figure, it can be seen that the MAJ3, MAJ5, and OR gates for both technologies and the $\overline{\text{MAJ5}}$ gate for today's technology fall below this threshold, and are not used here.

2.3.3 Functional-level Design

In [17], NAND based logic was applied to implement a full adder (FA) using CRAM. A NAND-based implementation of FA requires 9 stages of logic. As shown in [17], Carry and Sum can, respectively, be generated after 9 and 8 CRAM computation steps. This large number of sequenced operations for an addition can can incur high delays. FAs can be implemented more efficiently using majority logic [47] instead of NAND gates. While such implementations can reduce the number of steps required to implement a FA in the CRAM, MAJ3 and MAJ5 have low NM values (Fig. 2.5), but $\overline{\text{MAJ3}}$ and $\overline{\text{MAJ5}}$ gates have sufficient NM for advanced MTJs. Therefore, we adapt the MAJ-based FA designs to use complementary MAJ logic for advanced MTJs, and stay with NAND gates for today's MTJs.

We propose a modification of the MAJ-based FA using the $\overline{\text{MAJ}}$-based logic structure shown in Fig. 2.6(a) to implement the complement of a FA. It can easily be verified that this correctly produces the outputs $\overline{S}$ and $\overline{C_{out}}$ based on input bits A, B, and C. We will defer the precise set of scheduling operations to our discussion in Section 2.4.

To demonstrate how this complemented FA can be used to build an n-bit adder, we show a 4-bit ripple carry adder in Fig. 2.7. The LSB (zeroth bit) uses the logic in Fig. 2.6 to generate the complemented output carry, which is the complemented input carry $\overline{C_1}$



(a)

(b)

Figure 2.6: (a)The full adder implementation based on $\overline{\text{MAJ}}$ logic (b) scheduling CRAM operations on the adder. For simplicity, the output preset before each step is not shown in the schedule above.

of the first bit, and to generate the complemented sum bit $\overline{S_0}$. The latter is taken through an inverter to generate S0. Instead of inverting $\overline{C_1}$, we use "bubble-pushing" to implement the first bit, based on the observation that:

$$C_{out} = \overline{\text{MAJ3}}(\overline{A}, \overline{B}, \overline{C}) \tag{2.8}$$

$$S = \overline{\text{MAJ5}}(\overline{A}, \overline{B}, \overline{C}, C_{out}, C_{out}) \tag{2.9}$$

Thus, we invert A_1 and B_1, which are not on the critical path, instead of inverting $\overline{C_1}$, to generate S_1 and C_2, and so on. In general, for an n-bit adder, alternate bits use true and complemented inputs to the $\overline{\text{MAJ}}$-based FA. In this proposed scheme inversions are not required for any C_{out} bits (except for the MSB for an n-bit adder where n is odd – but it is unusual for n to be odd in real applications). Explicit inversions (i.e., NOT gates) are only required for the Sum outputs of alternate FAs in the n-bit adder.

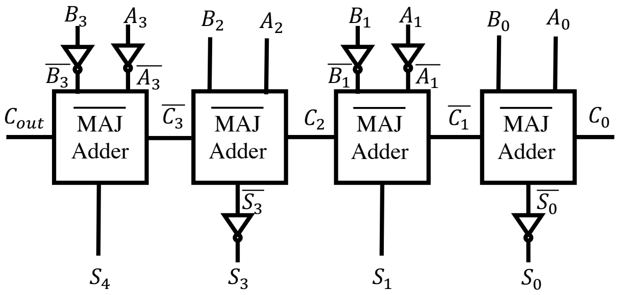


Figure 2.7: 4-bit ripple carry adder using bubble-pushing.

2.4 Scheduling CRAM Operations

Scheduling an n-bit addition on the CRAM: We begin with the implementation of single-bit addition in the CRAM and then move to multibit additions. The FA structure involves multiple steps that implement $\overline{\text{MAJ3}}$, $\overline{\text{MAJ5}}$, NOT, and BUFFER, and these computational steps are shown in Fig. 2.6(b). For each step, it is assumed that initializations (output presets) are performed before the shown computational steps.

Step 1 For the FAs corresponding to odd-numbered bits in n-bit addition, the input is not complemented. In Step 1, we initialize the Cout cell to 0, and then compute $\overline{C_{out}} \leftarrow \overline{\text{MAJ3}}(A, B, C)$ by activating the BLL transistor, after initializing the Cout cell to 0.

Step 2 We copy the computed C_{out} using D $\leftarrow$ BUFFER($\overline{C_{out}}$). The register D is used to store the value of $\overline{C_{out}}$, as two $\overline{C_{out}}$ operands are required for the next step.

Step 3 We compute $\overline{S} \leftarrow \overline{\text{MAJ5}}(A, B, C, \overline{C_{out}}, \overline{C_{out}})$.

In principle, this would have to be followed by Steps 4 and 5 (not shown in the figure), which use the NOT function to obtain the uncomplemented S and C_{out} outputs. However, bubble-pushing makes it unnecessary to invert a rippled carry output, and alternate output bits need no inversion on the sum bits, but need input inversions.

However, for odd-numbered FAs, we require a Step 4 to invert the Sum output, and for even-numbered bits, we add a “Step 0” that inverts the input bits A and B; note that neither of these is typically on the critical path.

The computation for even-numbered bits is analogous. We compute C_{out} using Equation (2.8), then copy it to another location D, and finally use Equation (2.9) to compute S.

We consider data placement and scheduling for an n-bit carry-propagate adder (CPA) using $n = 4$ to illustrate the idea, based on Fig. 2.7. Each of the four $\overline{\text{MAJ}}$-based FAs in this structure is implemented within a separate row of the CRAM, and the computation in each row is performed in separate steps that capture data dependencies.

Time	1	2	3	4	5	6	7	8	9
Row 0	$\overline{C_1}$	–	D_0	$\overline{S_0}$	S_0				
Row 1	–	$\overline{C_1}$	C_2	–	D_1	S_1			
Row 2	–	–	–	C_2	$\overline{C_3}$		D_2	$\overline{S_2}$	S_2
Row 3	–	–	–	–	–	$\overline{C_3}$	C_{out}	D_3	S_3

Figure 2.8: Scheduling table for the 4-bit CPA from $t = 1$ to 9.

The scheduling table of the 4-bit ripple carry adder is shown in Fig. 2.8, where the i^{th} bit-slice maps to CRAM row i. Once a carry in row i is generated, it is transferred to row $i + 1$. Thus, at $t = 1$, $\overline{C_1}$ is generated and is transferred to row 1 at $t = 2$; at $t = 3$, C2 is generated in row 1 and transferred to row 2 at $t = 4$, and at $t = 5$, $\overline{C_3}$ is generated and transferred to row 3 at $t = 6$. Now that all inputs to the MSB are available, using the schedule described in Fig. 2.6(b), three time units later, at $t = 9$, the computation is complete.

Multiplication: The dot notation is a useful tool to represent arithmetic operations [12]. The notation is illustrated in Fig. 2.9 for the addition and multiplication of two 4-bit binary digits. Each dot represents a place significance, and dots representing each input number correspond to its four bits, with weights of 1, 2, 4, and 8, from right to left. Fig. 2.9(a) shows that the sum of these two 4-bit numbers is a 5-bit number. The multiplication of two 4-bit numbers (Fig. 2.9(b)), generates a set of four shifted partial products that are added to generate the 8-bit product.

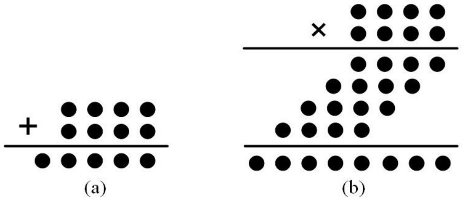


Figure 2.9: Dot notation representation [12]: (a) Addition of two 4-bit digits, (b) Multiplication of two 4-bit digits.

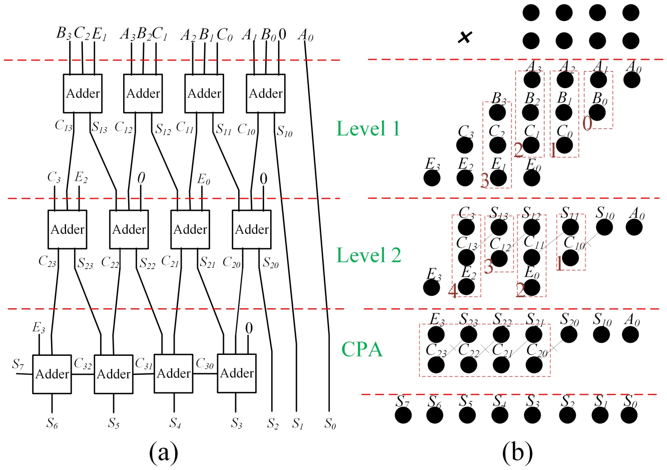
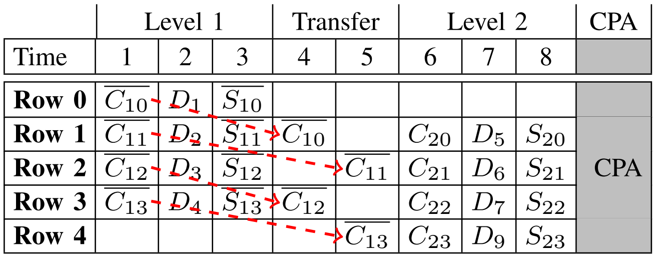


Figure 2.10: (a) Schematic and (b) dot notation representation for a 4 × 4 Wallace tree multiplier.

Breaking down the product computation further by mapping it to FA operations, a fast method for adding the partial products of a multiplication is to use Wallace/Dadda trees [48]. The schematic of 4×4 Wallace tree multiplier is shown in Fig. 2.10, annotated with the intermediate computations C_{ij} and S_{ij} for various values of i and j. At each level of the computation, we use a FA to reduce 3 (or sometimes 2) bits of the partial products to a sum bit and a carry bit that is propagated to the next column. For instance, in Level 1, A_2, B_1, and C_0 are added to produce S_{11} and C_{11}, which are added to similar terms in Level 2. Some FAs can be implemented as simpler half adders (HAs) since they have only two inputs. Each such FA/HA is shown by a red dotted rectangle containing 2 or 3 dots. The numbered label at the bottom left corner of the rectangle represents the CRAM row number that implements the FA operation. It can be seen that each column of the computation, which corresponds to a place significance, maps to the same CRAM row in each Level (e.g., the third-last column in Level 1 that adds A_2, B_1, and C_0 maps to CRAM row 2. The resultant sum S_{11} remains in that row and is added with other operands in Level 2, while the carry-out goes to the next row).



	Level 1			Transfer		Level 2			CPA
Time	1	2	3	4	5	6	7	8	
Row 0	$\overline{C_{10}}$	D_1	$\overline{S_{10}}$						CPA
Row 1	$\overline{C_{11}}$	D_2	$\overline{S_{11}}$	$\overline{C_{10}}$		C_{20}	D_5	S_{20}	
Row 2	$\overline{C_{12}}$	D_3	$\overline{S_{12}}$		$\overline{C_{11}}$	C_{21}	D_6	S_{21}	
Row 3	$\overline{C_{13}}$	D_4	$\overline{S_{13}}$	$\overline{C_{12}}$		C_{22}	D_7	S_{22}	
Row 4					$\overline{C_{13}}$	C_{23}	D_9	S_{23}	

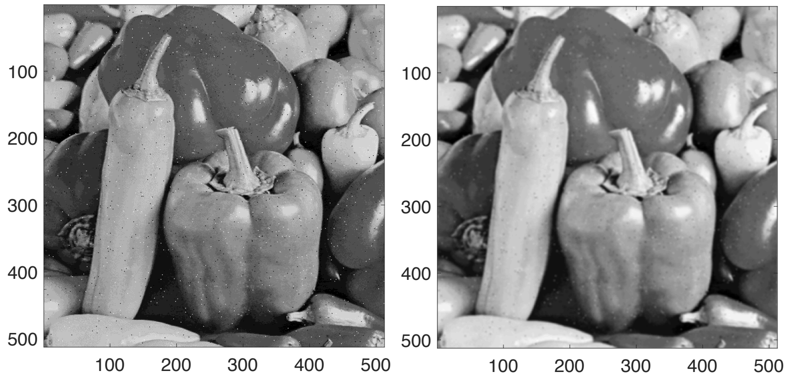
Figure 2.11: Scheduling table for the Wallace tree adder.

Another view of the scheduling of these computations is presented in Fig. 2.11. The implementation of each level requires 5 steps, and computations related to FAs within each level are performed in parallel. As in the case of the ripple carry adder, the first three steps for each FA at Level 1 involve computing the complement of the output carry, cloning the computed carry complement to another cell, and then computing the complement of the sum at $t = 1, 2, 3$, respectively. As before, bubble-pushing allows the inverted sum and carry outputs to be used directly in the next bit slice.

To begin the computations at Level 2, the computed carry values in row i must be sent to row $i + 1$, and this is accomplished at $t = 4, 5$. Note that due to the structure of the CRAM, this must be performed in two steps: when row i is connected to $i + 1$, we cannot simultaneously connect row $i + 1$ to $i + 2$, otherwise we create an inadvertent path from i to $i+2$. Therefore, transfers from all even-numbered rows to the next row occur in one time slot, and transfers from all odd-numbered rows in another. Three more steps are required to perform the FA computation at Level 2, which completes at $t = 8$, and the results then go to a CPA, implemented as in Section 2.4.

2.5 CRAM Applications

In this section, we present two applications of the CRAM: a two-dimensional (2D) convolution operation for image filtering using images and filters represented by multiple bits, and a binary neuromorphic inference engine for digit recognition.



(a)

(b)

Figure 2.12: Using the average (mean) filter to denoise an image: (a) the noisy image with numerous specks, and (b) the denoised version.

2.5.1 2D Convolution for Image Filtering

Convolution is a building block of many image processing applications, such as image filtering for sharpening and blurring. Fig. 2.12(a) shows an input image with 512×512 pixels that is convolved by a 3×3 filter to yield an output image with the same number of pixels, shown in Fig. 2.12(b). The output pixel in location (i, j) is computed as follows, as illustrated in Fig. 2.13(a):

$$O_{i,j} = \sum_{k=1}^{3} \sum_{l=1}^{3} f_{k,l} \cdot I_{i-k+2,j-l+2} \tag{2.10}$$

where f represents a matrix associated with the 3×3 filter, and I is the matrix of input pixels. The image I is represented using 4 bits, and the filter f uses two bits. Thus, each partial product $(f_{k,l} \cdot I_{i-k+2,j-l+2})$ has six bits.

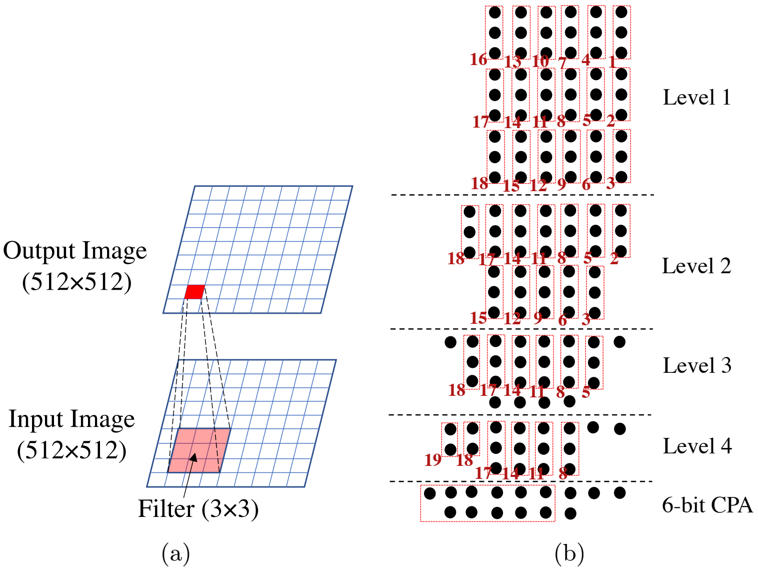


Figure 2.13: The implementation of convolution using CRAM: (a) a 512×512 image, with 4 bit per pixel, is filtered using a 3×3 filter, with 2 bits per word, and (b) the addition of nine six-bit partial products to compute the dot product that evaluates the output image pixel using a 4-level tree adder.

To compute $O(i,j)$, the result of the dot product representing a pixel of the output image, nine six-bit partial products are added together using a tree adder, as shown in Fig. 2.13(b). The tree adder has four levels, and uses a six-bit ripple carry adder at the final stage. As illustrated in the figure, the total number of rows required for the implementation of one dot product is 19. Similar to the adder and multiplier, it is easy to build a scheduling table for the implementation of the dot product: for conciseness, it is not shown here. The total number of steps for the implementation includes all steps for multiplication, additions within the rows, inter-row transfers, and the final CPA. The convolution for each output pixel can be computed in parallel.

2.5.2 A Neural Inference Engine

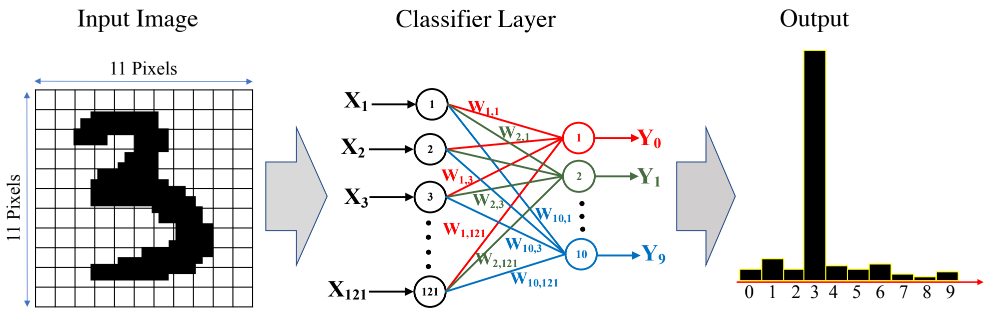
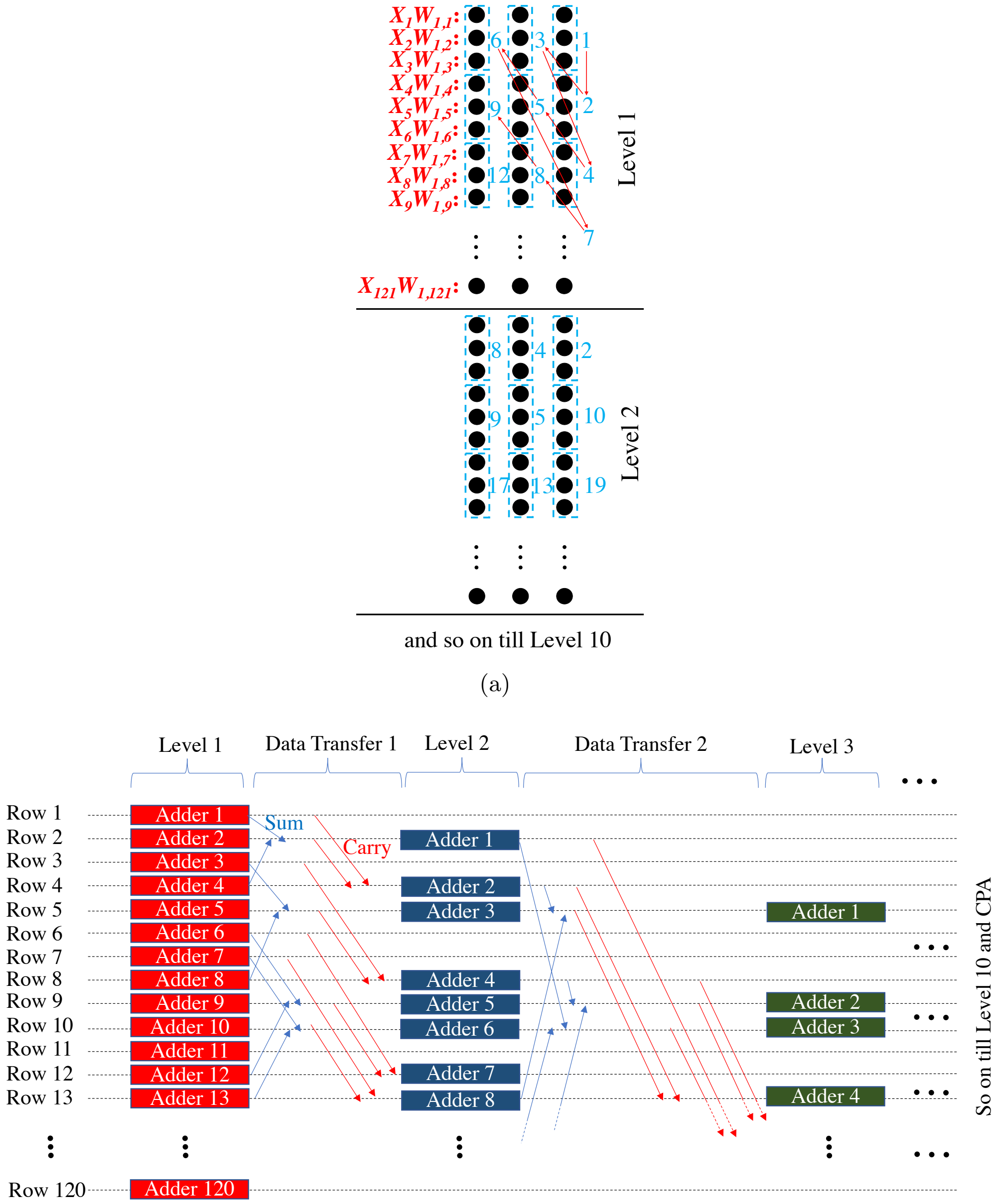


Figure 2.14: An inference engine for the digit recognition problem.

Using the building blocks described above, we show how the CRAM can be used to implement a neuromorphic inference engine for handwritten digit recognition using data from the MNIST database [49]. The neural network architecture from [50] (Fig. 2.14) is used to implement the recognition scheme. Each of the MNIST images is scaled to 11×11 as in [50], a transformation that maintains 91% recognition accuracy and reduces computation. Note that using the full image, or using a more complex neural engine with higher recognition accuracy, does not fundamentally change our approach or conclusions. This data is provided to the neural net with a set of one-bit inputs, $X_i, 1 \leq i \leq 121$, corresponding to each image bit, and fires one of 10 outputs, corresponding to the digits 0 through 9. In principle, this network can also be implemented using a larger number of bits, as in the previous application, rather than as a binary neural network; the unit operations for n bits have been described in Section 2.4. As in [50], the synaptic weights W_{ij} have three bits and are trained using supervised learning.



(a)

(b)

Figure 2.15: The implementation of each of the ten outputs, Y_i, of the inference engine, illustrating the (a) zigzag scheme for assigning addition operations to CRAM rows, and (b) the inter-row data transfers and the computation footprint of Y_i along the rows of the CRAM.

The outputs of the neural network are computed as the inner product:

$$Y_i = \sum_{j=1}^{121} W_{i,j} X_j \tag{2.11}$$

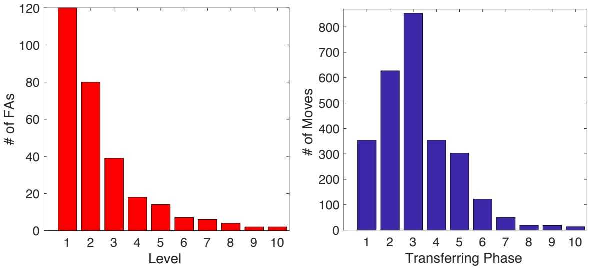
The inner product Y_i is computed as a sum of 121 partial products. Fig 2.15(a) uses the dot notation to represent the implementation of Y_i. Each partial product is a bitwise multiplication of a three-bit $W_{i,j}$ with a one-bit X_j, and this can be implemented using a bitwise AND operation. The resulting three-bit partial products, shown in each row at Level 1 in Fig. 2.15(a), are added together using a Wallace tree adder. Note that one can also use a Dadda tree adder or any other similar tree adder without changing the overall delay significantly. This is because the overall delay does not depend on the number of FAs in a level, as all FAs within a level act in parallel. As long as the number of tree levels (and the length of final ripple carry adder) is the same, the overall delay is quite similar.

Recall that each group of three dots in the figure is a FA computation performed in a row, after which the sum and carry are propagated to the next level of computation. In this case, the computation requires 9 levels. The choice of row assignments for the FA computations is critical in maintaining latency. In principle, it is possible to assign rows sequentially by column, e.g., the dots in the last column are assigned to rows 1 through $\lceil 121/3 \rceil$, then the second last column gets row $\lceil 121/3 \rceil + 1$ onwards, and so on. However, this would mean large communication latencies (e.g., the carry from row 1 may have to be sent to row 42 or higher). We use an approach that performs a zigzag assignment of rows, as shown in Fig. 2.15(a). After the computation in each row, the sum output is sent to the median row and the carry output to the largest row number. For example, the first three FAs in the right column are in rows 1, 2, and 4, according to the diagonal pattern. Their three sums are sent to the median row, row 2, and their carries are sent to the maximum row number, row 4. At Level 2, the same process repeats: the FAs in the first three groups, now in rows 2, 10, and 19, send their three sums to row 10 and three carrys to row 19, and so on. The Wallace tree has 9 levels in all, followed by a CPA stage.

Fig. 2.15(b) shows the footprint of computations, where the y-axis is the row number and the x-axis is time. Each colored block is an adder computation, which takes three steps if we use majority complementary logic (for advanced MTJ technology) or nine steps for NAND-based logic (using today's MTJ technology). As described above, once a computation in one level is complete, we transfer the sums to the median row number (as shown by blue arrows) and carrys to the largest row number (as shown by red arrows). For example, after Level 1, the Sum outputs for Rows 1 and 4 are transferred to Row 2, to set up the Level 2 computation that adds these to the Sum output produced in Row 2. Such inter-row transfers correspond to BUFFER operations that are carried out by activating the switches described in Section 2.2.3.

The span of rows involved in the computation shrinks from level to level. Fig. 2.16 shows the number of FA computations at each level of the Wallace tree and the number of inter-row data transfers involved before the beginning of the full adder computation at each level. Due to the scheme for moving sums and carrys across rows, as the computation proceeds, the span of rows that contain active data shrinks. For example, Level 1 involves all 120 rows, but fewer rows are involved at Level 2, starting from Row 2; at Level 3, the number reduces further, and the first active row is Row 5.

Our approach is shown on a binary neural network, a family of structures that has recently attracted great attention for low-energy deep learning on embedded platforms. However, the general concept and procedure for implementing our design can be applied to other neural inference engines, including multibit neural networks. In general, when the number of bits per pixel (for the same application) increases, the computation will employ unit operations with a greater number of bits (e.g., a tree adder with more



(a) (b)

Figure 2.16: (a) The distributions of the number of FAs in each level of Wallace tree. (b) The distribution of the total number of moves required in the data transfer phases.

levels and more FAs). This increases the number of steps and the size of the CRAM array for implementation. The fundamental operation in many neural computation models is a convolution of the form Equation (2.10) or a dot product of the form of Equation (2.11). As shown in Section 2.3, the CRAM architecture can perform the unit operations (addition and multiplication) for either. For example, the convolution layer in a convolutional neural network (CNN) involves dot product operations, and then a summation over the results of these dot products. Computations in other CNN layers, such as pooling and ReLU, also require simple arithmetic or Boolean operations that can be implemented on the CRAM substrate.

2.6 Evaluation and Results

We evaluate the performance of the CRAM for two applications: (a) performing 2D convolution to filter a 512×512 image, and (b) digit recognition, used to analyze 10,000 handwritten digit images from the MNIST database. In both applications, the execution time and energy of the CRAM are compared with those of a near-memory processing (NMP) system, where a processor is available at the edge of the memory array. We do not explicitly show comparisons between NMP and processor-based computing, where the data is taken from memory to a processor or coprocessor for computation, and the results are transported: it is well-documented [1,16,51] that this method is vastly inferior to the NMP approach due to the communication bottleneck described in Section 5.1. For example, [51] reports a 6.5× improvement through the use of NMP, as compared to processor-based computing. Note that this communication overhead limits the effectiveness of any processor or coprocessor that requires communication to and from memory, including specialized accelerator cores (e.g., neuromorphic units or GPUs).

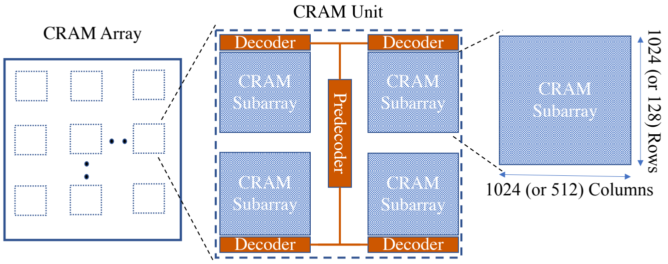


Figure 2.17: Each CRAM unit includes four CRAM subarrays and one predecoder. A predecoder block is at the center of the CRAM unit, and fans out to four CRAM column decoders.

The organization of the CRAM array is shown in Fig. 2.17. For the 2D convolution application, a 256Mb [512Mb] CRAM array is enough to compute all output pixels of a 512×512 image with 4 bits per pixel in parallel using the advanced [today's] MTJ device. For the digit recognition application, we require a 1Gb memory, where each image can be processed in parallel within a subarray. The overall array is divided into subarrays as shown in the figure. The operations in the CRAM array are scheduled by activating the appropriate LBL and BSL lines. In memory mode, the predecoder and decoder modules drive the selection of WL (see Fig. 2.1), while in logic mode, they drive the selection of LBL. The predecoder at the center of the CRAM unit fans out to a set of decoders in our evaluations: here, we show four decoders, but if a larger number of subarrays is used, this number can be different.

To calculate the energy and delay of the CRAM system, we considered the energy and delay components in both peripheral circuitry and the CRAM array. To determine the impact of the size of CRAM on execution time and energy, we considered two cases for the size of CRAM subarrays: 1024 rows × 1024 columns, and 128 rows × 512 columns.

2.6.1 Execution Time

CRAM: We assume that the data is placed in the appropriate CRAM array location. The execution time, t_{CRAM}, is:

$$t_{CRAM} = t_{MTJ} + t_{Dr}, \tag{2.12}$$

where t_{MTJ} and t_{Dr} are delay related to computations in the MTJ array and in the bitline drivers of Fig. 2.4, respectively. The total array delay is dominated by the MTJ delay,

$$t_{MTJ} = N_{step} t_{wr}, \tag{2.13}$$

where N_{step} and t_{wr} are, respectively, the number of computation steps and the MTJ write time per computation. Here,

$$N_{step} = N_{Mul} + N_L N_{FA} + \textstyle\sum_{i=1}^{N_L} I_{i \longrightarrow i+1} + t_{CPA} \tag{2.14}$$

where N_{Mul} is the number of steps required to generate partial products for the first level of the tree adder. The second term indicates total number of intrarow computation steps required for the implementation of the neural network, where N_L the number of levels in the implementation tree adder, and N_{FA} the number of steps for the implementation of a FA. The third term corresponds to the total number of steps for transferring data between rows of the CRAM array: at each level i of the tree, the number of such transfers is denoted by $I_{i \longrightarrow i+1}$. Finally, t_{CPA} is the time required for the carry propagation addition step at the end of the Wallace tree computations. The preset adds no execution time overhead and can be performed either during the write operation when CRAM is in the memory mode, or online during the computation when CRAM is in the logic mode. In the latter case, the output MTJs are preset in parallel with the logic computation of the previous step, adding no overhead to the compute time. During the logic operation LBLs, and BSLs are engaged in computation, and current flows through LLs (see Fig. 2.1 and Fig. 2.2). Simultaneously, one can also write the preset value for the next step, as only MBL and BSL (of another column) are involved in the writing operation, and there is no overlap between current path related to computation and that to output preset.

For the 2D convolution application, we have:

- From Section 2.5.1, $N_{Mul} = 9$ and $N_L = 4$.
- Based on Section 2.3.3, $N_{FA} = 3$ using the $\overline{\text{MAJ}}$ gates, with bubble-pushing, in advanced MTJ technologies, and $N_{FA} = 9$ using NAND-based logic in today's technology (where $\overline{\text{MAJ}}$ gates do not provide sufficient noise margin, as shown in Section 2.3.2).
- We count all number of steps in the inter-row communication phases, and find that $\sum_{i=1}^{N_L} I_{i \longrightarrow i+1} = 14$.
- Extending the argument from the four-bit adder in Section 2.3.3, $t_{CPA} = 13$ for the six-bit adder.

From (2.14), we obtain $N_{step} = 48$ (for the advanced CRAM with $\overline{\text{MAJ3}}$ based logic) and $N_{step} = 72$ columns (for today's CRAM with NAND-based logic). Thus, the computation for each pixel of the output image requires an array of 19 rows (Section 2.5.1) and N_{step} columns. By rounding the column counts to the nearest power of 2, and considering all 512×512 pixels of the output image, a CRAM array size of 256Mb is required for the computation on the advanced MTJ; the corresponding number for today's MTJ is 512Mb.

For the digit recognition application:

- From Section 2.5, $N_{Mul} = 6$, and $N_L = 10$.
- As before, $N_{FA} = 3$ using the advanced CRAM, and $N_{FA} = 9$ using today's technology.
- The number of steps in the inter-row communication phases is determined to be $\sum_{i=1}^{N_L} I_{i \longrightarrow i+1} = 247$.
- From Section 2.3.3, $t_{CPA} = 9$ for the four-bit adder.

Therefore, $N_{step} = 292$ for the advanced MTJ technology (using $\overline{\text{MAJ}}$ logic), and $N_{step} = 352$ using today's MTJs (using NAND logic). The computation of each image requires an array of 121 rows (corresponding to the partial products) times 10 outputs, and 292 or 352 columns (corresponding to the steps in the computation), depending on the type of MTJ used. Therefore, rounding 292 (or 352) to the nearest higher power of 2, in a 1024 × 1024 memory subarray, we can fit 18 images (9 images along the rows and 2 images along the columns). The entire set of 10,000 images thus requires 10,000/18 = 556 such arrays; rounding this up to 1024, we see that we require 1024 such subarrays, providing a total memory size of 1Gb, as listed earlier.

To incorporate the delay of bitline drivers in the CRAM, an overhead delay estimated in each step by considering the delay components of a DRAM array with the same physical size. We use the parameters and models in the NVSim memory simulator [52] at 10nm and 45nm to consider a subarray of this size and use the underlying parameters to obtain these delays. We tailor the NVSIM models to the specifics of the CRAM computation. Specifically, in logic mode, each computation step requires LBLs to be driven, similar to driving wordlines, but does not require bitline or sense amplifier circuitry. The load seen by the column drivers in logic mode can be modeled in the same way as the load seen by row drivers in memory mode: instead of driving a wordline transistor as in memory mode, the LBL drives the access transistor that connects a cell to the LL. For each step of computation, we calculate the sum of the delay of the decoder and predecoder using similar models as NVSim, and this value is multiplied by N_{step} to find the total overhead corresponding to t_{Dr}. The size of the bit-cell is also altered to reflect the increased size of the CRAM bit cell over that for an STT-MRAM cell.

NMP System: The near-memory processing (NMP) system takes data from the memory to a processor and performs its computation outside the memory system. We assume that the operation is based on a DRAM structure, with better performance characteristics than a spintronic memory. For the digit recognition application, to process all 10K images of the MNIST database, 10,000 images, each of size 121 bits, must be fetched from DRAM, for computations in the near-memory processor. The delay for this scenario is estimated using CACTI [53]. A similar approach is used for the 2D convolution application.

Computing one of the outputs, Y_i, of the neural net requires 121 MAC operations. To find the processing time of each block of data, it is assumed that the processor uses instruction pipelining technique and that it can perform the multiply-accumulator (MAC) operation in one clock cycle. In case multiple processing units are available on the processor, we appropriately scale the execution time by the number of processors. We pessimistically assume maximum parallelism, where each fetched image is processed in parallel, and the level of parallelism is only limited by the data rate from memory. The clock frequency of the processor is 1GHz, but due to the assumption above, the precise computing speed of the processor does not affect the evaluation of NMP execution time.

2.6.2 Energy

Analogous to delay, the CRAM energy is computed as:

$$E_{CRAM} = E_{MTJ} + E_{Dr} \tag{2.15}$$

where E_{MTJ} and E_{Dr} are the energy related to computations in the MTJ array and for the bitline drivers, respectively. The energy in the MTJ array is given by

$$E_{MTJ} = E_{Preset} + E_{Mul} + E_{row} + E_{transfer} + E_{CPA} \tag{2.16}$$

in which E_{Preset} is the preset energy before the logic operation starts; E_{Mul} is the energy for multiplication to produce partial products in Level 1 of the tree adders; E_{row} is the energy for intrarow computation, and can be obtained by enumerating all FAs working in parallel in the 9 levels of the implementation trees; $E_{transfer}$ is the energy for transferring data across rows between various levels of computation, and can be obtained by enumerating all inter-row moves and multiplying the count by the energy of BUFFER gate; E_{CPA} is the energy for the implementation of the final ripple carry adders. For the advanced MTJ technology using $\overline{\text{MAJ3}}$ gates, Eq. (2.16) can be rewritten as follows (a similar equation can be derived for today's MTJ):

$$\begin{aligned} E_{MTJ} =& N_{pre}E_{pre} + N_{NOT}E_{NOT} + N_{BUF}E_{BUF} + \\ & N_{\overline{MAJ3}}E_{\overline{MAJ3}} + N_{\overline{MAJ5}}E_{\overline{MAJ5}} \end{aligned} \tag{2.17}$$

Here, E_{pre} is the energy for preset the output of one gate; E_g and N_g are the energy for the implementation of a single gate g and the number of gates of type g, $g \in$ {NOT, BUFFER, $\overline{\text{MAJ3}}$, $\overline{\text{MAJ5}}$}. Note that N_{pre} is equal to the sum of counts of all gates, as we need to preset the outputs of all gates. colorredAs an example, the energy values and counts for gates and the output preset for the digit recognition application using advanced CRAM are listed in Table 2.3.

The value of driver energy, E_{Dr}, for the CRAM is estimated using NVSim, using analogous analysis techniques as for the delay computation. Since multiple columns may be driven in each step, we multiply the energy cost of driving each column by N_{eff}, the average number of columns driven in any part of the computation. The energy within each CRAM unit is the sum of energy of four CRAM subarrays and one decoder. This value is multiplied by N_{step} to obtain the total overhead corresponding to E_{Dr}.

For the near memory processing system, the energy consists of two components: (i) memory access energy and (ii) computation energy. The estimated cost for accessing 256 bits of the operand from memory is estimated using [1], normalized to CACTI.

Table 2.3: The energy cost for various CRAM gate types and preset operations under the advanced MTJ technology.

Gate	NOT	BUFFER	$\overline{\text{MAJ3}}$	$\overline{\text{MAJ5}}$	Preset
Energy/gate(aJ)	30.7	73.8	7.6	6.3	26.1
Count($\times 10^5$)	365	3017	657	294	4333

2.6.3 Comparison between CRAM and NMP

The results for execution time and energy for CRAM (at 10nm and 45nm) and NMP (at 16nm and 45nm) are evaluated for both applications (10nm data for CMOS/NMP was not available).

The evaluation result for the 2D convolution application is listed in Table 2.4. Based on the result, today's CRAM is 620× faster, and 23× more energy efficient than the NMP system. The advanced CRAM is 1500× faster, and 750× more efficient than a NMP system.

For the digit recognition application, the results for execution time and energy for CRAM (at 10nm and 45nm) and NMP (at 16nm and 45nm) are shown in Table 2.5 (10nm data for CMOS/NMP was not available). The value of E_{MTJ} is 53.8μJ for today's MTJ technology, and 35.4nJ for advanced MTJs, three orders of magnitude

Table 2.4: Comparison between the execution time, t, and energy, E, in CRAM and NMP based computations for the 2D convolution application. The size of the CRAM subarrays in this evaluation is 128×128.

	CRAM		NMP	
	10nm	45nm	16nm	45nm
t	54.0ns	231.2ns	84.3μs	144.4μs
E	252.1nJ	16.5μJ	189.2μJ	388.6μmJ

lower. While the driver energy also reduces from 45nm to 10nm, the reduction is more modest. As a result, the energy for advanced MTJs is dominated by the driver delay.

The improvements shown in the table can be attributed to (a) high locality of the operations and (b) large amounts of parallelism as each row computes in parallel. We see that

- For the 1024×1024 subarray, the CRAM energy is about 40× better than NMP at 45nm, and improves to over 2500× lower at 10nm. The execution time is 1400× better at 45nm, and about 1700× better at 10nm.
- The execution time [energy] for the 45nm CRAM are, respectively, over 500× [20×] better than 16nm NMP.
- The 10nm CRAM execution time [energy] is over 3× [80×] better than the 45nm CRAM.
- Further improvements are seen using the smaller subarray. The energy overhead associated with smaller subarrays is small at 45nm, but is magnified at 10nm, where the driver energy dominates the subarray energy.

The distributions of energy and delay for the CRAM, both using today's MTJs and advanced MTJs, with subarrays of 1024 rows × 1024 columns, and 128 rows × 512 columns, are shown in Fig. 2.18 and Fig. 2.19, respectively. For the 1024 × 1024 case, under today's technology, the MTJ array in the CRAM consumes a dominant component

Table 2.5: Comparison between the execution time, t, and energy, E, in CRAM and NMP based computations for neuromorphic digit recognition.

	CRAM				NMP	
	1024 × 1024		128 × 512			
	10nm	45nm	10nm	45nm	16nm	45nm
t	434ns	1381ns	338ns	1105ns	0.74ms	1.96ms
E	0.49μJ	60.3μJ	0.75μJ	63.8μJ	1.27mJ	2.57 mJ

of the energy. However, for advanced MTJs, due to the greatly improved energy of future MTJs, the energy bottleneck will be in the driver circuitry. By decreasing the size of the subarray to 128 rows×512 columns, the total execution time decreases due to a reduction in the driver circuitry delay. As a result, the execution time is dominated more strongly by the MTJ array. However, the driver circuitry plays a slightly more prominent role in determining the energy than for the larger subarray in Fig. 2.18. Thus, tradeoffs between energy and delay can be obtained by altering subarray sizes.

These result clearly shows that for both applications, CRAM outperforms the NMP system in both energy and execution time. In the NMP system, it is necessary to fetch the data from memory and process it in processor units. Even with the maximum level of parallelism in NMP by using multiple processor units, and exploiting hidden latency techniques, the delay overhead of fetching data to the NMP at the edge of the memory is a major bottleneck. In contrast, the CRAM does not face this delay penalty. Moreover, the CRAM computation model enables a very high degree of parallelism as each row can perform its computations independently.

For example, in the 2D convolution application, all dot products generating output pixels can be computed in parallel. In contrast, the NMP system faces a serial bottleneck in the way that data is fetched from the memory. Moreover, the energy cost of the cost of data transfers cannot be hidden in the NMP system as data must be taken along long lines to the edge of memory. In contrast, all communication in the CRAM is inherently

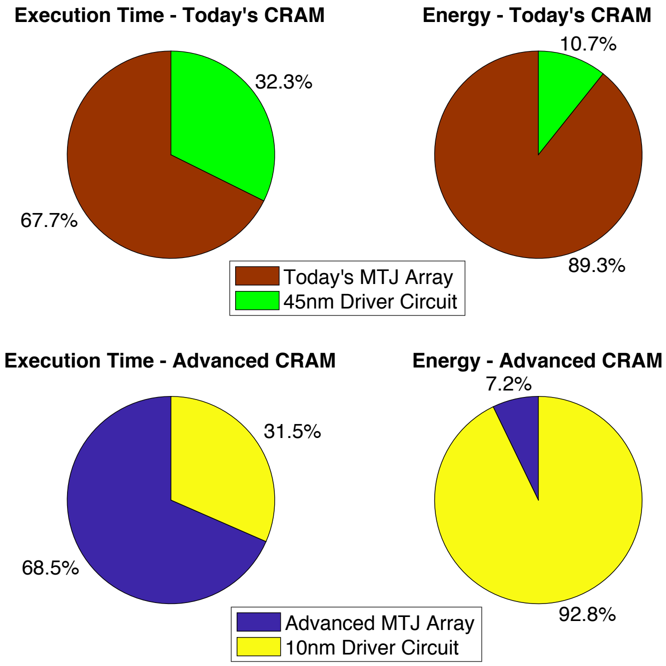


Figure 2.18: Distribution of energy and delay of the driver and CRAM array for CRAM with the subarray size of 1024 × 1024.

local within the subarray, providing large energy savings.

2.7 Related Work

Methods for addressing the communication bottleneck through distributed processing of data at the source have been proposed in [54, 55]. Such techniques feature a rich design space, which spans full-fledged processors [55, 56] and co-processors residing in memory [57, 58]. However, until recently, the promise of these approaches could not be translated to designs due to the incompatibility of the state-of-the-art logic and memory technologies.

This changed somewhat with the emergence of 3D-stacked architectures [59, 60], where a processor is placed next to the memory stack, has enabled the emergence of

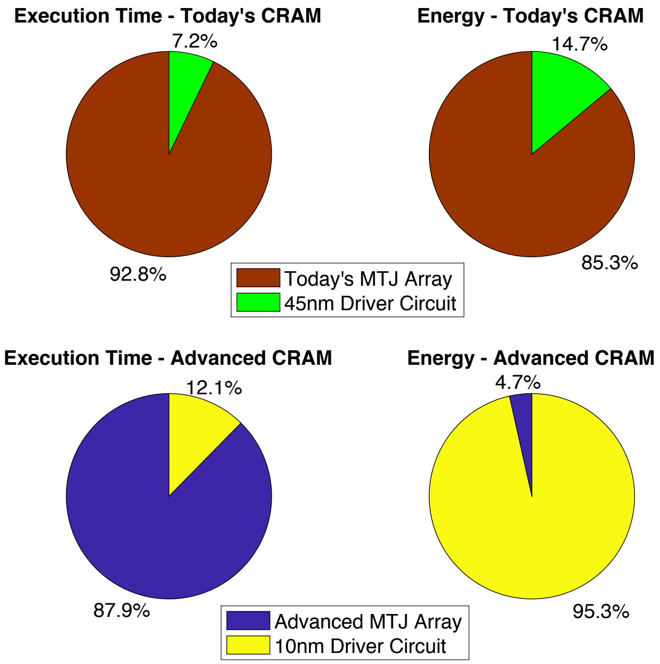


Figure 2.19: Distribution of energy and delay of the driver and CRAM array for CRAM with subarray size of 128 × 512.

several approaches for near-memory computing [61–63]. However, building true in-memory computing has been difficult. In CMOS-based technologies: the computing engine is overconstrained as it must use the same technology as memory, and typically, methods that are efficient for computation may not be so for memory. As a result, techniques that attempt in-memory computation must necessarily draw the data out to the periphery of the memory array, e.g., to a sense amplifier or auxiliary computational unit, to perform the computation and then write the result back to memory as needed. There are several examples of such platforms. The work in [64] performs search operations for content-addressable memory functionalities, which need no write-back but are less general than full in-memory computing; methods in [65] place a computational unit at the edge of memory; logic functionalities in [66] perform bitwise operations through the sense amplifier.

Post-CMOS technologies open the door to new architectures. The method in [67] presents a logic-in-memory platform that combines magnetic tunneling junctions (MTJs) with MOS transistors, and embeds computing elements within a memory array. However, this breaks the regularity of the array so that while it is efficient for computation, it may not be ideal for use as a memory module. SPINDLE [68], a spintronics-based deep learning engine proposes a tiered architecture of processing elements with a neural computing core and memory scratchpad at the edge, communicating with off-chip memory. The Pinatubo [69] processing-in-memory architecture performs bulk bitwise operations through redesigned read circuitry that performs computations at the periphery of a phase change memory array. A spintronics-based solution in [70] proposes a spin-transfer torque magnetic random access memory (STT-MRAM) approach that also performs bitwise computation at the periphery of the array by modifying the peripheral circuitry in a standard STT-MRAM module. Unlike CRAM, these methods perform computation at the edge of the memory array. Another architecture [71] builds a four-terminal domain wall device based on the spin-Hall effect, but incurs a significant area overhead. A memristor-based approach [72] shows the ability to perform logic functions in an array, but does not show larger applications.

2.8 Conclusion

This chapter presents a detailed view of how the CRAM in-memory computation platform can be designed, optimized, and utilized. As opposed to many of the approaches proposed so far to solve the memory bottleneck by bringing processing closer to memory, CRAM implements a true in-memory computing paradigm that performs logic operations within the memory array. Methods for implementing specific logic functions have been presented and have been used to perform basic arithmetic operations, namely, adders and multipliers. At the application level, the problems of 2D convolution on multibit numbers, and an inference engine for binary neuromorphic digit recognition, have been mapped to the CRAM. An evaluation of these methods shows that for the task of evaluating the entire MNIST benchmark suite, the CRAM achieves improvements of over three orders of magnitude in the execution time. For today's MTJ technology, improvements of about 40× in the energy are seen, a figure that improves to $> 2500\times$ for future generations of MTJs.

Chapter 3

Analyzing the Effects of Interconnect Parasitics in the STT CRAM In-Memory Computational Platform

3.1 Introduction

This chapter studies of the impact of interconnect parasitics in the spin transfer torque (STT) computational random access memory (CRAM), a true in-memory processing platform [2,17,18]. Only a few prior works [73,74] have attempted to incorporate the parasitic effects of interconnects in their analysis on in-memory computing, but their models did not consider all contributing factors based on realistic layout considerations.

As described in chapter 2, CRAM uses a small modification to the high-endurance MTJ-based [75] memory cell to enable true in-memory logic operations. In the CRAM, the per unit resistances of wires that carry the current are significantly smaller than the

MTJ resistances. This can falsely lead to this conclusion that the interconnect parasitic effects are negligible. In reality, the accumulative resistances of these wires can be significant. This work develops an analytical method based on layout considerations that is used to study the effects of design parameters on parasitics and performance, in order to build a robust CRAM design. The method considers multiple contributing factors simultaneously, e.g., reducing the access transistor resistance can potentially enhance the performance, but it also require larger size of unit cells, which increases the area of the array and increases interconnect lengths. Together, these effects can harm performance.

In Section 5.2, we provide an overview for derivation of bias voltages of different gates considering resistances of access transistors. We then motivate the problem in Section 3.3. Next, in Section 3.4, we develop a layout model for the CRAM in a FinFET technology, considering the both the cell level and array level while also specifying metal layer usage. We develop models for the impact of parasitics in Section 3.5, evaluate the results of our analysis in Section 5.6, and conclude in Section 5.7.

3.2 Overview for Derivation of Bias Voltages of Different Gates Considering Resistances of Access Transistors

The core storage unit in an STT-CRAM is the STT-MTJ, which consists of a fixed layer, with a fixed magnetization orientation, and a free layer whose magnetization can be in one of two possible states – parallel (P) and anti-parallel (AP) [39]. The two states have different electrical resistances: the parallel state resistance, $R_P < R_{AP}$, the anti-parallel state resistance. We denote the P and AP states as logic 0 and 1, respectively. The MTJ state can be altered by passing a critical current of magnitude I_c through it in the appropriate direction.

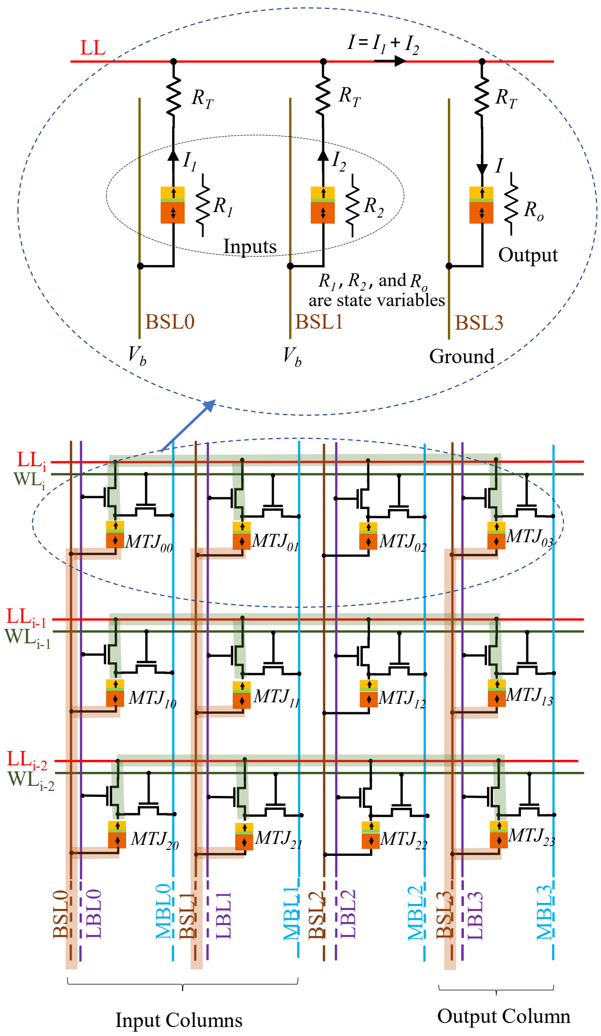


Figure 3.1: Structure of STT-CRAM array, highlighting the current paths during a logic operation with two inputs and one output.

Fig. 3.1 shows the structure of an STT-CRAM array, which uses a 2T1MTJ bit-cell [2, 17, 38]. The configuration of the array is very similar to that of a standard 1T1MTJ STT-MRAM, and as in the case of the STT-MRAM, each bit-cell is addressed using a memory word line (WL). The additional transistor in the STT-CRAM is used for logic operations, and is turned on by selecting the corresponding logic bit line (LBL).

The array can thus function in memory or logic mode. In memory mode, by applying an appropriate voltage to the bit select lines (BSLs) of the input bit-cells and grounding the BSL of the output, a state-dependent current, whose value depends on the resistance of MTJs and transistor resistance (R_T), flows through the output MTJ. If this current exceeds I_c, the output state (the resistance of the output) is altered; otherwise, it remains the same.

Different logic functions can be realized in the STT-CRAM by altering two parameters [2, 17, 38]: (a) the bias voltage (V_b) applied to the BSLs of the input MTJs, and (b) the output preset state. In chapter 2.3, for each gate a range for V_b without considering non-idealities is calculated.

Next, we show how chapter 2.3 derives an allowable range for V_b for a 2-input AND gate, ignoring the parasitic effects of lines and transistors. An AND gate in each row can be realized by the configuration shown in the upper side of Fig. 3.1, which highlights the path of current through the MTJs. Current I can be calculated by dividing V_b by the equivalent resistance $((R_1 + R_T)||(R_2 + R_T)) + R_o$, where "$||$" represents the equivalent resistance of parallel resistors. If $R_A = R_P + R_T$, $R_B = R_{AP} + R_T$, the current for each input state is:

$$I_{00} = V_b/(0.5R_A + R_B) \qquad I_{11} = 2V_b/(3R_B)$$
$$I_{01} = I_{10} = V_b/((R_A||R_B) + R_B)$$

Since $R_P < R_{AP}$, $R_A < R_B$, implying that

$$I_{11} < I_{01} = I_{10} < I_{00}. \tag{3.1}$$

The output MTJ is preset to logic 1. For an AND gate, $I_{01} = I_{10} > I_c$, switching the output state from 1 to 0. From (3.1),

$$V_b > (R_A||R_B + R_B)I_c \tag{3.2}$$

On the other hand, V_b cannot be too large; if it is, the output is switched regardless of the states of inputs, i.e., we must ensure that I_{11} must not be larger than I_c, i.e., from

(3.1),

$$V_b < 3R_BI_c/2 \tag{3.3}$$

Considering these two constraints, we can present a bias voltage range for the AND gate. The voltage ranges and preset values for other logic functions can be obtained in a similar manner and are summarized in Table 3.1. The precise range of V_b is technology-dependent. To account for anticipated advances in spintronics [6], this work considers MTJ specifications in today's technology and an advanced near-future technology [2]. In the rest of the chapter, we use today's and advanced MTJ parameters listed in Table B.1 (see appendix B) for our calculations and evaluations.

Table 3.1: Bias voltage ranges and output preset values [2]

Gate (Preset)	V_{min} = Minimum V_b	V_{max} = Maximum V_b
BUFFER(1)	$(R_A + R_B)I_c$	$2R_BI_c$
NOT(0)	$2R_AI_c$	$(R_A + R_B)I_c$
AND(1)	$(R_A\|\|R_B + R_B)I_c$	$1.5R_BI_c$
NAND(0)	$(R_A\|\|R_B + R_A)I_c$	$(0.5R_B + R_A)I_c$
OR(1)	$(0.5R_A + R_B)I_c$	$(R_A\|\|R_B) + R_B)I_c$
NOR(0)	$1.5R_AI_c$	$(R_A\|\|R_B + R_A)I_c$
MAJ3(1)	$(0.5R_A\|\|R_B + R_B)I_c$	$(R_A\|\|0.5R_B + R_B)I_c$
$\overline{\text{MAJ3}}$(0)	$(0.5R_A\|\|R_B + R_A)I_c$	$(R_A\|\|0.5R_B + R_A)I_c$
MAJ5(1)	$((1/3)R_A\|\|0.5R_B + R_B)I_c$	$(0.5R_A\|\|(1/3)R_B + R_B)I_c$
$\overline{\text{MAJ5}}$(0)	$((1/3)R_A\|\|0.5R_B + R_A)I_c$	$(0.5R_P\|\|(1/3)R_B + R_A)I_c$

3.3 Impact of Wire Parasitics

To show the impact of parasitics in a CRAM array, we consider a scenario where each row of the CRAM performs a BUFFER operation between Column 1 and Column 10. The CRAM array can be built using either today's technology (today's CRAM) or using advanced technology (advanced CRAM). An electrical model of the current path is shown in Fig. 3.2: the bias voltage is appied between BSL 1 and BSL 10, and in each row, the current path goes through input and output MTJs, two access transistors and a segment of LL. The model includes parasitic capacitances associated with each line segment and as well as the transistor resistance. For the motivational example in Section 3.3, built around Fig. 3.2, the transistor N_{fin} and N_{finger} are specific to the example and lead to the computed values of W_{cell} and L_{cell}. The remaining parameters are used throughout the rest of the chapter. The parameters used in the following motivational example, are listed for both today's and advanced CRAMs in the Table 3.2.

Table 3.2: Parameters in the motivational example

Parameter	Description	Today's CRAM	Advanced CRAM
N_{fin}	Number of fins	4	2
N_{finger}	Number of fingers	8	4
W_{cell}	Cell width	189nm	135nm
L_{cell}	Cell length	1323nm	675nm
R_T	Transistor resistance	0.178Ω	0.713Ω
d_{column}	Input-output distance	9	9
R_x	LL segment resistance	33.300Ω	25.100Ω
R_y	BSL segment resistance	0.026Ω	0.032Ω
C_x	LL segment capacitance	11.240fF	4.223fF
C_y	BSL segment capacitance	816aF	341aF
C_{gd}	Transistor g-d capacitance	320aF	320aF
C_{sg}	Transistor g-s capacitance	330aF	330aF
V_b	Applied bias voltage	670mV	96AmV

In the absence of wire parasitics and process variations, the bias voltage range for the implementation of a BUFFER gate can be obtained from Table 3.1. In [2], we reported numerical values of bias voltages for different gates. For the BUFFER gate, under today's technology and advanced technology the voltage ranges are 552mV to 788mV and 70mV to 121mV, respectively [2]. To maximize noise margin, we would choose the mid-point of the interval, $V_b = 670$mV for today's CRAM and $V_b = 96$mV for advanced CRAM, to implement the BUFFER. This voltage is applied through drivers at the edge of the CRAM array, where each driver has a resistance R_D. The CRAM rows in Fig. 3.2 are numbered from 1 (the nearest row to the driver) to N_{row} (the farthest row). When parasitics are accounted for, it can be seen that the path to row 1 encounters the fewest parasitics, and that to row N_{row} the most, due to IR drop along the line. Thus, the

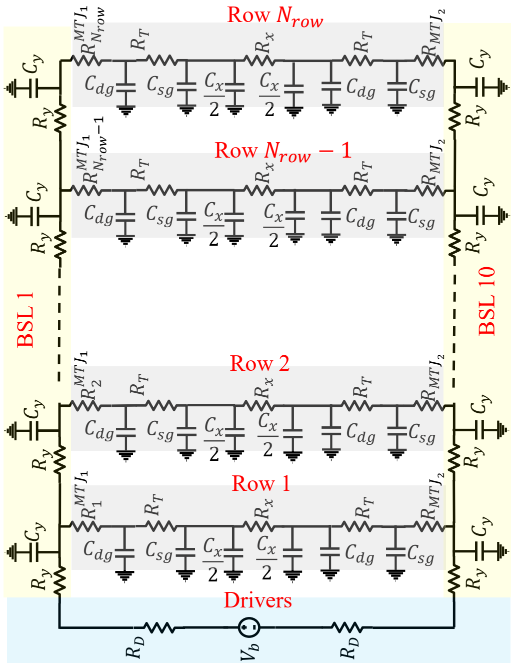


Figure 3.2: Circuit model of the current path for the implementation of BUFFER gates in CRAMs of various sizes.

voltage $V_b = 670$mV in today's CRAM (and $V_b = 96$mV in advanced CRAM) may not be significantly changed as it reaches the first row, but the voltage at row N_{row} may be significantly degraded.

Given the fixed voltage range (552–788mV for today's CRAM and 70–121mV for advanced CRAM) within which the BUFFER operates correctly, the entire array will operate correctly when the BSL voltage for Row 1 is at the maximum V_b value, and the BSL voltage for Row i is at the minimum allowable V_b for the BUFFER. Thus, the maximum allowable voltage drop is the difference between the maximum and minimum V_b, i.e., 226mV in today's CRAM and 51mV in advanced CRAM for BUFFER, and a similarly calculated value from Table 3.1 for any other gate. In practice, the drop must be even smaller to allow for noise margins.

We consider six CRAM array configurations, each with a different number of rows, and use Table 3.3 to show the degradation of V_b as it reaches the farthest row for each of these configurations. If each row performs an identical operation in the worst case, it should carry an equal current, I_{row}, and the total voltage drop to the last row is

$$nI_{row}R_y + (n-1)I_{row}R_y + \cdots I_{row}R_y = n(n+1)/2I_{row}R_y$$

i.e., the IR drop increases quadratically with the number of rows. For the 64-row array (in both CRAMs), Table 3.3 shows that this IR drop is not large, but for arrays with 256 rows and larger in today's CRAM (and for 2048 rows in advanced CRAM), the IR drop is a significant fraction of V_b. The quadratic trend is seen between the first few rows, but the trend becomes subquadratic in the last few rows: this is due to the high voltage drop, the current supplied to that row is significantly less than supplied to the first row (e.g., for $N_{row} = 2024$, the voltage level at the last row is about 20mV for today's CRAM (and 30mV for advanced CRAMA), far less than V_{min} for a buffer). This invalidates the assumption in the above derivation that an equal current of I_{row} is supplied to each row.

Next, we compute the impact of RC parasitics on the CRAM delay. Defining the transition time as the time to 90% of the final value, Table 3.3 shows that this time is negligible in comparison to the ns-range MTJ switching time. Thus, wire parasitics do not impact the delay, but only the IR drop.

3.4 Layout Modeling

The key parameters that affect the IR drop are the:

- number of rows, for reasons illustrated in Table 3.3.
- transistor resistance, R_T, which is in series with the MTJ resistance; a higher value reduces the noise margin (the value of R_T can be reduced by increasing the transistor width, which may increase the cell area).

Table 3.3: IR drop differential between the BSL voltage for the first row and the last row, and the RC delay of the transition

	Today's CRAM		Advanced CRAM	
# rows	IR drop	RC delay	IR drop	RC delay
64	5.8mV	87fs	0.2mV	52fs
128	22.4mV	346fs	0.6mV	210fs
256	82.3mV	1.3ps	2.3mV	798fs
512	250.0mV	4.1ps	8.5mV	3.1ps
1024	507.9mV	10.3ps	38.0mV	10.6ps
2048	650.3mV	18.4ps	75.6mV	26.1ps

- cell area, $A_{cell} = W_{cell}L_{cell}$ (where W_{cell} and L_{cell} are the cell width and length, respectively), which impacts the BSL and LL lengths, thus affecting IR drop.
- cell aspect ratio, $AR_{cell} = \frac{W_{cell}}{L_{cell}}$, which determines the BSL and LL lengths (a larger AR_{cell} makes the BSLs longer, causing increasing parasitics on them, while shortening LLs and reducing their parasitics).
- configuration of BSLs and LLs, whose resistance can be reduced using a multi-metal layer structure, and whose length and width depend on other parameter choices.

3.4.1 Layout of a CRAM Cell

Designing an STT-MRAM with a FinFET access transistor can reduce the cell area and improve leakage power and reliability [76]. The design in this work is based on ASAP 7nm Predictive PDK [7]. Fig. 3.3(a) shows the layout of a FinFET using a single fin. By applying the proper voltage to the gate (G), the current flows from drain (D) to the source (S) through the fin. By increasing the number of fins, the ON current increases and the drain-source resistance of the of FinFET decreases, at the cost of an increase in FinFET area. Such transistors can be drawn in multiple ways: a 4× FinFET is shown in Fig. 3.3(b) with 4 fins, or alternatively, in Fig. 3.3(c), using two fingers with 2 fins each. For the same transistor ON resistance, one can change the aspect ratio of the FinFET device by varying the numbers of fins and fingers.

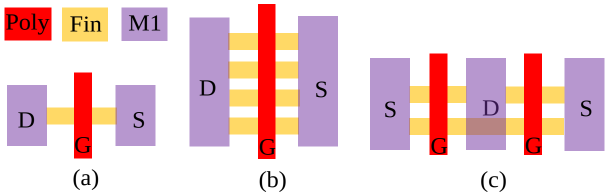


Figure 3.3: The layout of FinFET devices with (a) 1 fin, 1 finger, (b) 4 fins, 1 finger, and (c) 2 fins, 2 fingers.

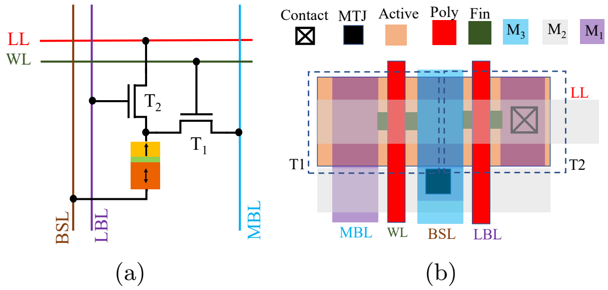


Figure 3.4: CRAM cell: (a) schematic, and (b) layout.

Fig. 3.4 shows the schematic and layout of the CRAM cell using a 1-fin 1-finger FinFET. The source of T1 is connected to MBL and M1 is allocated for MBL routing; the poly in T1 is used for WL; the drain of T1 is connected to the MTJ, which is physically placed between M2 and M3. For T2, the drain is connected to LL, the poly is used locally for LBL, and the source is connected to the MTJ. In the layout, a horizontal M2 stripe is used for LL routing, and a vertical M3 stripe is used for BSL. Larger transistor sizes can be achieved by using multiple fins and fingers for each transistor, changing the cell dimension in the vertical and horizontal directions.

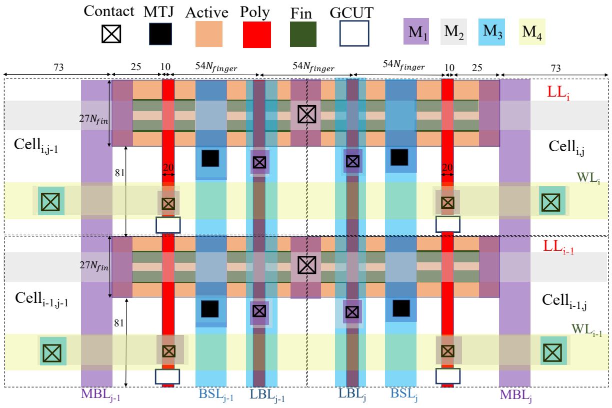


Figure 3.5: The layout of four adjacent CRAM cells in ASAP7.

3.4.2 Layout of the CRAM Array

The CRAM cell can be tessellated into an array. Fig. 3.5 shows the layout of four adjacent CRAM cells (2×2), again using a 1-fin 1-finger FinFET, under ASAP7 design rules [7, 8]. For example, the minimum allowable active width is 27nm; the poly length and pitch are 20nm and 54nm, respectively; the minimum active to active distance in our design can be 54nm: under these constraints, the size of the smallest 1-fin, 1-finger CRAM cell is 108nm×189nm. The addition of each fin increases the cell width (vertical dimension) by 27nm, keeping the cell length (horizontal dimension) fixed, while adding each finger increases the cell length by 108nm, leaving the width fixed. The width and length for a cell with N_{fin} fins and N_{finger} fingers are:

$$W_{cell} = 108 + 27(N_{fin} - 1) \tag{3.4}$$

$$L_{cell} = 189 + 162(N_{finger} - 1) \tag{3.5}$$

3.4.3 Impact of Layout Choices on $(A_{cell}, AR_{cell}, R_T)$

The choice of W_{cell} and L_{cell} can impact the cell area, $A_{cell} = W_{cell} \times L_{cell}$, and the cell aspect ratio, $AR_{cell} = W_{cell}/L_{cell}$. From Eqs. (3.4), and (3.5), the following trends can be inferred as the numbers of fins and fingers are changed:

- By increasing N_{fin} and N_{finger}, both W_{cell} and L_{cell} increase, increasing A_{cell}.
- For a fixed N_{fin}, the largest AR_{cell} has the lowest L_{cell}, i.e., $N_{finger} = 1$. If we fix N_{finger}, then by increasing N_{fin}, W_{cell} increases; thus, AR_{cell} increases.

Next, we study the impact of the numbers of fins and fingers on the transistor resistance, R_T, for both advanced CRAM and Today's CRAM. We apply the nominal voltage of ASAP7 (0.7V) to the FinFET gate, and for this value of gate-to-source voltage, V_{gs}, we use the transistor I-V curve to determine the resistance corresponding to the drain-to-source current $I_{ds} = I_c$ (Table B.1) required to switch an MTJ. Today's MTJ requires larger I_c than the advanced MTJ: hence, for the same N_{fin} and N_{finger}, R_T is larger for today's CRAM. The $N_{fin} = N_{finger} = 1$ case can deliver I_c for the advanced MTJ, but not for today's MTJ: larger sizes must be used for the latter. As expected, as N_{fin} and N_{finger} are increased, R_T reduces.

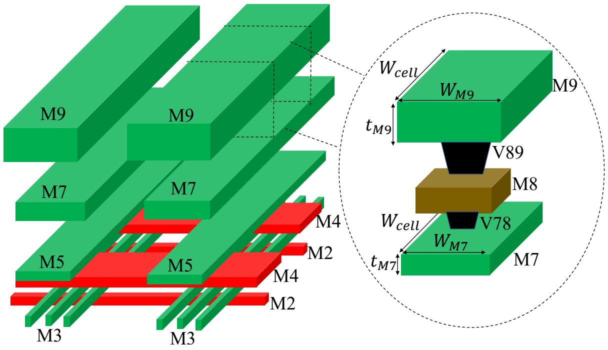


Figure 3.6: Configuration of BSLs and LLs. The green and red lines correspond to BSLs, and LLs, respectively.

3.4.4 Metal Layer Configurations and Specifications

As seen in Section 3.3, parasitics in the BSLs and LLs play a large part in limiting the allowable size of the CRAM array. To overcome this, we use a multi-metal layer architecture for BSLs and LLs, illustrated in Fig. 3.6 for the four adjacent CRAM cells of Fig. 3.5. Here, metal layers M3, M5, M7, and M9 are allocated to the BSLs, and M2 and M4 are allocated to LLs, with vias connecting each type of line across layers. The interconnect specifications – the metal thickness (t_M), resistivity (ρ_M), minimum spacing (S_{min}), minimum width (W_{min}), and via parameters – are taken from [7,8].

3.5 Thevenin Modeling for Each CRAM Row

From Section 3.3, V_b is degraded by IR drops as it reaches the last row. We use a Thevenin model to model the Thevenin voltage, V_{th} and resistance, R_{th}, at the last row of the CRAM (prior work [2] that neglects wire parasitics is a special case of our model where $V_{th} = V_b, R_{th} = 0$). We denote:

$$\alpha_{th} = V_{th}/V_b. \tag{3.6}$$

Clearly, $\alpha_{th} \leq 1$ because V_{th} is a degraded version of V_b due to the voltage drop across the wire parasitics. We propose recursive expressions (see appendix A) for R_{th} and α_{th} as functions of array parameters.

Fig. 3.7 illustrates the voltage ranges for implementations of the same gate in the first row and the last row. For the first row, the effect of parasitics is negligible and the allowable voltage range lies within the minimum and maximum values specified in Table 3.1: we denote these as V_{min} and V_{max}, respectively. However, for an implementation of the same gate in the last row, we must consider R_{th} in series with the equivalent resistance across the MTJ devices in the last row, and an applied voltage of V_{th}.

For example, for a BUFFER in the first row, the range of V_b is provided in Table 3.1. The last row is driven by with V_{th} in series with R_{th}, and the corresponding range is:

$$\begin{gathered}(R_A + R_B + R_{th})I_c \leq V_{th} \leq (2R_B + R_{th})I_c \\ \text{i.e., } (R_A + R_B + R_{th})\frac{I_c}{\alpha_{th}} \leq V_b \leq (2R_B + R_{th})\frac{I_c}{\alpha_{th}}\end{gathered} \tag{3.7}$$

where the latter expression follows from (3.6). Since $\alpha_{th} < 1$, this implies that both the lower and upper bound for V_b are higher in the last row than in the first row.

For each gate type, expressions for $V^{'}_{min}$ and $V^{'}_{max}$ can be modified from the parasitic-free cases of Table 3.1 as follows:

$$V^{'}_{min} = \frac{V_{min} + R_{th} \times I_c}{\alpha_{th}}; \quad V^{'}_{max} = \frac{V_{max} + R_{th} \times I_c}{\alpha_{th}} \tag{3.8}$$

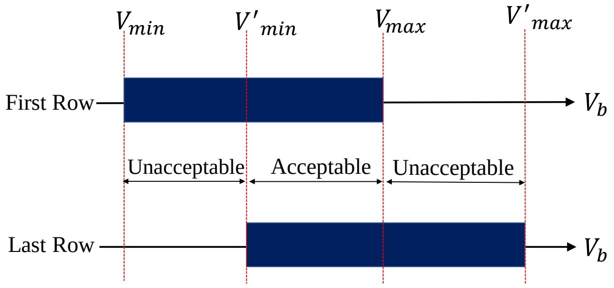


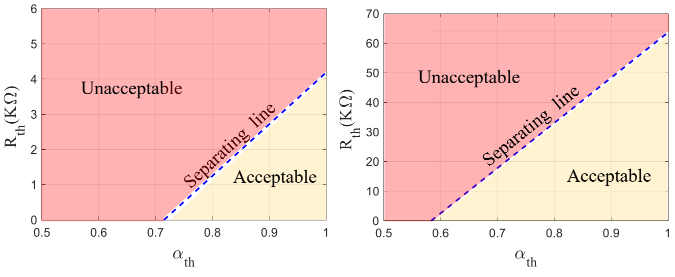
Figure 3.7: Required voltage ranges for implementations of the same gate in the first row and the last row.

For the gate to function correctly in all rows, the allowable range of V_b is the intersection of the intervals $[V_{min}, V_{max}]$ and $[V^{'}_{min}, V^{'}_{max}]$: this is marked as the acceptable region for V_b in Fig. 3.7. Clearly, for correct functionality, these two intervals must have nonzero intersection, i.e., $V^{'}_{min} < V_{max}$

For each gate type, this leads to a boundary ("separating line") between a functional and nonfunctional implementation. From Eq. (3.8), the separating line constraint is

$$R_{th} < (V_{max} \times \alpha_{th} - V_{min})/I_c \tag{3.9}$$

Fig. 3.8 shows the separating lines for today's and advanced MTJ technology in a R_{th} vs. α_{th} plot, while Eq. (3.8) shows the equation for the separating line for each gate. The separating line demarcates the unacceptable region, where the gate functions incorrectly, from the acceptable region. It can be observed that the acceptable region of advanced CRAM is larger that of today's CRAM (note that the y-axis scale in the plots is different), providing more choices for designing parameters in advanced CRAM.



(a) (b)

Figure 3.8: Separating lines of implementation for the AND gate in today's CRAM and the advanced CRAM.

We define the noise margin, NM, as the range of allowable values for V_b. When all wire parasitics are zero, $NM = (V_{max} - V_{min})/V_{mid}$, where $V_{mid} = (V_{max} + V_{min})/2$, but in the presence of parasitics, this changes to

$$NM = (V_{max} - V_{min}^{'})/V_{mid}^{'} \tag{3.10}$$

where $V_{mid}^{'} = (V_{max} + V_{min}^{'})/2$. Clearly, we desire $NM > 0$.

3.6 Results and Discussion

3.6.1 Impact of CRAM Parameters on NM

Effect of N_{row}: To examine how NM changes when the number of rows is altered, we fix the transistor configuration by choosing $N_{fin} = 2$, $N_{finger} = 4$. This corresponds to an R_T of 570Ω for today's CRAM and 597Ω for the advanced CRAM. We also fix $AR_{cell} = 0.26$ and set $d_{column} = 10$, i.e., we consider the worst-case NM when $d_{column} = 10$.

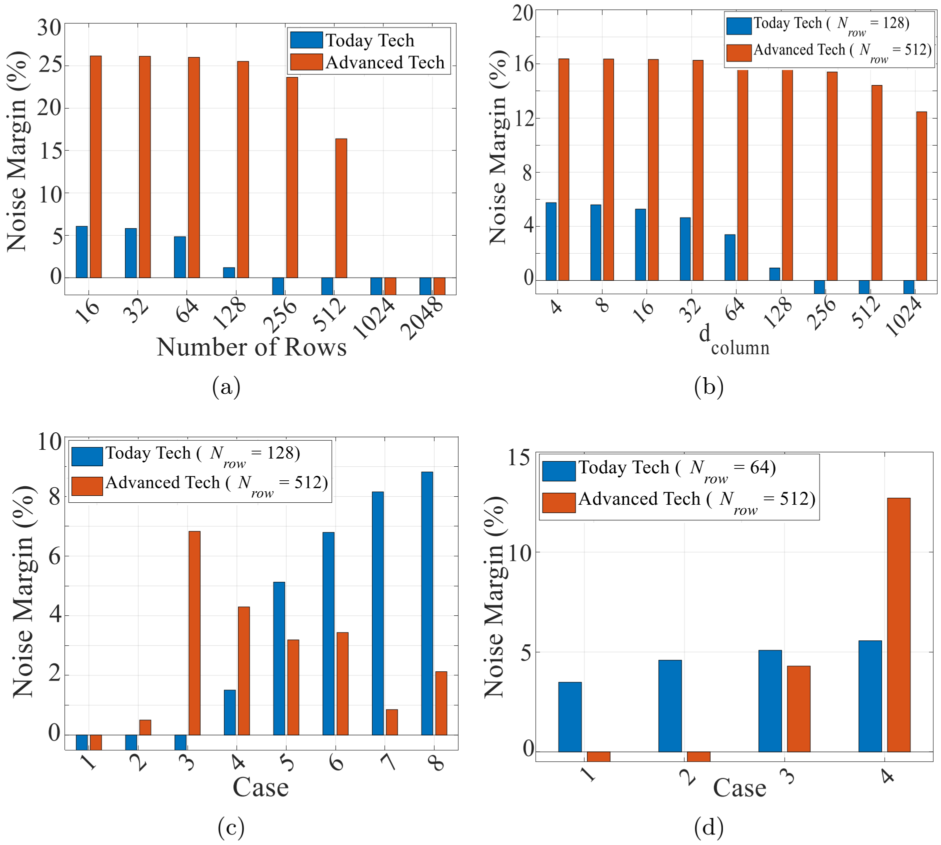
We analyze eight different cases with different N_{row} values (16, 32, 64, 128, 256, 512, 1024, and 2048) in today's and advanced CRAM. Each case corresponds to a point in the R_{th}-α_{th} plane. For today's CRAM, the points corresponding to $N_{row} \leq 128$ are located in the acceptable area, i.e., the maximum allowable N_{row} under this choice of $\{R_T, AR_{cell}, d_{column}\}$ is 128. For the advanced CRAM, the acceptable points correspond to $N_{row} \leq 512$. The noise margins are graphically depicted in Fig. 3.9(a).

Effect of d_{column}: By increasing the relative distance between input columns and the output column (d_{colum}), the parasitics associated with the LL in each row (R_x) increase. Fig. 3.9(b) shows NM for different cases with different d_{column} values in today's and advanced CRAM. For the advanced CRAM, the value of d_{column} does not affect NM significantly for the shown values because the LL parasitic resistance, $R_x \ll R_{MTJ}$, the MTJ resistance to which it is connected in series. For today's CRAM, only $d_{column} \leq 64$ provide a positive NM, because R_x is comparable to R_T for today's MTJs. High values of d_{column} create a large drop across the parasitics, causing V_b to be infeasible.

Similar trends are seen for the BUFFER, where the range of a copy operation is limited using today's CRAM. Thus, copy operations over large distances must be performed in multiple steps, adding to the energy and computation time.

Effect of R_T: To analyze the effect R_T, we must consider that A_{cell} changes accordingly if we vary R_T. The choice of R_T can affect NM through two mechanisms: (a) directly, since a reduction in R_T increases the noise margin for an array of constant size, and (b) indirectly, since a reduction in R_T increases the cell size, and hence the array size, thereby increasing line parasitics R_x and R_y.

In Table 3.4, we present eight cases where R_T gradually decreases from case 1 to case 8. For both advanced and today's CRAM, case 1 has the smallest R_T (the largest A_{cell}), and case 8 has the largest R_T (the smallest A_{cell}). We choose W_{cell} and L_{cell} so that the AR_{cell} values are roughly constant (it is not possible to ensure equality since R_T is changed over a discrete space by altering N_{fin} and N_{finger}). We set N_{row} to 128 and 512, respectively, for today's and advanced technologies, and set $d_{column} = 10$ for both cases.



(a)

(b)

(c)

(d)

Figure 3.9: NM for an AND gate in today's and advanced CRAM, varying (a) N_{row}, (b) d_{column}, (c) R_T, and (d) AR_{cell}.

Fig. 3.9(c) shows the NM for each of these cases. For today's technology, large R_T values (Cases 1, 2, and 3) cause negative NM, as in these cases, R_T values are comparable to today's MTJ resistances, and reducing R_T further improves NM. The direct mechanism is dominant here, and reducing R_T improves NM monotonically. In contrast, for advanced MTJs, there is a nonmonotone relationship as R_T is reduced. At first, NM improves due to the first mechanism, and then it worsens due to the second mechanism. Part of the nonmonotonicity (e.g., between Cases 5 and 6) can be attributed to the fact that AR_{cell} is not strictly constant in Table 3.4 (in fact, for Case

Table 3.4: Analyzing the effect of R_T

	Today's CRAM			Advanced CRAM		
	R_T	A_{cell}	AR_{cell}	R_T	A_{cell}	AR_{cell}
1	5.99KΩ	0.020μm^2	0.4	5.73KΩ	0.020μm^2	0.57
2	1.72KΩ	0.038μm^2	0.31	2.87KΩ	0.029μm^2	0.40
3	1.04KΩ	0.047μm^2	0.38	1.90KΩ	0.038μm^2	0.31
4	0.76KΩ	0.058μm^2	0.31	0.95KΩ	0.047μm^2	0.38
5	0.49KΩ	0.067μm^2	0.37	0.63KΩ	0.057μm^2	0.41
6	0.36KΩ	0.082μm^2	0.43	0.48KΩ	0.067μm^2	0.38
7	0.29KΩ	0.096μm^2	0.36	0.35KΩ	0.081μm^2	0.43
8	0.23KΩ	0.110μm^2	0.42	0.23KΩ	0.110μm^2	0.42

Table 3.5: Analyzing the effect of AR_{cell}

	Today's CRAM			Advanced CRAM		
	AR_{cell}	A_{cell}	R_T	AR_{cell}	A_{cell}	R_T
1	0.80	0.058μm^2	0.59KΩ	1.14	0.041μm^2	1.14KΩ
2	0.54	0.066μm^2	0.49KΩ	0.60	0.044μm^2	0.95KΩ
3	0.38	0.069μm^2	0.49KΩ	0.38	0.047μm^2	0.95KΩ
4	0.26	0.069μm^2	0.59KΩ	0.21	0.055μm^2	1.14KΩ

1 for the advanced CRAM, the FinFET has $N_{fin} = N_{finger} = 1$, and the corresponding $AR = 0.57$ is the only option). Over the eight choices, one can choose Case 3 as the optimal point that provides the best NM.

Effect of AR_{cell}: We now vary AR_{cell} by changing N_{fin} and N_{finger}, while keeping R_T and A_{cell} relatively fixed. As before, we set N_{row} to 128 and 512, respectively, for today's CRAM and the advanced CRAM; $d_{column} = 10$, and R_T and A_{cell} are kept roughly constant, to the extent possible in the discrete space of N_{fin} and N_{finger}.

Fig. 3.9(d) shows the results for the four cases Table 3.5. Cases with smaller AR_{cell}, which have shorter BSLs with lower parasitic resistances (R_y), have a larger NM. Thus, an appropriate choice of AR_{cell} can improve the performance of the CRAM without area overhead. For example, in advanced CRAM, the NM for case 1, with the smallest AR_{cell}, is negative, but NM improves as AR_{cell} is increased.

3.6.2 An Optimal Design for Each Gate

Table 3.6 evaluates the implementations of three types of arrays using both today's and advanced CRAMs with various degrees of versatility: Array 1 implements a basic set of combinational logic gates (INV, BUFFER, AND/NAND, OR/NOR); Array 2 adds the MAJ3 and $\overline{\text{MAJ3}}$ gates to this set; Array 3 further adds MAJ5 and $\overline{\text{MAJ5}}$. It is easily seen that for more versatile arrays, the array size is more constrained. The improvement from today's CRAM to the advanced CRAM is also visible: e.g., today's CRAM cannot implement Array 3, regardless of array size [2].

To obtain the largest allowable size of N_{row} we change the locations of the BSL drivers. As compared to the previous analysis where a driver was placed at one end of the array, we effectively double N_{row} by using a 2× driver in the middle of the array, or by using two 1× drivers at either end of the array (Fig. 3.10). The area overheads are modest.

Table 3.6: Optimal design options for arrays with different functionalities

		N_{fin}, N_{finger}	R_T (KΩ)	A_{cell} (μm^2), AR_{cell}	N_{row}, d_{column}	Subarray Size
1	Advanced	4, 4	0.357	0.127, 0.280	512, 512	32KB
	Today's	5, 7	0.113	0.251, 0.186	128, 64	1024KB
2	Advanced	2, 6	0.476	0.135, 0.134	256, 256	4KB
	Today's	4, 9	0.101	0.281, 0.127	128, 16	256KB
3	Advanced	3, 9	0.171	0.267, 0.098	256, 64	2KB
	Today's	-	-	-	-	-

Note that the constraint on N_{row} limits the array size but not the CRAM size: the overall CRAM consists of a tiled set of arrays, each with N_{row} rows, and all controlled by the same set of control signals. The choice of d_{column}, however, does not constrain the tile size, but merely the computation distance. If the operands of a computation are at a distance $> d_{column}$ from each other, then they must be copied to new cells that are within the d_{column} limit. This is often not a problem: all of the computations shown in [2] lie within the d_{column} constraint listed in Table 3.6. For this reason, practically,

N_{row} is much more constraining than d_{column}.

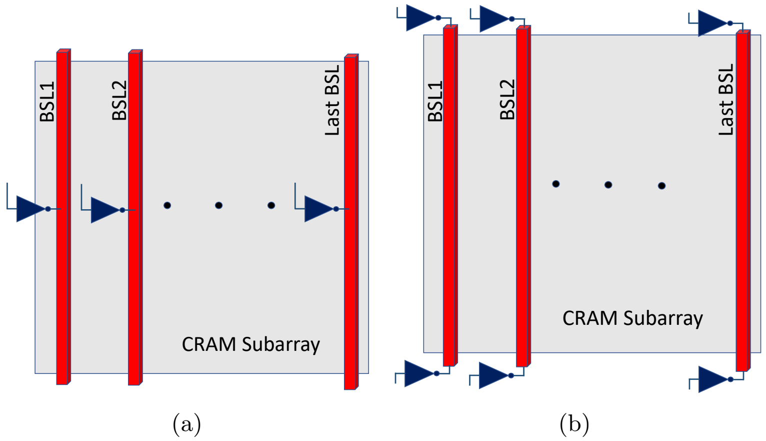


Figure 3.10: Increasing N_{row} by inserting (a) 2× drivers in the middle of the array, and (b) two 1× drivers at either end.

3.7 Conclusion

We have presented a methodology based on actual layout considerations for analyzing the parasitic effects in STT-CRAM. We have demonstrated that interconnect parasitics have a significant effect on CRAM performance and have developed a comprehensive model for analyzing this impact. Using this methodology, we have developed guidelines for the array size, N_{row}, and the maximum distance between columns for an operation. We show that for both today's and advanced technologies, CRAM cell layouts with smaller aspect ratios are desirable, as this helps control critical BSL parasitics. Reducing access transistor resistance is important for today's technology but is not a significant factor for advanced technologies. For the SHE-CRAM [19], a similar analysis shows that interconnect parasitics are not significant as the current values are much smaller.

Chapter 4

Using Spin-Hall MTJs to Build an Energy-Efficient In-memory Computation Platform

4.1 Introduction

To further improve previous CRAM efficiency, this chapter uses a novel 3-terminal MTJ, whose write mechanism is based on the spin-Hall effect (SHE). The SHE-MTJ delivers improved speed and energy-efficiency over the traditional 2-terminal STT-MTJ [13], and recent research on novel spin-Hall materials, e.g., sputtered BiSe$_x$ [3], which experimentally demonstrates very high spin-Hall angles, will lead to further gains over today's SHE-MTJs. Moreover, the separation of write and read paths in the 3-terminal SHE-MTJ makes this device more reliable than the STT-MTJ [4].

However, due to differences between the SHE-MTJ and the STT-MTJ (e.g., in the number of terminals and in the operation mechanism), building a SHE-CRAM is more complex than simply replacing STT-MTJs in the STT-CRAM with SHE-MTJs.

In this chapter, we show how the STT-CRAM architecture must be altered for SHE-MTJs, and that these changes require alterations to operation scheduling schemes. We propose a double-write-line array structure for the SHE-CRAM, and present new data placement and scheduling methods for the implementation of computational blocks in the SHE-CRAM. By evaluating computations on representative applications, we show that, in comparison with the STT-based CRAM, the SHE-CRAM demonstrates an overall improvement in latency and energy.

4.2 SHE CRAM Structure

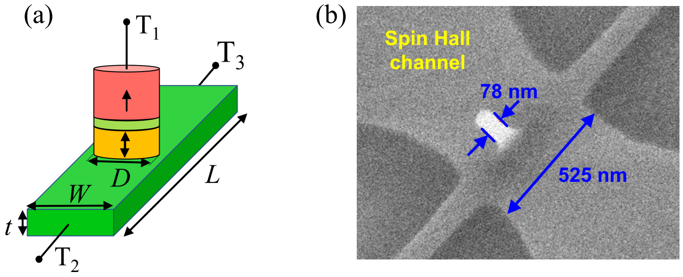


Figure 4.1: (a) Schematic of a 3-terminal SHE-MTJ. (b) SEM image taken at 60°from the perpendicular direction showing an MTJ pillar (with resist) fabricated on the W spin Hall channel.

The structure of 3-terminal SHE-MTJ is shown in Fig. 4.1(a). It is composed of a conventional perpendicular MTJ (pMTJ) stack seated on a spin-Hall (SH) channel, where the free layer of MTJ is directly contacted with the channel. Depending on the free layer orientation, the MTJ can have one of two resistance states: parallel (R_P, logic 0), and anti-parallel (R_{AP}, logic 1), where $R_{AP} > R_P$. The free layer orientation is controlled by the direction of the current through the SH channel (between T_2 and T_3 in Fig. 1(a)). Due to the SHE, when the current density exceeds the threshold current density J_{SHE}, the magnetization of the free layer of the MTJ is set according to the current direction [77]. The read operation measures the current between T_2 and

T_3, which depends on the MTJ state. Fig. 4.1(b) shows a fabricated SHE-MTJ with a diameter of 78 nm over a 525 nm wide SH channel; miniaturization to 10nm geometries is possible.

Fig. 4.2 shows the architecture of the SHE-CRAM array, which can operate in memory or logic mode. At the bitcell level, this structure is quite different from the STT-CRAM. The 2T1MTJ bitcell accommodates the 3-terminal SHE-MTJ: each cell has one SHE-MTJ with two terminals gated by access transistors. Each row has two select lines (SLs), ESL and OSL – which select the even and odd columns, respectively – and a logic line (LL); each column has a read and write word line (WLR, WLW). At the array level, the arrangement of wires must accommodate the connections required by the 3-terminal SHE-MTJ. Conventionally, the word line in a memory is drawn as a horizontal line, but we show a rotated array where the word lines run vertically. We make this choice for convenience so that we can compactly show the sequence of computations in later figures.

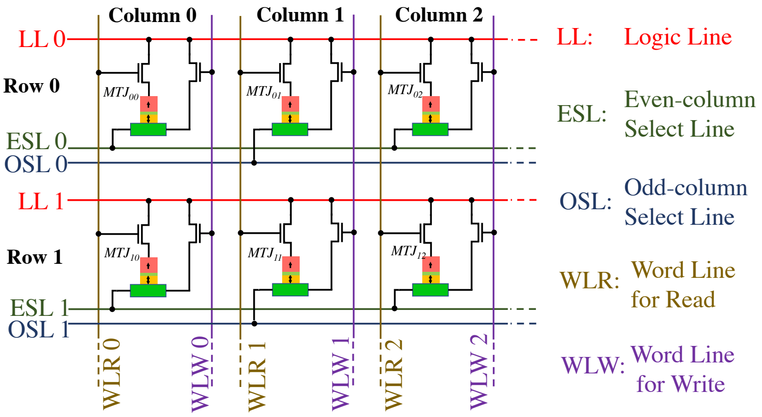


Figure 4.2: Overall structure of the SHE-CRAM.

In memory write mode (Fig. 4.3(a)), the transistor connected to WLR is off, WLW is high, turning on the write access transistor, and the SL is either positive or negative (i.e., one of two current directions is applied) depending on whether a 0 or 1 is to be written. This current through the SH channel writes to the MTJ. In memory read mode (Fig. 4.3(b)), WLR is set high to turn the read transistor on. A current is passed through the MTJ between LL and the SL to sense its resistance, i.e., the memory state, by connecting the SL to a sense amplifier.

In logic mode (Fig. 4.3(c)), a logic operation can be performed between cells in a CRAM row. For input cells, transistors connected to their WLR lines are turned on, and for the output cell, the WLW access transistor is turned on to allow current to flow through the spin-Hall channel. The LL is left floating, the SL for the inputs is set to a specified voltage, and the SL for the output is grounded. This implies that a current, whose value depends on the states of the inputs, passes through the spin-Hall channel of the output MTJ.

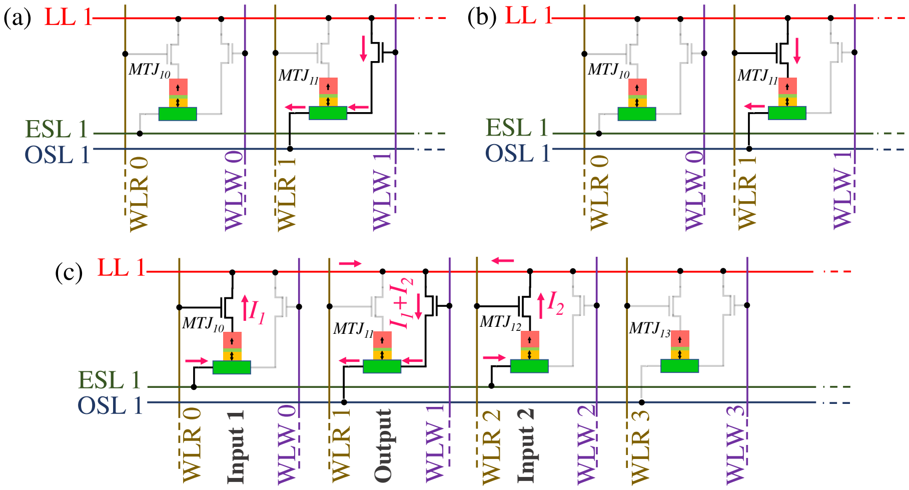


Figure 4.3: Current flow during: (a) memory write operation, (b) memory read operation, and (c) logic mode.

Fig. 4.4(a) isolates the part of the array involved in a logic operation with two inputs, and shows its equivalent circuit in Fig. 4.4(b), where the resistor values depend on the state variables (MTJ resistances) and transistor resistances. Before the computation starts, the output MTJ is initialized to a preset value. By applying bias voltage V_b

across the input and output cell SLs, current $I_1 + I_2$ flows through the spin-Hall channel of the output, where the magnitude of each current depends on the input MTJ state (i.e., resistance). If $I_1+I_2 > I_{SHE}$, where I_{SHE} is the SHE threshold current, then depending on the current direction, a value is written to the output MTJ state; otherwise, the preset output state remains intact. As explained in Sec. 4.3B, by appropriately choosing the voltages and output preset, different logic functions can be implemented.

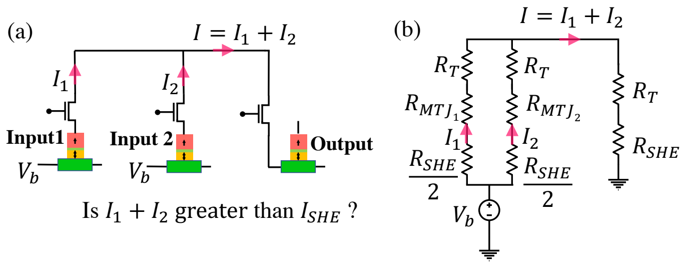


Figure 4.4: (a) Performing a logic operation in a row of SHE-CRAM, and (b) the equivalent circuit model.

Table 4.1: Status of lines and transistors in the SHE-CRAM during memory and logic modes.

Operation		BLW	BLR	Transistor Connected to BLW	Transistor Connected to BLR	WLA	WLB	LL
Memory Mode	Write	High	Low	ON	OFF	WLA active (cell in even column) WLB active (cell in odd column)		Ground
	Read	Low	High	OFF	ON	WLA active (cell in even column) WLB active (cell in odd column)		Ground
Logic Mode	Inputs Cells	Low	High	OFF	ON	Both WLA and WLB are active, $\lvert V_{WLA}$ - $V_{WLB} \rvert = V_b$		Float
	Outputs Cells	High	Low	ON	OFF			

Note that in the logic mode, all input operands must be in even-numbered columns, and the output must be in an odd-numbered column – or vice versa. This is unlike the STT-CRAM, where no such limitation is necessary, and is a consequence of the 3-terminal structure of the SHE-MTJ.

Table 4.2: SHE-MTJ specifications [3–5].

Parameters	Value
MTJ type	CoFeB/MgO p-MTJ
Spin Hall channel material	Sputtered BiSe_x
MTJ diameter (D)	10 nm
Spin Hall channel length (L)	30 nm
Spin Hall channel width (W)	15 nm
Spin Hall channel thickness (t)	4 nm
Spin Hall channel sheet resistance (R_{Sheet})	32 kΩ
Spin Hall channel resistance (R_{SHE})	64 kΩ
MTJ RA product	20 $\Omega\cdot\mu m^2$
MTJ TMR ratio	100%
MTJ Parallel resistance (R_P)	253.97 kΩ
MTJ Anti-parallel resistance (R_{AP})	507.94 kΩ
STT critical current density (J_{STT})	5×10^6 A/cm^2
SHE threshold current density (J_{SHE})	5×10^6 A/cm^2
STT threshold current (I_{STT})	3.9 µA
SHE threshold current (I_{SHE})	3 µA
SHE pulse width (t_{SHE})	1 ns
Transistor Resistance (R_T)	1 kΩ

The three modes – memory read/write and logic mode – are summarized in Table 4.1.

4.3 SHE CRAM Detail

Table 4.2 defines the parameters of the SHE-MTJ and provides typical values, to be used in the rest of this chapter. The dimensions of the SHE-MTJ in Table 4.2 are appropriately chosen to (a) provide an optimal margin window (see next sections), (b) provide a low I_{SHE}, and (c) avoid unwanted STT switching during logic operations.

4.3.1 Device-level design

The specifications of the SHE-MTJ in the SHE-CRAM are shown in Table 4.2. For our evaluation, the novel sputtered BiSe$_x$ is used as the SH channel, due to its high spin-Hall efficiency [3]. Fig. 4.5 demonstrates the SHE switching of such a structure which requires a very low switching current density. The device is a micron-size Hall bar, which is composed of BiSe$_x$ (5nm) / Ta (0.5 nm) as the SH channel and CoFeB (0.6nm) /Gd (1.2nm) /CoFeB(1.1nm) as the magnetic layer. The easy-axis of the magnetic layer is along the out-of-plane direction. Two magnetization states (up or down, corresponding to the positive or negative Hall resistance) are revealed from the loop in Fig. 4.5(a). The magnetization can be switched between the two states by injecting a current through the SH channel, as shown in Fig. 4.5(b). The threshold switching current density J_{SHE} is determined to be 4.4 × 105 A/cm^2, which is two orders lower than normal spin-Hall structures with metal like Ta, W, or Pt as the SH channel. In Table 4.2, J_{SHE} is set to 5 × 106 A/cm^2, based on [5]. Note that although an external magnetic field is applied to assist spin-Hall switching in Fig. 4.5(b), the external field is not necessary under field-free strategies [78,79]. Note that the choice for L, W, and t is based on an optimization described in Sec. 4.3C.

4.3.2 Gate-level design

In logic mode, the configuration of the SHE-CRAM into various gate types is controlled by two factors: (a) output preset value, (b) bias voltage, V_b (Fig. 4.4(a)). By modeling the current path of each gate as in Fig. 4.4(b), we can determine the conditions for implementing each gate type. The voltage V_b applied across the MTJ interconnections in logic mode falls across ESL and OSL. This voltage, applied across $(R_{SHE}/2+R_{MTJ_1})+R_T)||(R_{SHE}/2+R_{MTJ_2}+R_T)$ in series with $(R_{SHE}+R_T)$, is shown in Fig. 4.4(b). Here, $||$ represents the equivalent resistance of resistors in parallel. For the configuration in

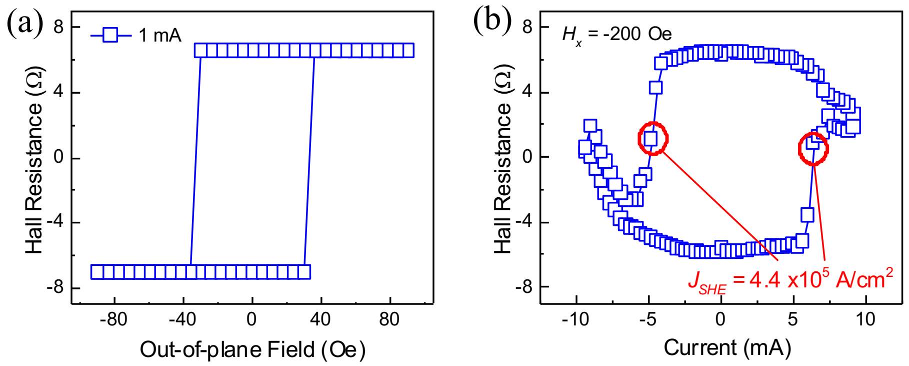


Figure 4.5: Demonstration of SHE switching with ultra-low J_{SHE} in a Hall bar device [3]. The SH layer is composed of BiSex (5) /Ta (0.5), and the perpendicular magnetic layer is composed of CoFeB (0.6) /Gd (1.2) /CoFeB (1.1) (all thicknesses in nm). (a) The out-of-plane hysteresis loop showing the two magnetization states of the device. (b) SHE switching loop of the device with a very low switching current.

Fig. 4.4(b), the current I through the logic line is

$$I = V_b/([(R_{SHE}/2 + R_{MTJ_1} + R_T)||(R_SHE/2 + R_{MTJ_2} + R_T)] + R_3) \tag{4.1}$$

If V_b is too low, $I < I_{SHE}$, and the current is insufficient to switch the output; if it is too high, $I > I_{SHE}$, and the output is switched regardless of the input state. The resistance of the MTJ may take on one of two values, R_P or R_{AP}. For conciseness, we define R_1, R_2, and R_3 as:

$$R_1 = R_{SHE}/2 + R_P + R_T \tag{4.2}$$

$$R_2 = R_{SHE}/2 + R_{AP} + R_T \tag{4.3}$$

$$R_3 = R_{SHE} + R_T \tag{4.4}$$

Consider the case where the gate in Fig. 4(a) is used to implement a 2-input AND gate. For each of the input states (00 through 11), we can calculate the currents flowing through the spin-Hall channel of the output MTJ as:

$$I_{00} = V_b/(R_1/2 + R_3) \tag{4.5}$$

$$I_{01} = I_{10} = V_b/((R_1||R_2) + R_0) \tag{4.6}$$

$$I_{11} = V_b/(R_2/2 + R_3) \tag{4.7}$$

For the AND gate the preset output value is 1. For correct AND operation, we must choose V_b appropriately so that $I_{00} > I_{SHE}$ and $I_{01} = I_{10} > I_{SHE}$ (i.e., both cases, the preset output is switched to 0), and $I_{11} < I_{SHE}$ (i.e., the output stays at 1). Since $R_P < R_{AP}$, $R_1 < R_2$. Therefore, from eq. (4.5), eq.(4.6) (as we discussed in section 2.3.1), and eq.(4.7),

$$I_{11} < I_{01} = I_{10} < I_{00}. \tag{4.8}$$

Thus, if we chose V_b to be large enough so that $I_{01} = I_{10} > I_{SHE}$, then $I_{00} > I_{SHE}$ must always be true. From eq. 4.6, the following constraint must be obeyed.

$$V_b > ((R_1||R_2) + R_3)I_{SHE} \tag{4.9}$$

However, to ensure the correctness of the 11 input case, V_b cannot be too large. Specifically, from eq. 4.7, it is required that $I_{11} < I_{SHE}$, which leads to the second constraint,

$$V_b < (R_2/2 + R_3)I_{SHE}. \tag{4.10}$$

These two constraints limit the range of V_b for the AND gate. A NAND gate is identical to the AND, except that a preset value of 0 is used; the range of V_b is identical to the AND. Similar constraints can be derived for other logic gates, and the bias voltage ranges to implement other gates can be calculated similarly. Table 4.3 summarizes the bias voltage ranges and the preset value for various gate types using the parameters of Table 4.2 for the SHE.

For each gate, we can define Noise Margin (NM) of V_b, which is defined as [2]:

$$NM = (V_{max} - V_{min})/V_{mid}; V_{mid} = (V_{max} + V_{min})/2 \tag{4.11}$$

Table 4.3: Bias voltage ranges, and output preset value.

Gate	Preset	Closed form formula for bias voltage range	Numerical value (Volt)
NOT	0	$(R_1 + R_3)I_{SHE} < V_b < (R_2 + R_3)I_{SHE}$	1.065 – 1.827
Buffer	1		
NAND	0	$\left(\frac{R_1R_2}{R_1 + R_2} + R_3\right)I_{SHE} < V_b < \left(\frac{R_2}{2} + R_3\right)I_{SHE}$	0.768 – 1.017
AND	1		
NOR	0	$\left(\frac{R_1}{2} + R_3\right)I_{SHE} < V_b < \left(\frac{R_1R_2}{R_1 + R_2} + R_3\right)I_{SHE}$	0.636 – 0.768
OR	1		
$\overline{\text{MAJ3}}$	0	$\left(\frac{R_1R_2}{R_1 + 2R_2} + R_3\right)I_{SHE} < V_b < \left(\frac{R_1R_2}{2R_1 + R_2} + R_3\right)I_{SHE}$	0.546 – 0.624
MAJ3	1		
$\overline{\text{MAJ5}}$	0	$\left(\frac{R_1R_2}{2R_1 + 3R_2} + R_3\right)I_{SHE} < V_b < \left(\frac{R_1R_2}{3R_1 + 2R_2} + R_3\right)I_{SHE}$	0.418 – 0.446
MAJ5	1		

where V_{max} and V_{min} are, respectively, the upper and lower limits on V_b, and V_{mid} is the midpoint of the bias voltage range. To maximize noise immunity, we chose V_{mid} as the actual applied voltage. The energy, E, dissipated by each gate, is

$$E = V_{mid} I_{SHE} t_{SHE}. \tag{4.12}$$

Using the values listed in Table 4.3, the NM and energy for various SHE-CRAM based logic implementations are computed. We compare the noise margin and energy of logic operations in the STT-CRAM for todayâs STT-MTJs as reported in [2], and the SHE-CRAM. From Fig. 4.6, the SHE-CRAM always results in higher noise margins compared to STT-CRAM. This can be attributed to the fact that the resistances (R_{MTJ}) associated with the logic inputs are significantly higher than the resistance R_{SHE} associated with the output, which provides a larger allowable interval for V_b. In contrast, the inputs and outputs for the STT-CRAM are both correspond to MTJ resistances. A comparison of energy-efficiency shows that in all cases, the SHE-CRAM has lower switching current and faster switching time than the STT-CRAM, resulting in better E (Fig. 4.7).

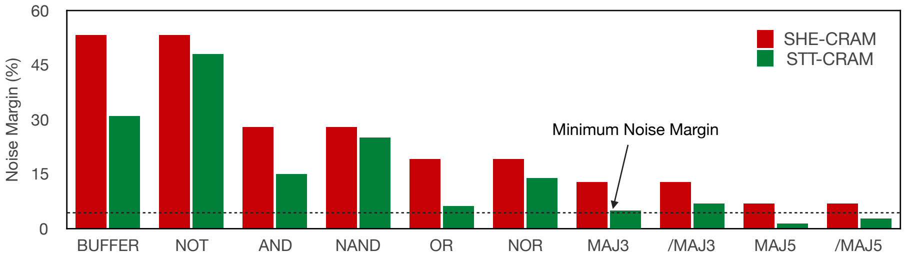


Figure 4.6: Comparison of noise margin between gates implemented using STT-CRAM and SHE-CRAM.

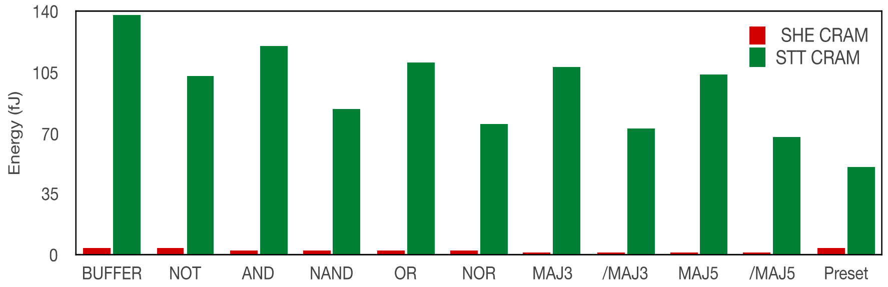


Figure 4.7: Comparison of energy between gates implemented using STT-CRAM and SHE-CRAM.

4.3.3 Optimization of spin-Hall channel dimensions

To further improve device performance, we can optimize the dimensions of spin-Hall channel in the SHE-MTJ device with respect to NM and E. The spin-Hall channel resistance is

$$R_{SHE} = R_{Sheet}(L/W) \tag{4.13}$$

where $L \geq W$. For a NAND (or AND) gate, from eq. 5.7,

$$NM_{NAND} = (R_2(1 - R_1/(R_1 + R_2)))/(R_2/2 + (R_1R_2)/(R_1 + R_2) + 2R_3). \tag{4.14}$$

Similarly, energy for the implementation of a NAND (or AND) gate is rewritten by:

$$E_{NAND} = (WtJ_{SHE})^2(R_2/4 + (R_1R_2)/(2(R_1 + R_2)) + R_3)t_{SHE} \tag{4.15}$$

In Fig. 4.8, the corresponding noise margin and energy of a NAND (or AND) gate is shown. In Fig. 4.8(a), by reducing the length to width ratio (L/W) of the SH channel, R_{SHE} decreases. In each case, the optimal V_b that maximizes the noise margin NM is found as the midpoint of the allowable interval of V_b. While NM depends on R_P and R_{AP} as well as R_{SHE}, it can be shown (by examining the sensitivity of NM to R_{SHE}) that NM is most sensitive to the reduction in R_{SHE} (details omitted due to space limitations). This causes NM to decrease with increasing (L/W). Increasing the channel thickness t reduces R_{Sheet}, thus decreasing R_{SHE}: as before, this increases NM.

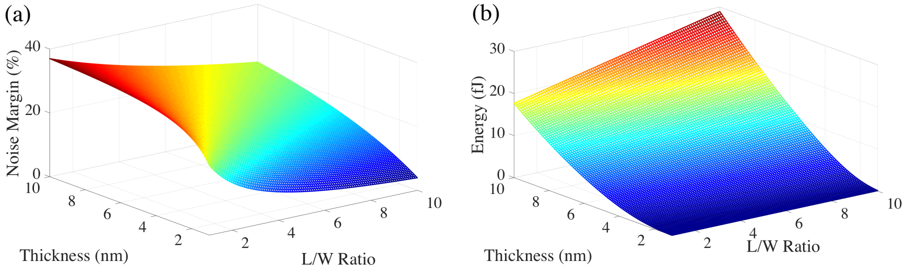


Figure 4.8: Impact of SHM geometry on NM and energy.

In Fig. 4.8(b), by increasing L/W (or t), the energy increases. To maximize noise margin and minimize energy, L/W should be as small as possible (due to fabrication considerations the ratio is considered 2 rather than 1). For the choice of t, a balance between NM and energy must be found. Although a larger thickness increases NM, it increases the energy. As a compromise, based on Fig. 4.8, we choose a near middle point of t = 4 nm (providing 32% energy improvement with 3% degradation in NM compared to the middle point of 5 nm).

4.3.4 Functional-level design

Full adders: The original STT-CRAM [17] built a NAND based implementation of full adder (FA) using 9 steps. Using majority logic one can implement a FA, as shown in

Fig. 4.9, and this requires only 3 steps [38]. STT-CRAM technology has very limited NM for majority gates; in contrast, the NM in SHE-CRAM is sufficiently high that majority implementations are realistic. However, SHE-CRAM array in Fig. 4.2 is limited by the fact that all input operands must be in even columns, and the output must be in an odd column, or vice versa. This affects multi-step operations where some intermediate results, which act as operands for the next step, may be in even columns, while others may be in odd columns. This requires additional steps to move some operands.

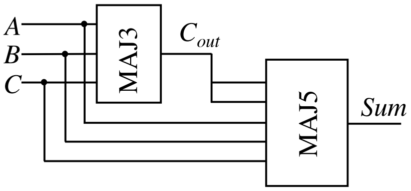


Figure 4.9: FA based on majority logic, where $C_{out} = MAJ3(A, B, C)$ and $Sum = MAJ5(A, B, C, C_{out}, C_{out})$.

Fig. 4.10 shows that the implementation of a majority logic based FA in a row of the SHE-CRAM requires 4 steps. In step 1, $C_{out} \leftarrow MAJ3(A, B, C)$ is calculated: the inputs are in even columns (0, 2, 4) and the output is in odd column 1. In steps 2 and 3, C_{out} is copied, $D \leftarrow BUFFER(C_{out})$, to two different even-numbered columns (6 and 8). Finally, in step 4, with all operands in even-numbered columns, we compute $Sum \leftarrow MAJ5(A, B, C, C_{out}, C_{out})$.

Note that due to the SHE-CRAM structure, C_{out} computed in step 1 cannot be used directly for computation of Sum and must be copied twice to proper locations at Step 2 and Step 3, meaning that this operation requires 4 steps, unlike the STT-CRAM, which would require 3 steps; however, as stated earlier, SHE-CRAM provides better NM than STT-CRAM.

Multibit adders: Using the majority logic based FA, we show the implementation of a 4-bit ripple carry adder (Fig. 4.11), with the computations scheduled as shown in

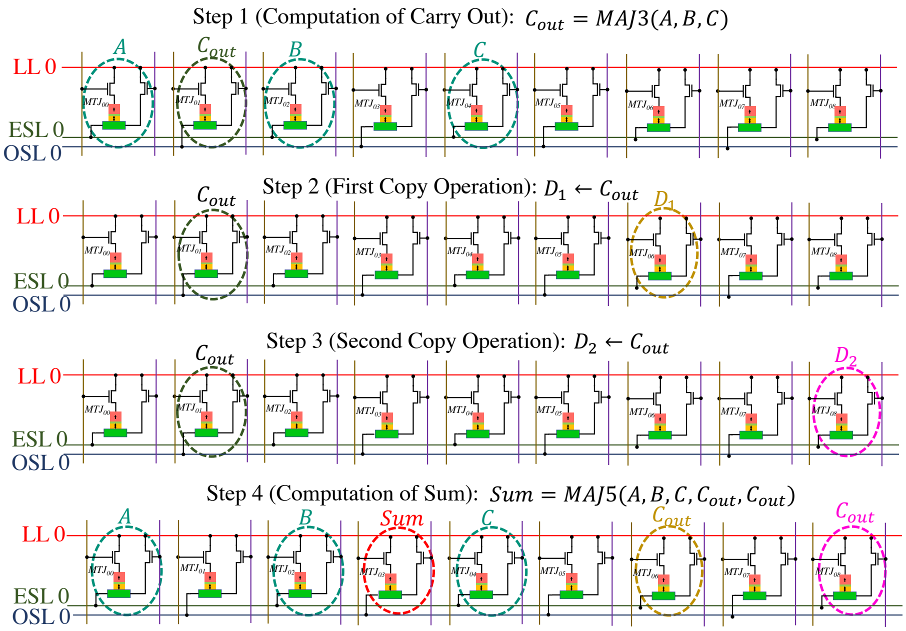

Step 1 (Computation of Carry Out): $C_{out} = MAJ3(A, B, C)$
A
C_{out}
B
C
LL 0
ESL 0
OSL 0
Step 2 (First Copy Operation): $D_1 \leftarrow C_{out}$
C_{out}
D_1
LL 0
ESL 0
OSL 0
Step 3 (Second Copy Operation): $D_2 \leftarrow C_{out}$
C_{out}
D_2
LL 0
ESL 0
OSL 0
Step 4 (Computation of Sum): $Sum = MAJ5(A, B, C, C_{out}, C_{out})$
A
B
Sum
C
C_{out}
C_{out}
LL 0
ESL 0
OSL 0

Figure 4.10: Four required steps for the implementation of the FA based on majority logic in a row.

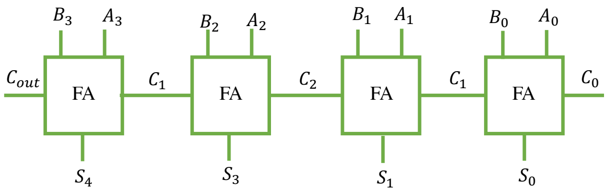


Figure 4.11: 4-bit ripple carry adder using 4 FAs.

Step	1	2	3	4	5	6	7	8	9	10
Row 0	C_1							D_1	D_5	S_0
Row 1		C_1	C_2					D_2	D_6	S_1
Row 2				C_2	C_3			D_3	D_7	S_2
Row 3						C_3	C_{out}	D_4	D_8	S_3

Figure 4.12: Scheduling table for a 4-bit ripple carry adder.

Fig. 4.12. At step 1, C_1 is generated in row 0. At step 2, C_1 is transferred to row 1.

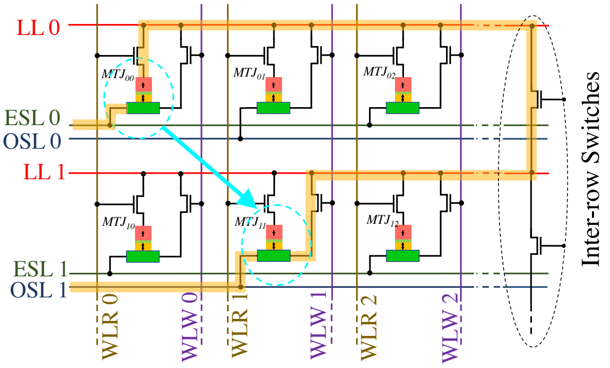


Figure 4.13: Inter-row transfer between cells in two adjacent rows (shown by the blue arrow) using switches inserted between rows. The current path is highlighted in orange.

Similarly, the generated Carrys from the second FA (implemented in row 1) and third FA (implemented in row 2) are transferred to rows 2 and 3 at steps 4 and 6, respectively. Once all Carrys are generated in their corresponding rows, we can copy Carrys twice to proper locations (D_1 to D_8), and then compute Sums (recall that input operands are required to be in all-even or all-odd columns). We transfer the Carry from one row to its adjacent row using inter-row switches (Fig. 4.13).

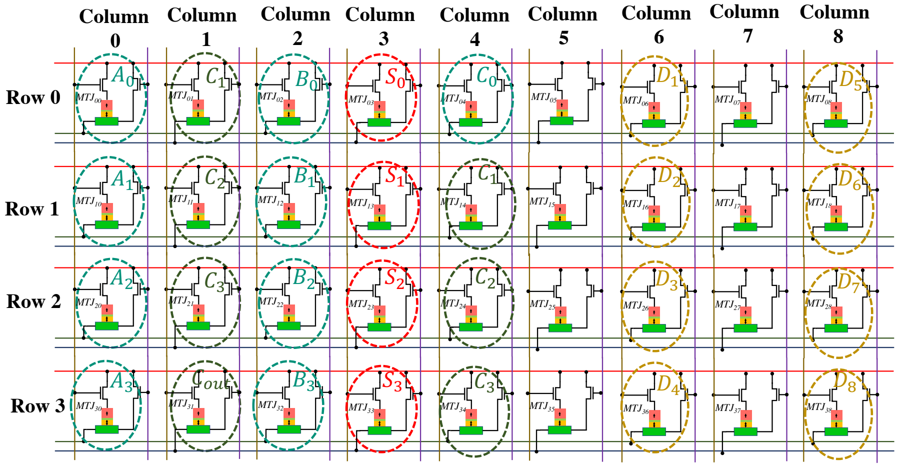


Figure 4.14: Data layout of the SHE-CRAM, implementing the 4-bit ripple carry adder, at the end of step 10.

Table 4.4: Counts of gates and their corresponding energy values for the calculation of the energy required for the implementation of the 4-bit ripple carry adder in the SHE-CRAM.

	BUFFER	/MAJ3	/MAJ5	PRESET	Total Energy (fJ)
Number of gates	11	4	4	19	
Energy/gate (fJ)	4.34	1.76	1.30	3.74	
Total Energy (fJ)	47.74	7.04	5.12	71.06	130.96

Fig. 4.14 shows the data layout of the 4-bit ripple carry adder at the end of step 10. The location of each cell can be specified by (Row number, Column number). Initially, 4-bit numbers $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ are stored in (0 to 3, 0) and (0 to 3, 2), respectively, and Carry-in C_0 is stored in (0, 4). At step 1, C_1 is calculated in (0, 1), and at step 2 it is transferred to (1, 4).

Similarly, C_2 and C_3 are generated and transferred between step 3 and 6. At step 7, C_{out} is calculated in (3, 1). The content of (0 to 3, 1) is then copied to (0 to 3, 6) and (0 to 3, 8) based on the abovementioned schedule. Finally, at step 10, S_0, S_1, S_2, and S_3 are calculated in (0 to 3, 3).

The execution time is determined by counting the number of steps and multiplying them by the logic delay for a majority function, which is dominated by the MTJ switching time. The energy is calculated by considering numbers of gates and their corresponding energy (Table 4.4). The dominant energy component of this implementation is related to the output presetting of gates (see Fig. 4.15).

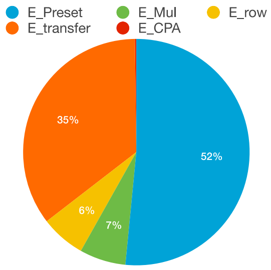


Figure 4.15: Energy distribution for the implementation of 4-bit ripple carry adder using SHE-CRAM. Energy for preset is the dominant component.

More complex building blocks: Similar principles can be used to implement structures such as multipliers and dot products, which can be integrated to implement applications using SHE-CRAM; details are omitted due to space limitations.

4.4 Application-Level Analysis

To benchmark SHE-CRAM performance at the application level, we study its performance when it is deployed on two applications that were analyzed for the STT-CRAM in [2]: (a) 2-D convolution, where a 512×512 image is filtered using a 3×3 filter, and (b) neuromorphic digit recognition using 10K testing images in the MNIST database.

For both applications, we compare the energy and execution time using SHE-CRAM, STT-CRAM, and a near-memory processing (NMP) system (representative of current state-of-the-art). The NMP system places a processor at the periphery of a memory, and is superior to a system in which data is fetched from memory to processor (or coprocessor) [1] [16] [80]. Also, note that in evaluations of STT-CRAM and SHE-CRAM, the effect of peripheral circuitry is considered.

The results of the comparison are presented in Table 4.5. SHE-CRAM outperforms STT-CRAM in both execution time and energy, and both SHE-CRAM and STT-CRAM beat the NMP system in term of energy and execution time. In both applications, SHE-CRAM is at least 4× more energy efficient, and 3× faster than STT-CRAM. For 2-D convolution, SHE-CRAM is over 2000× faster, and 130× more energy-efficient than an NMP system. The corresponding numbers for the neuromorphic application are over 4000× and 190×, respectively.

The improvements in SHE-CRAM over the STT-CRAM can be attributed to the speed and energy-efficiency of the SHE-MTJ device. Note that the ratio of speed improvement is almost the same as the 3× improvement of the SHE-MTJ over the STT-MTJ, but the energy improvement is less than the ratio of STT-MTJ to SHE-MTJ

Table 4.5: Comparison between execution time and energy of NMP, SHE-CRAM, and STT-CRAM. The CMOS-based NMP data is based on the calculations in [2].

Application	Parameters	NMP	STT-CRAM	SHE-CRAM
2-D Convolution	Execution Time	144.4 μs	231 ns	63 ns
	Energy	388.6 μJ	16.5 μJ	2.9 μJ
Digit Recognition	Execution Time	1.96 ms	1105 ns	408 ns
	Energy	2.57 mJ	63.8 μJ	13.5 μJ

switching energy, primarily because of the significant energy overhead of the peripheral driver circuitry of the memory array. Using larger subarrays in the memory can provide up to 25% energy improvements, while degrading the speedup from 3× to just over 2×.

The superiority of both CRAMs over the NMP system can be attributed to the low memory access time of the in-memory computing paradigm, and high levels of available parallelism in CRAM. In contrast, in the NMP system, the energy and execution time consists of two components: (a) fetching data from the memory unit, and (b) processing data in processor units. We can have maximum parallelism in a NMP systems by using multiple processor units and latency hiding techniques, but energy and execution time cost of fetching data from the memory are a severe bottleneck. This bottleneck does not exist in the CRAM due to data locality.

4.5 Conclusion

SHE-CRAM leverages the speed and efficiency of the 3-terminal SHE device, and we demonstrate a new in-memory computing architecture using this device. We propose a design method which contains consideration in device, gate, and functional levels. At the device level, the 3-terminal SHE-MTJ integrated with highly efficient spin-Hall material is served as the unit cell of CRAM. At the gate level, we show that energy and noise margin of implementation of a gate using SHE-CRAM is always superior to those of STT-CRAM. Moreover, we optimize the dimensions of the spin-Hall channel with respect to the noise margin and the implementation energy of a gate. At the functional level, we illustrate how a FA can be implemented in SHE-CRAM, principles that can be extended to more complex structures. Finally, at the application level, we have analyzed the SHE-CRAM performance for 2-D convolution and neuromorphic digit recognition. We show a large improvement in speed and energy over both the STT-CRAM and a NMP system.

Chapter 5

Using 3D XPoint as an In-Memory Computing Accelerator

5.1 Introduction

The substrate that we work on is 3D XPoint [34], a class of memory technology that fills a unique place within the memory hierarchy between solid state storage drive (SSD) and the system main memory. In comparison with the NAND-based SSD (which is the most ubiquitous storage device available today [81]), it has the advantage of being faster, denser, and more scalable. Its nonvolatility differentiates it from competing technologies such as NAND-based SSDs and dynamic random access memories (DRAMs), although NAND-based SSDs are more cost-effective today and DRAMs are faster. Other emerging nonvolatile technologies face limitations: stand-alone PCM must deal with resistance drift, where the cell resistance increases over time [82]; FeFET is handicapped by its large operating voltage and limited endurance [83]; MRAMs require an access transistor (unlike 3D XPoint), leading to a larger cell size than 3D XPoint; ReRAM is not commercially viable to the level of 3D XPoint and MRAM.

The operation and performance of 3D XPoint as a memory unit are discussed in [9, 10, 13, 84, 85]. In our work, rather than focusing again on the memory aspects of 3D XPoint, we explore the possibility of exploiting 3D XPoint arrays to perform in-memory computation. This means not only that 3D XPoint can function as a storage unit, but also that it can perform computation inside its array without the need for the data to leave the array [86]. Therefore, unlike conventional computational systems, the information can be processed locally rather than being sent to a processor through the memory hierarchy.

The analysis in this chapter considers wire non-idealities and physical design of 3D XPoint subarray. We first show the implementation of thresholded matrix-vector multiplication (TMVM), which is a building block for neural networks (NNs) and deep learning applications. Second, using this core operation, we discuss the implementation of a neural network inference engine. Finally, we discuss how to enable 3D XPoint for more complex versions of these implementations (e.g. multi-bit operations and multi-layer NNs).

For in-memory computing platforms, wire resistances are a substantial source of non-ideality that must be taken into account during the implementations [87]. Some works attempt to analyze the parasitic effects of wires but do not consider all contributing factors with realistic layout considerations [73, 74]. In [88], a framework is presented to incorporate the effects of nonidealities in 2D resistive crossbar performance. In [89], an analytical approach is developed to study the effects of the parasitic of wires for the implementation on spintronics computational RAM.

We discuss the feasibility of using 3D XPoint as an in-memory computing engine for neuromorphic applications, and evaluate its performance for MNIST digit recognition. We present novel methods for the implementation of TMVM and NN on 3D XPoint. We use 3D-stacked PCM memory layers in the 3D XPoint subarray to compute and store the computation results entirely inside the array, without sending the data to the periphery of the array. Our method is scalable by using multiple arrays to handle a large computational workload. In addition, multi-bit operations are supported in our methodology. We evaluate a realistic size of the 3D XPoint subarray and metal features (based on the ASAP7 7nm technology [7, 8]) for accurate electrical correctness. We develop a comprehensive method to analyze the impact of wire parasitics of wires in the 3D XPoint subarray and devise a methodology to determine the maximum size of a 3D XPoint subarray that ensures electrically correct operation.

Next, we discuss the structure of 3D XPoint in Section 5.2. In Section 5.3, we describe the implementation of TMVM, and NN. In Section 5.4, we explore the methods for more complex implementations. We develop the models for the effect of wire parasitics in Section 5.5, evaluate the results of our analysis in Section 5.6, and then conclude the chapter in Section 5.7.

5.2 Background

Some recent works study the implementation of logic operations using a 3D crossbar array architecture. In [90], a double layer Pt/HfO_{2-x}/TiN ReRAM crossbar array is used and it is experimentally shown that the array can implement MVM and CNN. Additional peripheral circuitry (e.g., AND gates) is required for obtaining the computation results. In [91], it is shown that stateful logic operation can be performed on a memristive TiO_2-based 3D crossbar array. The adverse effects of wire non-idealities are not incorporated in the implementations. Similarly in [92], the authors map logic operations on a memristive 3D crossbar without considering wire parasitic or technology design rules. In [93], the authors use 3D memristive crossbars for neuromorphic computation. For the implementation of a neuron, additional amplifier circuitry is required at the periphery of the crossbar.

Fig. 5.1 shows the overall structure of a 3D XPoint subarray. A two-level PCM stack is integrated at the top of CMOS peripheral circuitry. The storage device is based on phase-change memory (PCM) technology, which is connected to a compatible ovonic threshold switch (OTS) made of AsTeGeSiN [94–96]. Word lines at the top (WLTs), word lines at the bottom (WLBs), and bit lines (BLs) in the middle provide the current path to each individual memory cell [97]. The compatibility of the junction of PCM and OTS devices is a key factor in allowing access to individual cells without facing sneak path problems [98]. The total number of PCM cells in the 3D XPoint subarray with N_{row} rows and N_{column} columns is $(2N_{row} \times N_{column})$, with half in the top PCM level and half in the bottom PCM level, as shown in the figure.

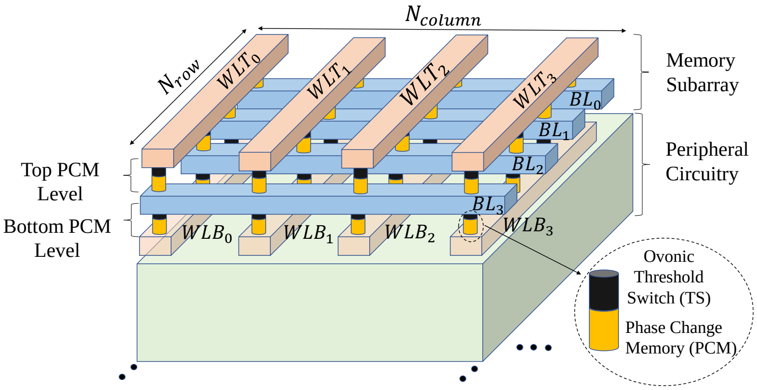


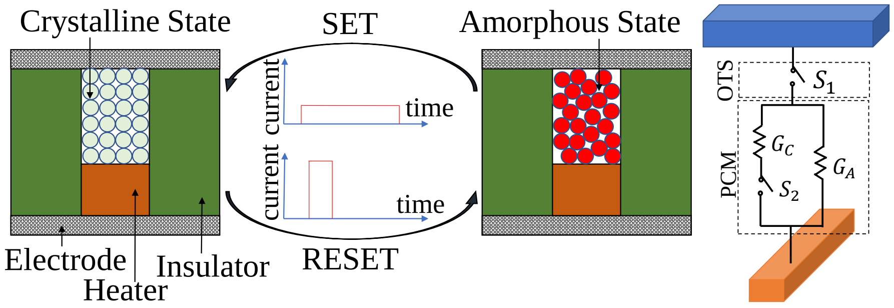
Figure 5.1: The structure of a 3D XPoint subarray. The CMOS peripheral circuitry is located underneath the memory subarray.

PCM is a non-volatile memory technology exploiting GeSbTe (GST) alloys (e.g., $Ge_2Sb_2Te_5$) as the storage medium [99]. PCM has two states: a crystalline phase with high conductance (G_C) and an amorphous phase with low conductance (G_A). The GST alloy transition between amorphous and crystalline states is triggered by changing the temperature level [100, 101]. In early explorations of PCM technology (1970s–early 2000s), the temperature level was changed using a laser source [102]. The state-of-the-art research on PCM is focused on using electrical impulses to change the temperature, and hence the state, of the PCM device by applying an electric current (or voltage) pulse across the PCM device [103].

Fig. 5.2(a) shows that applying a fast high-amplitude current pulse of amplitude I_{RESET} (called the RESET pulse) heats up the GST material to the melting temperature T_{melt} (∼600°C or higher [101]), erasing the previous periodic and ordered atomic arrangement. After quenching, the new disordered atomic structure will be frozen, making the transition from high conductance crystalline state to low conductance amorphous state possible. To change the state of the GST from amorphous to crystalline, a slow, relatively low amplitude current pulse of amplitude I_{SET} (called the SET pulse) must pass through the GST material. The SET current pulse causes the GST material to heat up to crystalline temperature T_{cryst} (∼400°C [101]). Over a long SET time of several tens of nanoseconds, this is a high enough temperature (still lower than T_{melt}) for the reconfiguration and crystallization of the previous amorphous atomic region to the crystalline state. The desirable PCM characteristics are a lower amplitude of the RESET current and a shorter SET time. A RESET current as low as 10μA and a SET time as low as 25ns for individual PCM devices is already demonstrated with sub-20nm scalability, high endurance 10^{12} cycles, and a projected 10-year retention time at 210°C [82].

Fig. 5.2(b) shows the electrical model of PCM cell. The resistance across the PCM cell can be modeled by two voltage controlled switches [13]. Depending on the status of switches S_1 and S_2, different currents flow between two lines connected to the terminals of the PCM cell, determined by G_A and G_C. The ON/OFF states of the memory cell are determined by OTS: If the voltage level across the OTS of a cell is larger than a threshold, the cell is considered to be ON, and it is OFF otherwise. In today's technologies, the OTS conductance for the OFF state is up to $10^8\times$ smaller than for the ON state.

The value stored in the PCM device can represent either logic 1 (crystalline phase) or logic 0 (amorphous phase). Three memory operations available in 3D XPoint: write logic 1 (using the fast high-amplitude SET pulse), write logic 0 (using long low-amplitude RESET pulse), and read. For the memory read operation, since it is undesirable to change the state of the PCM cell, a pulse with relatively very small amplitude will be applied, increasing the temperature slightly above the ambient temperature but below T_{cryst} (and of course T_{melt}).



(a)

(b)

Figure 5.2: PCM model: (a) the transition between amorphous and crystalline phases by applying SET and RESET pulses across a pillar type PCM device, and (b) PCM cell can be modeled using a resistive circuit with two voltage control switches [13].

5.3 Realization of In-Memory Computing

5.3.1 Implementation of TMVM

TMVM is a fundamental step in the implementation of many applications, and is a fundamental computational kernel in machine learning (ML) applications. Using 3D

XPoint as the TMVM computation engine can tremendously decrease the ML computational workload, as the data does not need to leave the 3D XPoint array during the computation.

To show how the first step of a TMVM, let us multiply, without thresholding, matrix $G \in \mathbb{R}^{(N_x+1)\times(N_y+1)}$ and vector $V = [V_0 V_1 V_2 ... V_{N_x}]^T \in \mathbb{R}^{(N_x+1)}$, where G is given by:

$$G = \begin{pmatrix} G_{0,0} & G_{0,1} & ... & G_{0,N_y} \\ G_{1,0} & G_{1,1} & ... & G_{1,N_y} \\ . & . & ... & . \\ . & . & ... & . \\ G_{N_x,0} & G_{N_x,1} & .. & G_{N_x,N_y} \end{pmatrix} \tag{5.1}$$

This computes $O = \left[O_0 O_1 O_2, ... O_{N_y}\right]^T \in \mathbb{R}^{(N_y+1)})$ where each element of vector O is a dot product. For example,

$$O_0 = G_{0,0}V_0 + G_{1,0}V_1 + ... + G_{N_x,0}V_{N_x} \tag{5.2}$$

This is computed in the 3D XPoint array by applying voltages across a set of conductances to produce a current O_0.

Today's PCM cells can only store binary values. Hence, we assume that elements of matrix G and vector V are binary. To implement a neuron-like operation using TMVM, the O_0 value and computed O_i values are followed by a thresholding operation. In (5.2), if the sum of products exceeds the current required to flip the output bit, then logic 1 is stored as the conductance, G_{O_0}, of the PCM cell O_0; otherwise, the stored logic value is 0. Similarly, for other O_is, the values after thresholding are stored as the conductance states, G_{O_i}.

Fig. 5.3(a) shows the implementation of the TMVM on a 3D XPoint subarray with $(2N_{row} \times N_{column})$ PCM cells ($(N_{row} \times N_{column})$ cells each at top PCM level and bottom PCM level) where $N_{column} = N_x + 1$ and $N_{row} = N_y + 1$. For clarity, as compared to

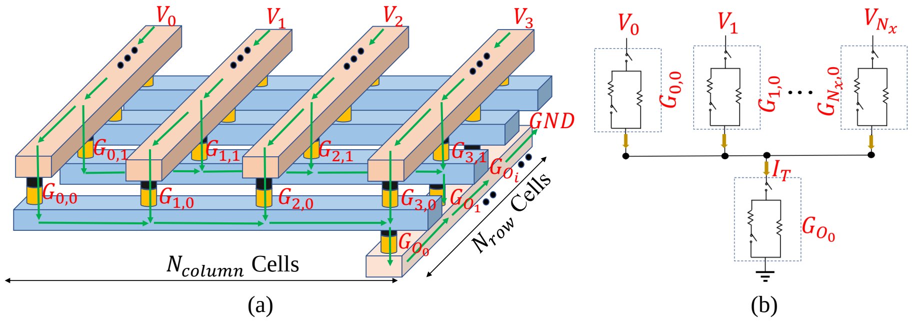


Figure 5.3: (a) Using 3D XPoint as an in-memory computing engine for TMVM of GV. (b) The equivalent circuit model for the implementation of a dot product (to calculate O_0).

Fig. 5.1, only the lines and PCM cells engaged in the computation are shown, and the rest of the lines and the PCM cells (at the bottom) are removed from the figure. All elements of O will be calculated simultaneously and are stored in the same column with N_{row} PCM cells. Considering that today's 3D XPoint cannot store multiple values in a cell, we assume that elements of vector V and G are binary.

The conductances G are first programmed in the top PCM level by memory write operations or by previous computation.

- Before the computation starts, cells that store G_{O_i}s at the bottom are preset to logic 0.

- Then, voltages $V_i, 0 \leq i \leq N_x$ are applied to the word lines WLTs connected to input cells located at top. If V_i represents logic 1, voltage V_{DD} is applied ($V_i \leftarrow V_{DD}$) to the WLT_i and the current that flows through the corresponding input cell is proportional to $G_{0,i}V_{DD}$.

- If V_i represents logic 0, WLT_i is floated ($V_i \leftarrow \mathit{float}$) and no current passes through the corresponding PCM cell.

- The summation of currents (I_T) from input cells flows through the G_{O_0} in a time interval t_{SET}. Based on the values of V_i and $G_{i,0}$, different currents pass through the input cells that store $G_{i,0}$. If $I_T > I_{SET}$, the state of G_{O_0} changes to logic 1. However, we require $I_T < I_{RESET}$ to avoid erroneous computation.

To calculate the minimum and maximum allowable applied voltage (V_{DD}) to the lines, we consider a simplified electrical model for the implementation of a dot product (e.g., for O_0) shown in Fig. 5.3(b). Current I_T can be written as follows

$$I_T = G_{O_0} \frac{\sum_{i=0}^{N_x} V_i G_{i,0}}{\sum_{i=0}^{N_x} G_{i,0} + G_{O_0}} \tag{5.3}$$

When the computation begins, $G_{O_0} \approx G_A$ since the preset is 0, and $I_T(t = 0)$ is small (of the order of few hundred nAs). However, by the passage of time, the amorphous region near the heater in the output PCM starts to turn crystalline, resulting in increasing G_{O_0} (and consequently I_T) and heat (generated by the flow of more electric current). If the applied voltage V_{DD} is large enough to provide a current larger than I_{SET}, crystallization repeats until a threshold point where the whole amorphous region in the output PCM turns into a crystalline region with high conductivity, representing logic 1. On the other hand, the V_{DD} must not be so large that the generated temperature exceeds T_{melt}, causing erroneous computation.

To calculate the V_{DD} range for the accurate implementation of the described dot product, we analyze the two cases corresponding to V_{min} (the minimum acceptable voltage) and V_{max} (the maximum acceptable voltage). For the V_{min} case, we assume that all V_is, and all $G_{i,0}$s represent logic 1, i.e., V_{DD} voltages are applied to all WLTs and the conductances of input cells are in the high conductance state corresponding to G_C. In this case, from (5.3), $I_T = \left(\frac{N_x+1}{N_x+2}\right) G_C V_{DD}$. Since $I_{SET} \leq I_T \leq I_{RESET}$, the V_{min} requirement implies a first constraint, requiring that V_{DD} to lie in the range:

$$\mathcal{R}_1 = \left[\left(\frac{N_x+2}{N_x+1}\right)\left(\frac{I_{SET}}{G_C}\right), \left(\frac{N_x+2}{N_x+1}\right)\left(\frac{I_{RESET}}{G_C}\right)\right] \tag{5.4}$$

For the V_{max} case, all V_is are set to logic 1, while all $G_{i,0}$s are set to logic 0. Since the result of the dot product should be at logic 0, we expect that the preset value of PCM stored O_0 remains intact. At the maximum voltage possible, we hypothetically assume that the applied voltage pulse should be below the level required to change the output state from logic 0 to logic 1, even if conductances of all input cells are G_A. In other words, $I_T = \left(\frac{(N_x+1)G_A G_C}{(N_x+1)G_A+G_C}\right) V_{DD} < I_{SET}$, i.e., the output state cannot be altered. Therefore, the second set of constraints require V_{DD} to lie in the range

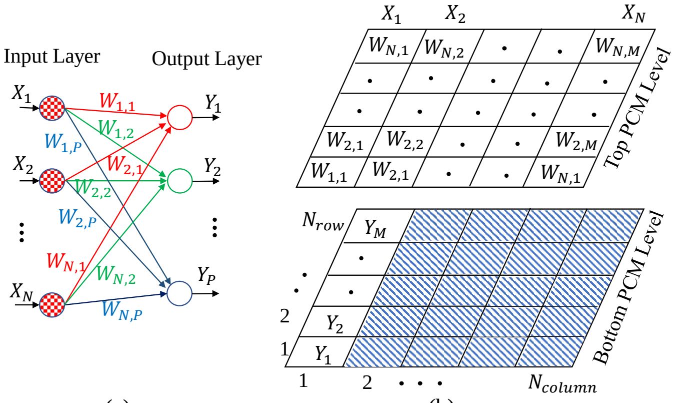
$$\mathcal{R}_2 = \left[0, \left(\frac{(N_x+1)G_A + G_C}{(N_x+1)G_A G_C}\right) I_{SET}\right] \tag{5.5}$$

The acceptable range for V_{DD} is $\mathcal{R}_1 \cap \mathcal{R}_2$. Therefore, the minimum and maximum acceptable voltages are $V_{min} = \min(\mathcal{R}_1)$ and $V_{max} = \min(\max(\mathcal{R}_1), \max(\mathcal{R}_2))$, respectively.

5.3.2 Implementation of NN

Using the TMVM implementation, we implement a neuromorphic inference engine. Fig. 5.4(a) shows a single-layer NN with N inputs and P outputs. Fig. 5.4(b) shows the data layout for the NN implementation on a 3D XPoint subarray. The top PCM cells are allocated for storing the weights ($W_{i,j}$s), similar to $G_{i,j}$s in TMVM that were stored in the top PCM cells, and the bottom PCM cells are allocated for storing the outputs (Y_is), similar to $O_i s$ in TMVM. The inputs (X_is) are applied to WLT_is as voltage pulses (similar to V_is in TMVM).If $N \leq N_{column}$ and $P \leq N_{row}$, then all Y_js can be determined simultaneously in one step. The output elements of the NN can be stored in any column at the bottom (here, we choose column 1). In Fig. 5.4(b), all other cells at the bottom patterned by diagonal stripes are not engaged in the computation of Y_is, i.e., WLBs connected to these cells are floated.

An application for the proposed NN implementation is handwritten digit recognition of MNIST dataset with 10K test images [49]. Analyzing each MNIST test image can



(a) (b)

Figure 5.4: NN implementation: (a) A single-layer neuromorphic inference engine. (b) Data layout for the NN implementation.

be performed using a similar NN shown in Fig. 5.4(a). Here $P = 10$, as in MNIST each image represents a digit (from 0 to 9). In each computational step, $\lfloor \frac{N_{row}}{P} \rfloor$ images can be processed and stored in a column.

5.4 Enabling More Complex Implementations

In this section, we discuss three concepts that enable more complex computations. Then, we provide the implementation of a multi-layer NN as an example. Our approach can also be extended to perform multi-bit computation directly using the principles in [104].

5.4.1 3D XPoint with Four Stacked Levels of PCM Cells

Industry projections show that the next generation of 3D XPoint will have four-level stacked PCM cells [105]. If the number of PCM levels increases, then the volume of stored information per footprint area increases, and more complex implementations are possible. Although a two-level subarray of PCM cells is sufficient to implement any NN (see Section 5.4D), we will illustrate how we can use a four-level subarray of PCM cells to implement a multi-layer NN by exploiting the extra PCM levels. The NN in Fig. 5.5

has three layers: an input layer, a hidden layer, and an output layer. At the top PCM level, the first set of weights are stored. In the next PCM level, the hidden layer data is calculated, and by applying the second set of weights as voltage pulses, we obtain the outputs Y_i of the NN at the third PCM level.

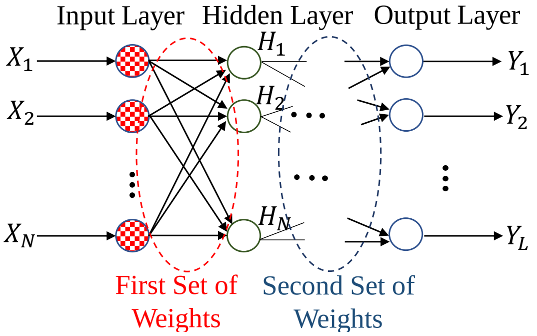
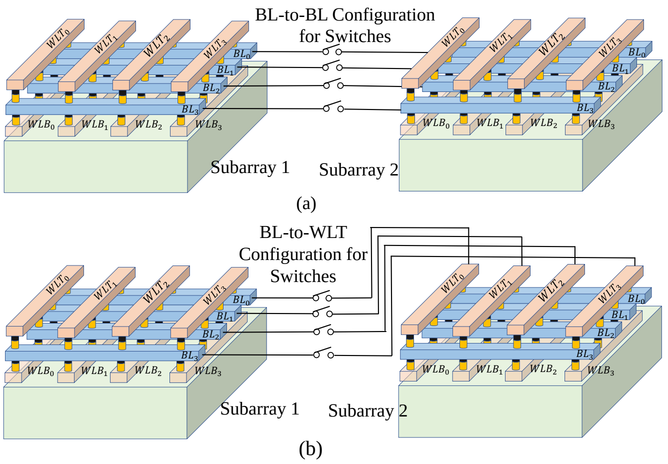


Figure 5.5: Multi-layer NN with an input, hidden, and output layer.

5.4.2 Scalability of 3D XPoint to Large Computations

We can connect multiple 3D XPoint subarrays to create a larger array to handle computations with higher matrix dimensions. In Fig. 5.6(a), switches connect BLs of subarray 1 to those of subarray 2, enabling current flow from the BLs of subarray 1 to those of subarray 2. The WLB of subarray 2 that is scheduled to store the computation results will be connected to ground, while all other WLBs not engaging in the computation (in both subarrays 1 and 2) are floated. Hence, the computation results in subarray 1, are being calculated and stored at the bottom PCM level of subarray 2 (BL-to-WLT). In Fig. 5.6(b), switches connect BLs of the subarray 1 to WLT of subarray 2. In this configuration, the results are being calculated at the top PCM level of subarray 2. The status of lines during the computation for these two configurations are listed in appendix D.



(a)

(b)

Figure 5.6: Two configurations for communication between 3D XPoint subarrays: (a) switches connect BLs of subarray 1 to BLs of Subarray 2, and (b) switches connect BLs of the subarray 1 to WLTs of subarray 2.

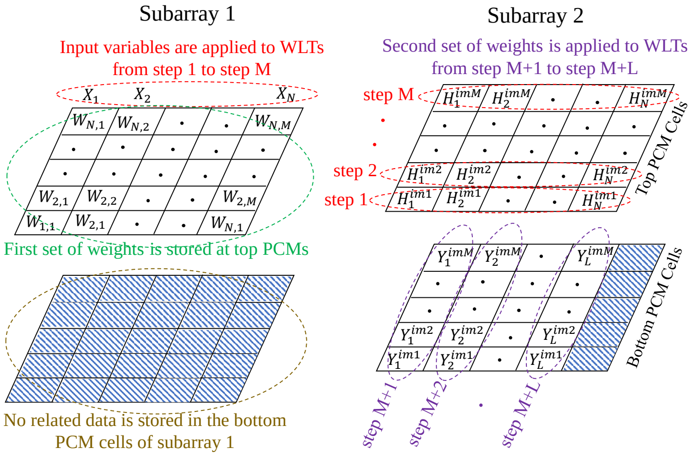


Figure 5.7: Data layout for the implementation of 3-layer NN.

5.4.3 Multi-Layer NN Implementation in a Two-Level PCM Stack

We now illustrate how a multi-layer NN can be implemented using a three-layer PCM stack. As an example, we discuss the implementation of the three-layer NN (shown in Fig. 5.5) using two two-level 3D XPoint subarrays. Let us assume that the NN is required to analyze 10K images of the MNIST dataset. The data layout of this implementation is illustrated in Fig. 5.7 using two subarrays connected with BL-to-WLT configuration (see Fig. 5.6(b)). The first set of weights is stored at the top PCM cells of subarray 1. The inputs $(X_0, X_1, ..., X_N)$ are applied as the voltages to the WLTs of subarray 1. We assume that at each time step, the hidden layer values $(H_1, H_2, ..., H_N)$ for a specific image from MNIST are being processed. For example, the hidden layer values of the second image $(H_1^{im2}, H_2^{im2}, ..., H_N^{im2})$ is calculated in step 2. Assuming that we calculate the hidden layer values of $M(= N_{row})$ images in each set of computation, we require M steps to calculate and store the hidden layer values of M images at the top PCM cells of subarray 2. In each of these steps, the corresponding BL in subarray 2 is connected to GND, and the remaining BLs in subarray 2 are floated. After all hidden layer values are stored at the top PCM cells of subarray 2, we apply the second set of the weights (as voltage pulses) to the WLTs of subarray 2. At each column at the bottom PCM cells of subarray 2, the outputs (Y_is) of M images are calculated and stored.

5.5 Analyzing Interconnect Parasitic Effects

To ensure the electrical correctness of the implementations in in-memory computing platforms, we must consider non-idealities due to wire parasitic effects [87, 89]. As an example, we consider the implementation of a TMVM illustrated in Fig. 5.3(a). In thhe equivalent circuit model shown in Fig. 5.8, the WLTs, BLs, and WLBs have nonzero parasitics that cause a voltage drop in the current path across the 3D XPoint subarray, that may potentially lead to errors in the results of TMVM. Let G_x and G_y be the conductances of the segments of BLs and WLs, respectively. The conductances for WLT and WLB are considered equal (both G_y) due to the symmetry and equal allocation of metal resources to WLTs and WLBs. We use $G_{i,j}$ to denote the conductance of PCM cell (i, j) at the top level, and G_{O_j}s to denote conductances of a column of PCM cells at the bottom level. In the worst case, each row performs an identical operation, and carries an equal current I_{row}. The total voltage drop to the last row is

$$\frac{I_{row}}{G_y} + \frac{2I_{row}}{G_y} + \ldots + \frac{N_{row}I_{row}}{G_y} = \frac{N_{row}(N_{row}+1)I_{row}}{2G_y} \tag{5.6}$$

where the first, second, and last terms on the left side of the equation are for voltage drops of $Segment_{N_{row}}$, $Segment_{N_{row}-1}$, and $Segment_1$, respectively. The voltage drop of the last row increases quadratically with the number of rows, and this causes a significant limit on the accuracy of the implementations [87,89]. Hence, it is important to find the maximum allowable subarray size in which the voltage drop does not impair the electrical of implementations.

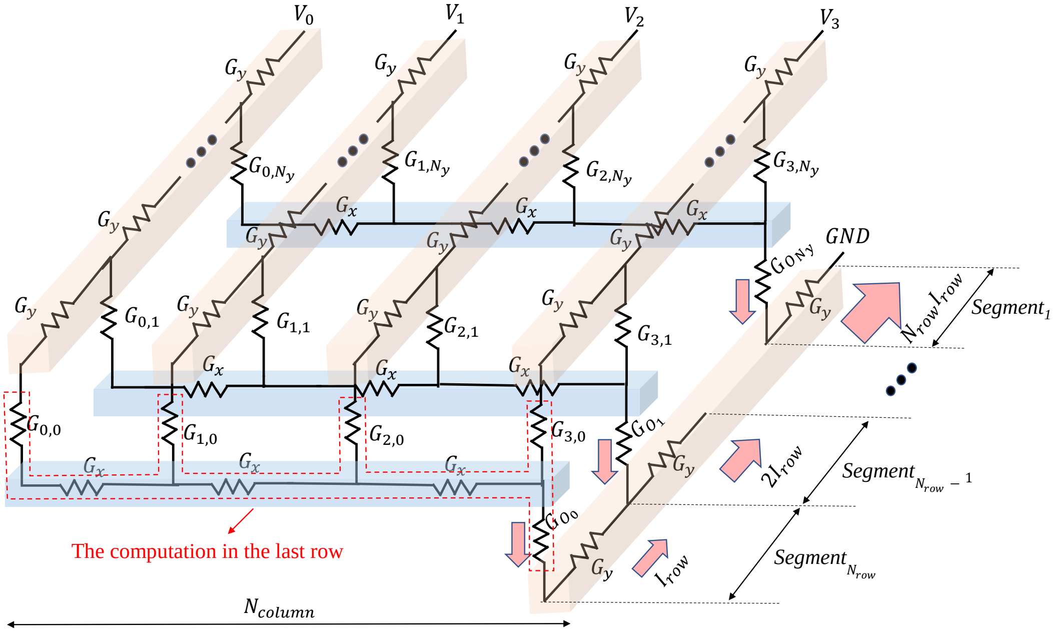


Figure 5.8: The equivalent circuit model for the TMVM implementation with considering wire parasitics.

During the computation, the resistive network shown in Fig. 5.8 can have different configurations based on the applied voltage to WLTs. For example, if $V_2 \leftarrow$ *float*, then all $G_{2,j}$s and their connected parasitics must be removed from the equivalent circuit model of Fig. 5.8. To analyze parasitic effects, we consider the corner case for voltage drop, where only $V_0 \leftarrow V_{DD}$ and the rest of the V_is are floated, resulting in minimum equivalent conductance for inputs and wire parasitics. Moreover, for the corner case, we assume that inputs and outputs are located N_{column} columns away from each other (the farthest possible distance). The value of all inputs assumed to be 1, and therefore, the current flows from the inputs of the TMVM computation must be sufficient enough to change the state of the output of the TMVM computation. An excessive voltage drop across the input and output cells causes a failure in the TMVM implementation discussed earlier.

The rows far from the drivers have larger parasitics between them and the driver. In particular, for the last row (farthest from the driver, see Fig. 5.8), the voltage drop is the worst. If the electrical correctness for the last row does not hold up, the implementations would be unacceptable. We observe the rest of the circuit from the last row and calculate (for the worst case) the Thevenin resistance (R_{th}) and Thevenin voltage (V_{th}) (see Fig. 5.9(a)). We define the Thevenin coefficient, $\alpha_{th} = \frac{V_{th}}{V_{DD}}$, and its value is between 0 and 1. Both R_{th} and α_{th} can be obtained analytically using a recursive approach explained in Appendix C. Both are functions of parameters such as N_{row}, N_{column}, PCM cell width (W_{cell}) and length (L_{cell}) as well as other parameters of PCM and wire devices. Fig. 5.9(b) and (c) shows R_{th} and α_{th} for different N_{row} values. For the smaller N_{row}, PCM resistances are the dominant resistances, and the parasitic effect of wires is minimum. When the N_{row} increases the values of the collective parasitic resistances become comparable to those of PCM devices and hence can degrade the electric correctness of the subarray. The configuration of lines are based on configuration 1 that will be discussed in Table 5.1 in the next Section.

There are negligible parasitics between the first row and the driver, and the voltage range that ensures accuracy of computing in the first row is closer to $[V_{min}, V_{max}]$ (discussed in Section 5.3) than that of the last row. For the last row, values of α_{th} and R_{th}

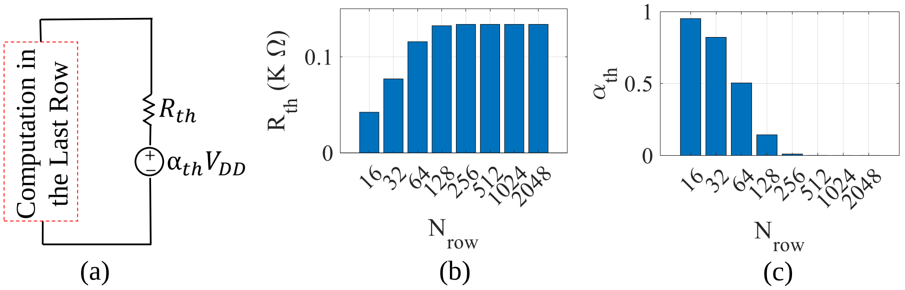


Figure 5.9: (a) Thevenin equivalents can be observed from the last row, (b) effects of N_{row} on R_{th}, (c) and on α_{th}.

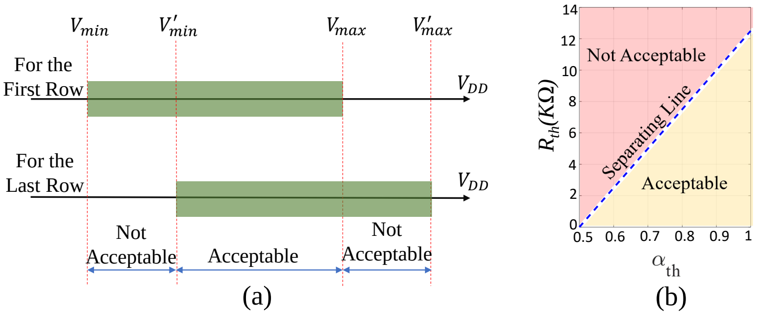


Figure 5.10: (a) Calculated voltage ranges for the first and last rows. (b) Acceptable and unacceptable regions in the (α_{th}, R_{th}) plane.

are significantly affected by parasitics. Let us assume that the new voltage range ensures electrical correctness of the last row is $[V'_{min}, V'_{max}]$. The voltage ranges for the first row and last row are shown in Fig. 5.10(a). We use the voltage ranges of first row and last row as two corner cases (with least and most voltage drops, respectively), and we find a voltage range the satisfy both corner cases; the obtained voltage range guarantees the electrical correctness for intermediate rows as well. The final acceptable voltage range is the overlap between two voltage ranges shown in Fig. 5.10(a), $[V'_{min}, V_{max}]$, ensuring all of the rows from first row to the last row receiving the proper voltage.

The noise margin (NM) in implementations is defined by

$$NM = \frac{V_{max} - V'_{min}}{V_{mid}} \tag{5.7}$$

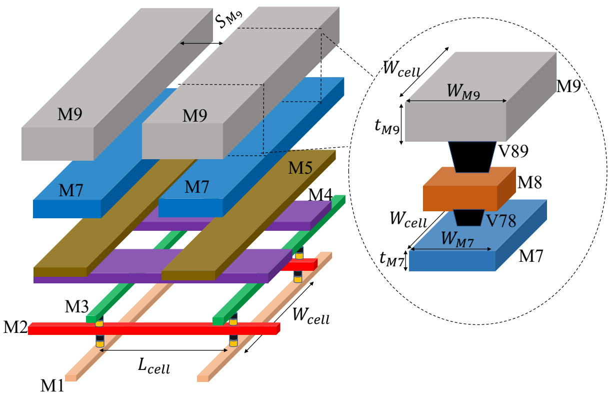


Figure 5.11: Multi-metal layer configuration can be utilized for the design of WLTs, BLs, and WLBs of 3D XPoint subarray.

Table 5.1: Different configurations of metal lines in the 3D XPoint subarray based on ASAP7 design rules.

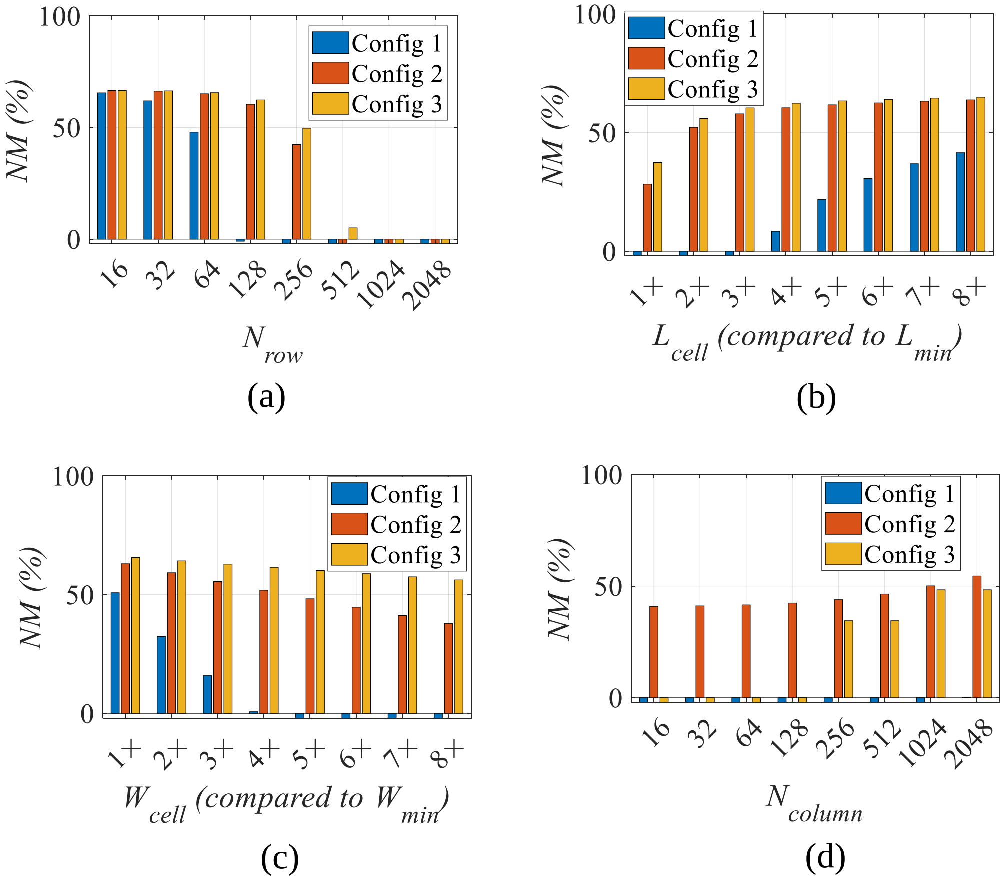
Config	*WLT*	*WLB*	*BL*	$W_{min}\times L_{min}$
1	M3	M1	M2	36nm×36nm
2	M3, M6, M8	M1, M7, M9	M2, M4, M5	48nm×80nm
3	M3, M5, M6, M8	M1, M4, M7, M9	M2	36nm×80nm

where $V_{mid} = (V_{max} + V'_{min})/2$. Clearly, we desire $NM \geq 0$. In Fig. 5.10(b), the acceptable and unacceptable regions in the (α_{th}, R_{th}) plane is shown. The NM on the separating line is 0, above it NM is negative (unacceptable), and below it NM is positive (acceptable). Our goal is to choose wire configurations so that the corresponding (α_{th}, R_{th}) of the design falls into the acceptable region with maximum NM possible.

5.6 Results and Discussion

5.6.1 NM Evaluation

To realistically analyze the effect of the parasitics, we assumed that metal layers in 3D XPoint are constructed based on ASAP7 design rules [7,8] (see Fig. 5.11). We can create



(a)

(b)

(c)

(d)

Figure 5.12: NMs of the three metal line configurations: (a) changing N_{row} (while $N_{column} = 128$, $L_{cell} = 4L_{min}$, and $W_{cell} = W_{min}$), (b) changing L_{cell} (while $N_{column} = 128$, $N_{row} = 128$, and $W_{cell} = W_{min}$), (c) changing W_{cell} (while $N_{column} = 128$, $N_{row} = 64$, and $L_{cell} = 4L_{min}$), and (d) changing N_{column} (while $N_{row} = 256$, $L_{cell} = 4L_{min}$, and $W_{cell} = W_{min}$).

different configurations for allocating metal lines to WLTs, WLBs, and BLs. Table 5.1 lists three possible configurations. In configuration 1, only M1, M2, and M3 (the first three metal lines) in ASAP7 are exploited for 3D XPoint, and they are allocated to WLB, BL, and WLT, respectively. For configurations 2 and 3, we assume that other than M1, M2, and M3, the other metal layers (M4 to M9) can also be allocated to the 3D XPoint lines. In configuration 2, we allocate M4 and M5 to the BLs, and M6 to M9 are allocated equally between WLTs and WLBs. In configuration 3, we assume that all metals from M4 to M9 are allocated equally between WLTs and WLBs; no extra top metal lines are allocated to BLs. We report the minimum cell width (W_{min}) and length (L_{min}) for each configuration based on the minimum required width of a line and space between adjacent lines in each layer. The values of parameters for metal lines and PCM devices are available in appendix D.

NM **improves with increasing** N_{row}: Fig. 5.12(a) shows NMs of different N_{row} values. NM is significantly sensitive to N_{row}. For N_{row} as large as 2048, the implementations are not valid due to excessive voltage drop, and hence negative NM. Configuration 3 provides the best NM, because more metal resources dedicated to WLT and WLB causes smaller parasitics in the current path across rows.

NM **improves with increasing** L_{cell}: Fig. 5.12(b) shows the NMs for different L_{cell}s (for each configuration, values are normalized to L_{min} listed in Table 5.1). By increasing L_{cell}, the width of the WLTs and WLBs increase, decreasing the parasitic resistances related to WLTs and WLBs.

NM **decreases with increasing** W_{cell}: Fig. 5.12(c) shows the NMs for different W_{cell} (for each configuration, values are normalized to W_{min} listed in Table 5.1). By increasing W_{cell}, the length of the WLTs and WLBs increase, and consequently, parasitics related to WLTs and WLBs considerably increase. Therefore, for all cases, smaller W_{cell} causes larger NM.

NM **remains unchanged with increasing** N_{column}: Fig. 5.12(d) shows that the increase in N_{column} does not affect NM significantly. By increasing N_{column}, parasitics of BLs increase. However, since the BL resistances are in series with those of PCM devices with orders of magnitude larger resistance, the increase in BL resistance does not affect NM.

5.6.2 Implementing NNs on the 3D XPoint Substrate

We list the performance of various 3D XPoint subarrays of various sizes for the digit recognition of MNIST dataset in Table 5.2. Each MNIST test image is scaled to 11×11 as in [106], a transformation that maintains 91% recognition accuracy and reduces computation. We use configuration 3 in all cases, as it provides the best NM among all alternatives. For the smallest subarray with size 64×128, NM is the maximum among all cases. For the largest subarray with size 1024×1024, we increase L_{cell} by 2.6× (compare to that of 64×128 subarray) to decrease the parasitics of lines. Consequently, we achieve acceptable NM of 34.5%. With this relatively large subarray, we have more parallelism that allow to process a larger number of MNIST images in each computational step, reducing the total execution time (17× faster than that of 64×128 subarray). The energy per image is similar for all cases because the subarray sizes listed in Table 5.2 are large enough to allow fully processing an 11×11 MNIST image locally without extra data movement between subarrays or peripheral circuitry.

The table also shows the impact of 10% process variation on NM. In the second to the last column, interconnect resistances are changed by 10%. For small arrays, the change in NM is negligible because the resistance of the PCM element and OTS dominate wire resistance. For larger arrays, this effect becomes more noticeable (but is still not significant). In the last column, PCM device parameters as well as interconnect resistances are simultaneously varied by 10% to find the worst NM due to variations of all parameters. NMs for all arrays are still positive and acceptable (between 12.4% for the largest subarray and 46.6% for the smallest subarray).

Table 5.2: Evaluation of different subarray sizes for digit recognition application.

Subarray Size	Cell Size (nm×nm)	#Image per Step	Energy per Image	Subarray Area (μm^2)	Execution Time (μs)	V_{max} (V)	V'_{min} (mV)	NM	NM w/ 10% Var.
64×128	36×240	6	21.5pJ	62.9	133.3	1.25	636.2	65.1%	64.9%
128×256	36×320	12	21.5pJ	335.5	66.7	1.25	650.6	63.1%	62.7%
256×512	36×400	25	20.7pJ	1677.7	32.0	1.25	681.0	58.9%	58.1%
512×1024	36×480	51	20.3pJ	8053.0	15.7	1.25	732.5	52.2%	50.8%
1024×2048	36×640	102	20.3pJ	42949.6	7.8	1.25	882.2	34.5%	31.5%

5.7 Conclusion

We have presented methods for the implementations of TMVM, NN, and 2D convolution on 3D XPoint. To ensure the accuracy of the implementations, we considered wire parasitics in our implementations. We have demonstrated that interconnect parasitics have a significant effect on the implementations performance and have developed a comprehensive model for analyzing this impact. Using this methodology, we have developed guidelines for the 3D XPoint Subarray size and configurations based on ASAP7 technology design rules. We used different size 3D XPoint subarrays for digit recognition of MNIST dataset. Using the our methodology methodology, we design a relatively large subarray of 2 Mb with acceptable NM of 34.5%, providing the opportunity for processing more images per step without any energy overhead.

Chapter 6

Conclusion

The first contribution of this work is a method for building an in-memory computing platform, incorporating considerations in device, gate, and functional block levels. After evaluating the alternatives, we chose the Spin Transfer Torque (STT) Magnetic Tunnel Junction (MTJ) as the memory device technology, for its high endurance, robustness, and nonvolatility (ability to store data when the power is turned off). Next, a method is developed for building the reconfigurable memory array, which can act as a memory or a computational unit. In addition, all peripheral circuitry is designed for an STT-CRAM in-memory platform. Methods are developed for implementing and scheduling elementary computational operations, such as additions, multiplications, and dot-products, on the CRAM array. The set of underlying computational operations was carefully chosen so that it would not only leverage the underlying STT-RAM technology, but also map on to big data computations. It is demonstrated that for a set of data-intensive applications in image processing and artificial intelligence, CRAM shows energy improvements of $> 2500\times$ over todayâs technologies.

The effectiveness of the CRAM can be attributed to (1) the large number of operations that can be simultaneously performed within the CRAM array, leading to massive parallelism and short computation times, and (2) the removal of the data access bottleneck by obviating the need to move data outside the memory. Analysis techniques are created for a more detailed analysis of the practical limitations that might prevent the CRAM structures from being successful. Specifically, it is discovered that wire resistance in the lines that provide the power supply to the CRAM array could be significantly limit the size of the CRAM array, thus limiting the available parallelism and overall efficiency. A method is devised to choose and engineer an optimal physical size and chip layout for the in-memory platform that maximizes the array size while guaranteeing robust, error-free operation inside the platform. It is shown that through careful design, the gains predicted earlier can be delivered.

A novel architecture for CRAM-based computational platform based on SHE MTJ is proposed to improve the performance. SHE MTJ device has three terminals (as compared to the two terminals of the STT MTJ), and required a complete redesign of the fundamental memory âbitcellâ as well as the memory array. New scheduling tables and techniques for the implementation of computational blocks (e.g., adder and multiplier) are proposed. For two specific applications (2-D convolution and neuromorphic digit recognition), we show that SHE-CRAM is 3× faster and has over 4× lower energy than a prior STT-based CRAM implementation, and is over 2000× faster and at least 130 × more energy-efficient than state-of-the-art near-memory processing

We show that how 3D XPoint memory arrays can be used as in-memory computing accelerators. We first show that thresholded matrix-vector multiplication (TMVM), the fundamental computational kernel in many applications including machine learning, can be implemented within a 3D XPoint array without requiring data to leave the array for processing. Using the implementation of TMVM, we then discussed the implementation of a binary neural inference engine. We discussed the application of the core concept to address issues such as system scalability, where we connected multiple 3D XPoint arrays, and power integrity, where we analyzed the parasitic effects of metal lines on noise margins. To assure power integrity within the 3D XPoint array during this implementation, we carefully analyzed the parasitic effects of metal lines on the accuracy of the implementations. We quantified the impact of parasitics on limiting the size and configuration of a 3D XPoint array, and estimated the maximum acceptable size of a 3D XPoint subarray.

Bibliography

[1] S. W. Keckler, W. Dally, B. Khailany, M. Garland, and D. Glasco. GPUs and the future of parallel computing. *IEEE Micro*, 31:7–17, Nov 2011.

[2] M. Zabihi, Z. Chowdhury, Z. Zhao, U. Karpuzcu, J.-P. Wang, and S. Sapatnekar. In-memory processing on the spintronic CRAM: From hardware design to application mapping. *IEEE Transactions on Computers*, 68:1159–1173, Aug 2019.

[3] M. DC, R. Grassi, J. Chen, M. Jamali, D. Hickey, D. Zhang, Z. Zhao, H. Li, P. Quarterman, Y. Lv, M. Li, A. Manchon, K. Mkhoyan, T. Low, and J.-P. Wang. Room-temperature high spin orbit torque due to quantum confinement in sputtered BixSe films. *Nature Materials*, 17, 09 2018.

[4] G. Prenat, K. Jabeur, P. Vanhauwaert, G. D. Pendina, F. Oboril, R. Bishnoi, M. Ebrahimi, N. Lamard, O. Boulle, K. Garello, J. Langer, B. Ocker, M.-C. Cyrille, P. Gambardella, M. Tahoori, and G. Gaudin. Ultra-fast and high-reliability SOT-MRAM: From cache replacement to normally-off computing. *IEEE Transactions on Multi-Scale Computing Systems*, 2(1):49–60, 2016.

[5] C. Zhang, S. Fukami, H. Sato, F. Matsukura, and H. Ohno. Spin-orbit torque induced magnetization switching in nano-scale Ta/CoFeB/MgO. *Applied Physics Letters*, 107(1):012401, 2015.

[6] A. Hirohata, H. Sukegawa, H. Yanagihara, I. Zutic, T. Seki, S. Mizukami, and R. Swaminathan. Roadmap for emerging materials for spintronic device applications. *IEEE Transactions on Magnetics*, 51(10):1–11, October 2015.

[7] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric. ASAP7: A 7-nm FinFET predictive process design kit. *Microelectronics Journal*, 53:105–115, July 2016.

[8] L. T. Clark, V. Vashishtha, D. M. Harris, S. Dietrich, and Z. Wang. Design flows and collateral for the ASAP7 7nm FinFET predictive process design kit. In *Proceedings of the IEEE International Conference on Microelectronic Systems Education*, pages 1–4, June 2017.

[9] K. Son, K. Cho, S. Kim, S. Park, G. Park, S. Kim, T. Shin, and J. Kim. Modeling and verification of 3-dimensional resistive storage class memory with high speed circuits for core operation. In *IEEE Asia-Pacific Microwave Conference*, pages 694–696, March 2019.

[10] K. Son, K. Cho, S. Kim, S. Park, D. H. Jung, J. Park, G. Park, S. Kim, T. Shin, Y. Kim, and J. Kim. Signal integrity design and analysis of 3-D X-Point memory considering crosstalk and IR drop for higher performance computing. *IEEE Transactions on Components and Packaging Technologies*, 10(5):858–869, May 2020.

[11] D. Fujiki, X. Wang, A. Subramaniyan, and R. Das. *In-/Near-Memory Computing.* Morgan Claypool, San Rafael, CA, USA, 2021.

[12] L. Dadda. Some schemes for parallel multipliers. *Alta Frequenza*, 34:349–356, March 1965.

[13] K. Son, K. Cho, S. Kim, G. Park, K. Song, and J. Kim. Modeling and signal integrity analysis of 3D XPoint memory cells and interconnections with memory size variations during read operation. In *IEEE Symposium on Electromagnetic Compatibility, Signal Integrity and Power Integrity*, pages 223–227, July 2018.

[14] A. McAfee, E. Brynjolfsson, and T. H. Davenport. Big data: The management revolution. *Harvard Business Review*, October 2012.

[15] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, and S. Karp. Exascale computing study: Technology challenges in achieving exascale systems. *DARPA Information Processing Techniques Office (IPTO) Sponsored Study*, 2008. www.cse.nd.edu/Reports/2008/TR-2008-13.pdf.

[16] M. Horowitz. Computing's energy problem (and what we can do about it). In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 10–14, February 2014.

[17] J. P. Wang and J. D. Harms. General structure for computational random access memory (CRAM), December 29 2015. US Patent 9,224,447 B2.

[18] M. Zabihi, Z. Zhao, Z. I. Chowdhury, S. Resch, M. DC, T. Peterson, U. R. Karpuzcu, J.-P. Wang, and S. S. Sapatnekar. True in-memory computing with the CRAM: From technology to applications. In *Proceedings of the ACM Great Lakes Symposium on VLSI*, pages 379–379, 2019.

[19] M. Zabihi, Z. Zhao, Z. I. Chowdhury, M. Resch, T. Peterson, Mahendra DC, , J.-P. Wang, U. R. Karpuzcu, and S. S. Sapatnekar. Using spin-Hall MTJs to build an energy-efficient in-memory computation platform. In *Proceedings of the IEEE International Symposium on Quality Electronic Design*, pages 52–57, March 2019.

[20] H. Cilasun, S. Resch, Z. I. Chowdhury, E. Olson, M. Zabihi, Z. Zhao, T. Peterson, K. K. Parhi, J.-P. Wang, S. S. Sapatnekar, and U. R. Karpuzcu. Spiking neural networks in spintronic computational RAM. *ACM Transactions on Architecture and Code Optimization*, 2021.

[21] S. Resch, S. K. Khatamifard, Z. I. Chowdhury, M. Zabihi, Z. Zhao, H. Cilasun, J.-P. Wang, S. S. Sapatnekar, and U. R. Karpuzcu. MOUSE: Inference in non-volatile memory for energy harvesting applications. In *Annual IEEE/ACM International Symposium on Microarchitecture*, pages 400–414, October 2020.

[22] J.-P. Wang, S. S. Sapatnekar, U. R. Karpuzcu, Z. Zhao, M. Zabihi, S. Resch, Z. I. Chowdhury, and T. Peterson. Computational random access memory (CRAM) based on spin-orbit torque devices, September 2020. US Patent 20,200,279,597 A1.

[23] H. Cilasun, S. Resch, Z. I. Chowdhury, E. Olson, M. Zabihi, Z. Zhao, T. Peterson, J.-P. Wang, S. S. Sapatnekar, and U. Karpuzcu. CRAFFT: High resolution FFT accelerator in spintronic computational RAM. In *ACM/IEEE Design Automation Conference*, pages 1–6, October 2020.

[24] Z. I. Chowdhury, M. Zabihi, S. K. Khatamifard, Z. Zhao, S. Resch, M. Razaviyayn, J.-P. Wang, S. S. Sapatnekar, and U. R. Karpuzcu. A DNA read alignment accelerator based on computational RAM. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 6(1):80–88, 2020.

[25] Z. I. Chowdhury, S. K. Khatamifard, Z. Zhao, M. Zabihi, S. Resch, M. Razaviyayn, J.-P. Wang, S. Sapatnekar, and U. R. Karpuzcu. Spintronic in-memory pattern matching. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 5(2):206–214, 2019.

[26] S. Resch, S. K. Khatamifard, Z. I. Chowdhury, M. Zabihi, Z. Zhao, J.-P. Wang, S. S. Sapatnekar, and U. R. Karpuzcu. PIMBALL: Binary neural networks in spintronic memory. *ACM Transactions on Architecture and Code Optimization*, 16(4), October 2019.

[27] S. Resch, S. K. Khatamifard, Z. I. Chowdhury, M. Zabihi, Z. Zhao, H. Cilasun, J.-P. Wang, S. S. Sapatnekar, and U. R. Karpuzcu. A machine learning accelerator in-memory for energy harvesting, August 2019. arXiv: 1908.11373[cs].

[28] Z. I. Chowdhury, M. Zabihi, S. K. Khatamifard, Z. Zhao, S. Resch, M. Razaviyayn, J.-P. Wang, S. S. Sapatnekar, and U. R. Karpuzcu. Computational RAM to accelerate string matching at scale, December 2018. arXiv:1812.08918.

[29] H. Cilasun, S. Resch, Z. I. Chowdhury, M. Zabihi, Z. Zhao, T. Peterson, J.-P. Wang, S. S. Sapatnekar, and U. R. Karpuzcu. Seeds of SEED: H-CRAM: In-memory homomorphic search accelerator using spintronic computational RAM. In *International Symposium on Secure and Private Execution Environment Design*, pages 70–75, 2021.

[30] Z. I. Chowdhury, S. Resch, H. Cilasun, Z. Zhao, M. Zabihi, S. S. Sapatnekar, J.-P. Wang, and U. R. Karpuzcu. *CAMeleon: Reconfigurable B(T)CAM in Computational RAM.* June 2021.

[31] H. Cilasun, S. Resch, Z. I. Chowdhury, E. Olson, M. Zabihi, Z. Zhao, T. Peterson, K. K. Parhi, J.-P. Wang, S. S. Sapatnekar, and U. R. Karpuzcu. An inference and learning engine for spiking neural networks in computational RAM (CRAM), June 2020. arXiv:2006.03007.

[32] D. Fujiki, S. Mahlke, and R. Das. In-memory data parallel processor. *ACM Special Interest Group on Programming Languages*, 53(2):1–14, March 2018.

[33] M. Imani, S. Gupta, and T. Rosing. Digital-based processing in-memory: A highly-parallel accelerator for data intensive applications. In *Proceedings of the International Symposium on Memory Systems*, pages 38–40, September 2019.

[34] Y. LeCun. 3D XPoint technology. `https://www.intel.com/content/www/us/en/architecture-and-technology/intel-micron-3d-xpoint-webcast.html`.

[35] X. Liu, M. Mao, B. Liu, H. Li, Y. Chen, B. Li, Y. Wang, H. Jiang, M. Barnell, Q. Wu, and J. Yang. Reno: A high-efficient reconfigurable neuromorphic computing accelerator design. In *Proceedings of the ACM/ESDA/IEEE Design Automation Conference*, 2015.

[36] L. Fick, D. Blaauw, D. Sylvester, S. Skryniarz, M. Parikh, and D. Fick. Analog in-memory subthreshold deep neural network accelerator. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 1–4, July 2017.

[37] J.-P. Wang, S. S. Sapatnekar, C. H. Kim, P. Crowell, S. Koester, S. Datta, K. Roy, A. Raghunathan, X. S. Hu, M. Niemier, A. Naeemi, C.-L. Chien, C. Ross, and R. Kawakami. A pathway to enable exponential scaling for the beyond-CMOS era. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 1–6, June 2017.

[38] Z. Chowdhury, S. K. Khatamifard, M. Zabihi, J. D. Harms, Y. Lv, A. P. Lyle, J.-P. Wang, S. Sapatnekar, and U. Karpuzcu. Efficient in-memory processing using spintronics. *IEEE Computer Architecture Letters*, 17(1):42–46, June 2018.

[39] J. Kim, P. A. Crowell, S. J. Koester, S. S. Sapatnekar, J.-P. Wang, and C. H. Kim. Spin-based computing: Device concepts, current status, and a case study on a high-performance microprocessor. *Proceedings of the IEEE*, 103(1):106–130, 2015.

[40] F. Ren and D. Markovic. True energy-performance analysis of the MTJ-based logic-in-memory architecture (1-bit full adder). *IEEE Transactions on Electron Devices*, 57(5):1023–1028, 2010.

[41] A. Lyle, J. Harms, S. Patil, X. Yao, D. J. Lilja, and J.-P. Wang. Direct communication between magnetic tunnel junctions for nonvolatile logic fanout architecture. *Applied Physics Letters*, 97(152504), 2010.

[42] G. Jan, L. Thomas, S. Le, Y.-J. Lee, H. Liu, J. Zhu, R.-Y. Tong, K. Pi, Y.-J. Wang, D. Shen, R. He, J. Haq, J. Teng, V. Lam, K. Huang, T. Zhong, T. Torng, and P.-K. Wang. Demonstration of fully functional 8Mb perpendicular STT-MRAM chips with sub-5ns writing for non-volatile embedded memories. In *Proceedings of the IEEE International Symposium on VLSI Technology*, 2014.

[43] H. Maehara, K. Nishimura, Y. Nagamine, K. Tsunekawa1, T. Seki, H. Kubota, A. Fukushima, K. Yakushiji, K. Ando, and S. Yuasa. Tunnel magnetoresistance above 170% and resistance-area product of $1\Omega(\mu\mathrm{m})^2$ attained by in situ annealing of ultra-thin MgO tunnel barrier. *Applied Physics Express*, 4(03300), 2011.

[44] H. Noguchi, K. Ikegami, K. Kushida, K. Abe, S. Itai, S. Takaya, N. Shimomura, J. Ito, A. Kawasumi, H. Hara, and S. Fujita. 3.3ns-access- time 71.2μW/MHz 1Mb embedded STT-MRAM using physically eliminated read-disturb scheme and normally-off memory architecture. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 1–3, March 2015.

[45] S. Ikeda, J. Hayakawa, Y. Ashizawa, Y. M. Lee, K. Miura, H. Hasegawa, M. Tsunoda, F. Matsukura, and H. Ohno. Tunnel magnetoresistance of 604% at 300K by suppression of Ta diffusion in CoFeB / MgO/ CoFeB pseudo-spin-valves annealed at high temperature. *Applied Physics Letters*, 93(8(082508)), 2008.

[46] H. Almasi, M. Xu, Y. Xu, T. Newhouse-Illige, and W. G. Wang. Effect of Mo insertion layers on the magnetoresistance and perpendicular magnetic anisotropy in Ta/CoFeB/MgO junctions. *Applied Physics Letters*, 109(3(032401)), 2016.

[47] H. M. Martin. Threshold logic for integrated full adder and the like, 1971. US Patent 3,609,329.

[48] E. E. Swartzlander. Recent results in merged arithmetic. In *SPIE Proceedings*, volume 3461, pages 576–583, 1998.

[49] Y. LeCun. The MNIST database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`.

[50] M. Liu, L. R. Everson, and C. H. Kim. A sable time-based integrate-and-fire neuromorphic core with brain-inspired leak and local lateral inhibition capabilities. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, 2017.

[51] J. Jeddeloh and B. Keeth. Hybrid memory cube new DRAM architecture increases density and performance. In *Proceedings of the IEEE International Symposium on VLSI Technology*, pages 87–88, June 2012.

[52] X. Dong, C. Xu, Y. Xie, and N. Jouppi. NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(7):994–1007, July 2012.

[53] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. CACTI 6.0: A tool to model large caches. Technical Report HPL-2009-85, HP Laboratories, 2009.

[54] H. S. Stone. A logic-in-memory computer. *IEEE Transactions on Computers*, C-19(1):73–78, January 1970.

[55] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick. A case for intelligent RAM. *IEEE Micro*, 17(2):34–44, 1997.

[56] S. Rixner, W. J. Dally, U. J. Kapasi, B. Khailany, A. Lopez-Lagunas, P. R. Mattson, and J. D. Owens. A bandwidth-efficient architecture for media processing. In *IEEE International Symposium on Microarchitecture*, pages 3–13, December 1998.

[57] Y. Kang, W. Huang, S.-M. Yoo, D. Keen, Z. Ge, V. Lam, P. Pattnaik, and J. Torrellas. FlexRAM: Toward an advanced intelligent memory system. In *Proceedings of the IEEE International Conference on Computer Design*, pages 192–201, 1999.

[58] J. B. Brockman, P. M. Kogge, T. L. Sterling, V. Freeh, and S. K. Kuntz. Microservers: a new memory semantics for massively parallel computing. In *Proceedings of the 16th International Conference on Supercomputing*, 1999.

[59] J. T. Pawlowski. Hybrid memory cube (HMC). In *Proceedings of the IEEE HotChips Symposium*, 2011.

[60] J. Macri. AMD's next generation GPU and high bandwidth memory architecture: FURY. In *Proceedings of the IEEE HotChips Symposium*, 2015.

[61] R. Nair, S. F. Antao, C. Bertolli, P. Bose, J. R. Brunheroto, T. Chen, C. Y. Cher, C. H. A. Costa, J. Doi, C. Evangelinos, B. M. B. M. Fleischer, T. W. Fox, D. S. Gallo, L. Grinberg, J. A. Gunnels, A. C. Jacob, P. Jacob, H. M. Jacobson, T. Karkhanis, C. Kim, J. H. Moreno, J. K. O'Brien, M. Ohmacht, Y. Park, D. A. Prener, B. S. Rosenburg, K. D. Ryu, O. Sallenave, M. J. Serrano, P. D. M. Siegl, K. Sugavanam, and Z. Sura. Active memory cube: A processing-in-memory

architecture for exascale systems. *IBM Journal of Research and Development*, 59(2/3):17:1–17:14, 2015.

[62] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski. TOP-PIM: Throughput-oriented programmable processing in memory. In *Proceedings of the International Symposium on High-performance Parallel and Distributed Computing*, pages 85–98, 2014.

[63] J. Ahn, S. Yoo, O. Mutlu, and K. Choi. PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture. In *Proceedings of the ACM International Symposium on Computer Architecture*, pages 336–348, June 2015.

[64] S. Jeloka, N. B. Akesh, D. Sylvester, and D. Blaauw. A 28 nm configurable memory (TCAM/BCAM/SRAM) using push-rule 6T bit cell enabling logic-in-memory. *IEEE Journal of Solid-State Circuits*, 51(4):1009–1021, April 2016.

[65] J. Draper, J. Chame, M. Hall, C. Steele, T. Barrett, J. LaCoss, J. Granacki, J. Shin, C. Chen, C. W. Kang, I. Kim, and G. Daglikoca. The architecture of the DIVA processing-in-memory chip. In *Proceedings of the 16th International Conference on Supercomputing*, pages 14–25, 2002.

[66] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry. Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology. In *IEEE International Symposium on Microarchitecture*, 2017.

[67] A. Matsunaga, J. Hayakawa, S. Ikeda, K. Miura, T. Endoh, H. Ohno, and T. Hanyu. MTJ-based nonvolatile logic-in-memory circuit, future prospects and issues. In *Proceedings of Design, Automation & Test in Europe*, pages 433–435,

June 2009.

[68] S. G. Ramasubramanian, R. Venkatesan, M. Sharad, K. Roy, and A. Raghunathan. SPINDLE: SPINtronic Deep Learning Engine for large-scale neuromorphic computing. In *Proceedings of the ACM International Symposium on Low Power Electronics and Design*, 2014.

[69] S. Li, C. Xu, Q. Zhou, J. Zhao, Y. Lu, and Y. Xie. Pinatubo: a processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *Proceedings of the ACM/ESDA/IEEE Design Automation Conference*, 2016.

[70] W. Kang, H. Wang, Z. Wang, Y. Zhang, and W. Zhao. In-memory processing paradigm for bitwise logic operations in STT-MRAM. *IEEE Transactions on Magnetics*, 2017.

[71] S. Angizi, Z. He, N. Bagherzadeh, and D. Fan. Design and evaluation of a spintronic in-memory processing platform for non-volatile data encryption. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017. (in press).

[72] N. Talati, S. Gupta, P. Mane, and S. Kvatinsky. Logic design within memristive memories using Memristor Aided loGIC (MAGIC). *IEEE Transactions on Nanotechnology*, 15(4):635–650, July 2016.

[73] Y. Jeong, M. A. Zidan, and W. D. Lu. Parasitic effect analysis in memristor-array-based neuromorphic systems. *IEEE Transactions on Nanotechnology*, 17(1):184–193, Jan 2018.

[74] Naveen Murali G., F. Lalchhandama, K. Datta, and I. Sengupta. Modelling and simulation of non-ideal MAGIC NOR gates on memristor crossbar. In *Proceedings of the International Symposium on Embedded Computing and System Design*, pages 124–128, Dec 2018.

[75] J.-P. Wang, S. S. Sapatnekar, C. H. Kim, P. Crowell, S. Koester, S. Datta, K. Roy, A. Raghunathan, X. S. Hu, M. Niemier, A. Naeemi, C.-L. Chien, C. Ross, and R. Kawakami. A pathway to enable exponential scaling for the beyond-CMOS era. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 1–6, June 2017.

[76] A. Shafaei, Y. Wang, and M. Pedram. Low write-energy STT-MRAMs using FinFET-based access transistors. In *Proceedings of the IEEE International Conference on Computer Design*, pages 374–379, Oct. 2014.

[77] L. Liu, C.-F. Pai, Y. Li, H.-W. Tseng, D. Ralph, and R. Buhrman. Spin-torque switching with the giant spin hall effect of tantalum. *Science (New York, N.Y.)*, 336:555–8, 05 2012.

[78] S. Fukami, C. Zhang, S. DuttaGupta, A. Kurenkov, and H. Ohno. Magnetization switching by spin-orbit torque in an antiferromagnet-ferromagnet bilayer system. *Nature Materials*, 15(5):535–541, May 2016.

[79] Z. Zhao, A. K. Smith, M. Jamali, and J.-P. Wang. External-field-free spin hall switching of perpendicular magnetic nanopillar with a dipole-coupled composite structure, 2017, 1603.09624.

[80] J. Jeddeloh and B. Keeth. Hybrid memory cube new dram architecture increases density and performance. In *2012 Symposium on VLSI Technology (VLSIT)*, pages 87–88, 2012.

[81] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy. Design tradeoffs for SSD performance. In *USENIX 2008 Annual Technical Conference*, page 57â70, June 2008.

[82] M. Le Gallo and A. Sebastian. An overview of phase-change memory device physics. *Journal of Physics D Applied Physics*, 53(21):213002, May 2020.

[83] S. Beyer, S. Dunkel, M. Trentzsch, J. MÃ¼ller, A. Hellmich, D. Utess, J. Paul, D. Kleimaier, J. Pellerin, S. MÃ¼ller, J. Ocker, A. Benoist, H. Zhou, M. Mennenga, M. Schuster, F. Tassan, M. Noack, A. Pourkeramati, F. Muller, M. Lederer, T. Ali, R. Hoffmann, T. Kampfe, K. Seidel, H. Mulaosmanovic, E. T. Breyer, T. Mikolajick, and S. Slesazeck. FeFET: A versatile CMOS compatible device with game-changing potential. In *Proceedings of the IEEE International Memory Workshop (IMW)*, 2020.

[84] Q. Lou, W. Wen, and L. Jiang. 3DICT: A reliable and QoS capable mobile process-in-memory architecture for lookup-based CNNs in 3D XPoint ReRAMs. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 1–8, Jan. 2018.

[85] J. Yang, L. Bingzhe, and D. Lilja. Exploring performance characteristics of the optane 3D XPoint storage technology. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 5(1), Feb. 2020.

[86] M. Zabihi, S. Resch, H. Cilasun, Z. I. Chowdhury, Z. Zhao, U. R. Karpuzcu, J.-P. Wang, and S. S. Sapatnekar. Exploring the feasibility of using 3-D XPoint as an in-memory computing accelerator. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 7(2):88–96, 2021.

[87] D. Ielmini and G. Pedretti. Device and circuit architectures for in-memory computing. *Advanced Intelligent Systems*, 2(7):2000049–1–2000040–19, May 2020.

[88] S. Jain, A. Sengupta, K. Roy, and A. Raghunathan. RxNN: A framework for evaluating deep neural networks on resistive crossbars. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(2):326–338, 2021.

[89] M. Zabihi, A. K. Sharma, M. G. Mankalale, Z. I. Chowdhury, Z. Zhao, S. Resch, U. R. Karpuzcu, J. Wang, and S. S. Sapatnekar. Analyzing the effects of interconnect parasitics in the STT CRAM in-memory computational platform. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 6(1):71–79, June 2020.

[90] Y. Kim, J. Kim, S. S. Kim, Y. J. Kwon, G. S. Kim, J. W. Jeon, D. E. Kwon, J. H. Yoon, and C. S. Hwang. Kernel application of the stacked crossbar array composed of self-rectifying resistive switching memory for convolutional neural networks. *Advanced Intelligent Systems*, 2(2):1900116, 2020.

[91] N. Xu, T. G. Park, H. J. Kim, X. Shao, K. J. Yoon, T. H. Park, L. Fang, K. M. Kim, and C. S. Hwang. A stateful logic family based on a new logic primitive circuit composed of two antiparallel bipolar memristors. *Advanced Intelligent Systems*, 2(1):1900082, 2020.

[92] N. G. Murali, P. S. Vardhan, F. Lalchhandama, K. Datta, and I. Sengupta. Mapping of boolean logic functions onto 3d memristor crossbar. In *Proceedings of the International Conference on VLSI Design*, pages 500–501, 2019.

[93] B. R. Fernando, Y. Qi, C. Yakopcic, and T. M. Taha. 3d memristor crossbar architecture for a multicore neuromorphic system. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1–8, 2020.

[94] M. Lee, D. Lee, H. Kim, H. Choi, J. Park, H. G. Kim, Y. Cha, U. Chung, I. Yoo, and K. Kim. Highly-scalable threshold switching select device based on chaclogenide glasses for 3D nanoscaled memory arrays. In *IEEE International Electronic Devices Meeting*, pages 2.6.1–2.6.3, Dec. 2012.

[95] G. W. Burr, R. S. Shenoy, K. Virwani, P. Narayanan, A. Padilla, B. Kurdi, and H. Hwang. Access devices for 3D crosspoint memory. *Journal of Vacuum Science & Technology B*, 32(4):040802–1–040802–23, July 2014.

[96] DerChang Kau, S. Tang, I. V. Karpov, R. Dodge, B. Klehn, J. A. Kalb, J. Strand, A. Diaz, N. Leung, J. Wu, Sean Lee, T. Langtry, Kuo-wei Chang, C. Papagianni, Jinwook Lee, J. Hirst, S. Erra, E. Flores, N. Righos, H. Castro, and G. Spadini. A stackable cross point phase change memory. In *IEEE International Electronic Devices Meeting*, pages 1–4, March 2009.

[97] W. Chien, C. Yeh, R. L. Bruce, H. Cheng, I. T. Kuo, C. Yang, A. Ray, H. Miyazoe, W. Kim, F. Carta, E. Lai, M. J. BrightSky, and H. Lung. A study on OTS-PCM pillar cell for 3-D stackable memory. *IEEE Transactions on Electron Devices*, 65(11):5172–5179, Nov. 2018.

[98] L. Shi, G. Zheng, B. Tian, B. Dkhil, and C. Duan. Research progress on solutions to the sneak path issue in memristor crossbar arrays. *Nanoscale Advances*, 2:1811–1827, March 2020.

[99] J. Y. Wu, M. Breitwisch, S. Kim, T. H. Hsu, R. Cheek, P. Y. Du, J. Li, E. K. Lai, Y. Zhu, T. Y. Wang, H. Y. Cheng, A. Schrott, E. A. Joseph, R. Dasaka, S. Raoux, M. H. Lee, H. L. Lung, and C. Lam. A low power phase change memory using thermally confined TaN/TiN bottom electrode. In *IEEE International Electronic Devices Meeting*, pages 3.2.1–3.2.4, Jan. 2011.

[100] H. . P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson. Phase change memory. *Proceedings of the IEEE*, 98(12):2201–2227, Dec. 2010.

[101] G. W. Burr, M. J. Brightsky, A. Sebastian, H. Cheng, J. Wu, S. Kim, N. E. Sosa, N. Papandreou, H. Lung, H. Pozidis, E. Eleftheriou, and C. H. Lam. Recent progress in phase-change memory technology. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 6(2):146–162, June 2016.

[102] M. Wuttig and N. Yamada. Phase-change materials for rewriteable data storage. *Nature Materials*, 6(11):824â832, Nov. 2007.

[103] Q. Zheng, Y. Wang, and J. Zhu. Nanoscale phase-change materials and devices. *Journal of Physics D: Applied Physics*, 50(24):243002, May 2017.

[104] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *isca*, pages 27–39, August 2016.

[105] 3D XPoint with four-level PCM cells. https://www.anandtech.com/show/15972/intel-previews-4layer-3d-xpoint-memory-for-secondgeneration-optane-ssds.

[106] M. Liu, L. R. Everson, and C. H. Kim. A scalable time-based integrate-and-fire neuromorphic core with brain-inspired leak and local lateral inhibition capabilities. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 1–4, July 2017.

Appendix A

Thevenin Models of Current Paths in STT CRAM

Thevenin Model for 1-Input Gates

We derive recursive expressions for R_{th} and α_{th} for each row in the CRAM array. The exposition here derives an expression for V_{th}, and α_{th} is trivially obtained from Eq. (3.6).

Within the footprint area of a CRAM cell, we define R_y, R_x, and R_{Via} as lumped resistances for BSL segment, LL segment, and vias, respectively. Fig. C.1 shows the equivalent simplified circuit of the path for the implementation of BUFFER (or NOT) gates on CRAM rows. Row i is separated from its predecessor by resistances R_y at each end, and is connected through R_{via} to an input MTJ cell, represented by $R_i^{MTJ_1}$ and R_T. The input cell is connected to the output cell, d_{column} rows away, through a resistance R_x, and the output cell is represented by a transistor resistance R_T in series with a preset MTJ resistance, R_{MTJ_2}. The resistances in the last row are rearranged to create a two-port structure consisting of the MTJ resistances, so that the rest of the network can be modeled using Thevenin Equivalents (R_{th} and V_{th}).

Depending on the states of the inputs in different rows, which are application-dependent, the resistances of the MTJs, and hence the Thevenin parameters, change and typically vary in different rows. To provide a robust design, we consider the worst case in which the combination of the values of inputs in different rows results in the worst voltage drop across the MTJs of the last row. This worst case corresponds to the worst-case current, which is drawn when all inputs MTJs in all rows are in the parallel state, creatingÂ $N_{row} - 1$ paths with the lowest possible resistance possible between input and output BSLs.

As explained in Section 3.4.4, the BSL and LL lines have a multi-layer structure. The metal layer resistances are considered to be in parallel, and R_y and R_x are expressed by:

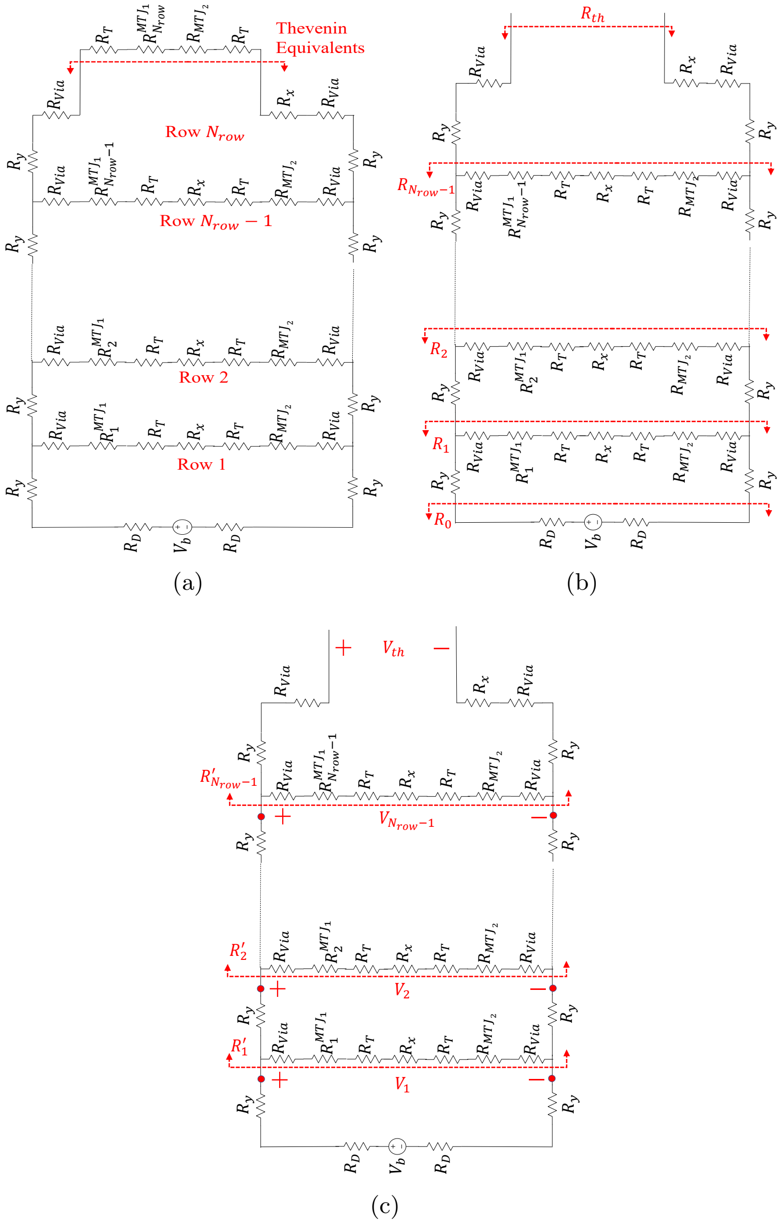
$$R_y^{-1} = R_{M_3}^{-1} + R_{M_5}^{-1} + R_{M_7}^{-1} + R_{M_9}^{-1} \tag{A.1}$$

$$R_x^{-1} = d_{column}^{-1} \left(R_{M_2}^{-1} + R_{M_4}^{-1} \right) \tag{A.2}$$

where d_{column} is the number of wire segments between the input and output columns. The resistance, R_{M_k}, is given by: $R_{M_k} = \frac{\rho_{M_k} L_{M_k}}{t_{M_k} W_{M_k}}$, where ρ_{M_k}, L_{M_k}, t_{M_k}, and W_{M_k} are, respectively, the resistivity, length, thickness, and width in metal layer k. The equivalent resistance of the vias, R_{Via}, depends on the configuration of the CRAM cell: a larger CRAM cell contains more vias in its footprint area, as a consequence of which R_{Via} is smaller. The number of vias between two metal layers in the footprint area of a CRAM cell can be calculated based on the via characteristics in Table D.3, as the parallel resistance of the available number of vias.

The above equivalent resistive introduces a minor simplification because the parallel wires do not coincide at a single point but the vias are a small distance apart. HSPICE simulations show that causes less than 0.5% error.

To calculate R_{th} and V_{th}, we derive recursive expressions. For conciseness, we define



(a)

(b)

(c)

Figure A.1: (a) The circuit model for 1-input gates, showing the observation point for calculating Thevenin equivalents; Notations used in the chain of rows for defining the (b) Thevenin resistance (R_{th}), and (c) Thevenin voltage (V_{th}).

the resistance, R_{row_i}, of row i as:

$$R_{row_i} = 2(R_{Via} + R_T) + R_x + R_i^{MTJ1} + R_{MTJ2} \tag{A.3}$$

The input logic value depends on the application, i..e, R_i^{MTJ1} can be either R_P and R_{AP}. For each gate, the output resistance is a known preset value: for a Buffer (NOT) gate implemented across all rows, the preset is 1 (0). Therefore, R_{MTJ2} for the Buffer and NOT gates are R_{AP} and R_P, respectively.

We can obtain R_{th}, using the notations in Fig. C.1(b), as:

$$R_{th} = 2(R_y + R_{Via}) + R_x + R_{N_{row}-1} \tag{A.4}$$

where $R_{N_{row}-1}$ is calculated using the recursive expression:

$$R_i = \frac{R_{row_i}(R_{i-1} + 2R_y)}{R_{row_i} + R_{i-1} + 2R_y} \tag{A.5}$$

The base case corresponds to the driver row that precedes the first row, and is $R_0 = 2R_D$, as seen in Fig. C.1(a).

To compute V_{th}, as illustrated in Fig. C.1(c), we first compute the intermediate variable R'_j, which corresponds to the effective downstream resistance (away from the source) seen from node j. The computation proceeds in a recursive fashion from the last row towards the first as:

$$R'_{j-1} = \frac{R_{row_{j-1}}(R'_j + 2R_y)}{R_{row_{j-1}} + R'_j + 2R_y} \tag{A.6}$$

with the base case $R'_{N_{row}-1} = R_{row_{N_{row}-1}}$.

Having computed R'_j, we may now compute $V_{th} = V_{N_{row}}$, using a recursive computation on V_i:

$$V_j = \frac{R'_j}{2R_y + R'_j} V_{j-1} \tag{A.7}$$

in which $2 \leq j \leq N_{row} - 1$ and the base case is:

$$V_1 = \frac{R'_1}{R'_1 + 2R_y + 2R_D} V_b \tag{A.8}$$

Thevenin Model for N-input Gates

The 1-input model can easily be extended for the case where the number of inputs $N > 1$. For this case, we have N columns of input MTJs that connect to an output MTJ in each row: the worst case corresponds to the scenario where all inputs are adjacent to each other and d_{column} columns away from the output. As a simplification, we assume that all units are equally far and that the resistance to the output for each is R_x: this is reasonable because the horizontal resistance between adjacent columns is negligible. In this case, in column i, N parallel structures, each consisting of series connections of R_y, R_{via}, $R_i^{MTJ_1}$, and R_T, connect through R_x to the output cell, modeled as a series connection of R_T and R_{MTJ_2}. We generalize Equation (C.1) to

$$R_{Row_i} = \left(1 + \frac{1}{N}\right)(R_{Via} + R_T) + R_x + \frac{R_i^{MTJ1}}{N} + R_{MTJ2} \tag{A.9}$$

Proceeding similarly to the 1-input case, we generalize Equation (C.2) to compute R_{th} as:

$$R_{th} = \left(1 + \frac{1}{N}\right)(R_y + R_{Via}) + R_x + R_{N_{row}-1} \tag{A.10}$$

where $R_{N_{row}-1}$ can be obtained using the recursion:

$$R_i = \frac{R_{Row_i}(R_{i-1} + \left(1 + \frac{1}{N}\right) R_y)}{R_{Row_i} + R_{i-1} + \left(1 + \frac{1}{N}\right) R_y} \tag{A.11}$$

where $1 \leq i \leq N_{row} - 1$ and the base case is $R_1 = \left(1 + \frac{1}{N}\right) R_D$, corresponding to the fact that each input line is driven by a source V_b with a series resistance R_D to the first via.

Similarly, one can recursively compute V_{th}. Analogously to Equation (C.4), we first compute R'_j recursively, from the last row to the first, as:

$$R'_{j-1} = \frac{R_{Row_{j-1}}(R'_j + \left(1 + \frac{1}{N}\right) R_y)}{R_{Row_{j-1}} + R'_j + \left(1 + \frac{1}{N}\right) R_y)} \tag{A.12}$$

where the base case is $R_{N_{row}-1} = R_{Row_{N_{row}-1}}$.

We can then compute $V_{th} = V_{N_{row}}$ recursively using the following recursion for V_j:

$$V_j = \frac{R'_j}{\left(1 + \frac{1}{N}\right) R_y + R'_j} V_{j-1} \tag{A.13}$$

in which $2 \leq j \leq N_{row} - 1$ and the base case is:

$$V_1 = \frac{R'_1}{R'_1 + \left(1 + \frac{1}{N}\right) R_y + \left(1 + \frac{1}{N}\right) R_D} V_b \tag{A.14}$$

Appendix B

STT CRAM Parasitics

MTJ Parameters

We consider two sets of models for MTJs: (a) today's MTJ where parameters are typical values for today's STT MTJ technology, (b) advanced MTJ in which parameters are corresponding to the realistic near-future process. With the fast development of MTJ technology, only considering today's MTJ in our analysis could soon be obsolete. Moreover, analyzing the method for advanced MTJ technology provide a superior understanding of future bottlenecks, design issues, and opportunities. The parameters for today's and advanced near-future technologies are listed in Table B.1.

Interconnect specifications for ASAP7

The interconnect specifications are listed in Table D.2, which shows the metal thickness (t_M) and resistivity (ρ_M), the minimum line spacing (S_{min}), minimum line width (W_{min}), and Table D.3, which shows the via parameters [7, 8].

Table B.1: MTJ specifications [2, 6]

Parameters	Today's MTJ	Advanced MTJ
MTJ type	Interfacial PMTJ	Interfacial PMTJ
Material	CoFeB/MgO/CoFeB	CoFeB(SAF)/MgO/CoFeB
MTJ diameter	45nm	10nm
TMR	133% [42]	500%
RA product	$5\Omega\mu\text{m}^2$	$1\Omega\mu\text{m}^2$ [43]
J_c	$3.1 \times 10^6\text{A/cm}^2$	10^6A/cm^2
I_c	50μA	0.79μA
t_{wr}	3ns [44]	1ns [42]
R_P	3.15KΩ	12.73KΩ
R_{AP}	7.34KΩ	76.39KΩ

Table B.2: Specification of metal layers in ASAP7 [7, 8]

Metal	t_M	S_{min}	W_{min}	ρ_M
M1(V)	36nm	18nm	18nm	43.2Ω.nm
M2(H)	36nm	18nm	18nm	43.2Ω.nm
M3(V)	36nm	18nm	18nm	43.2Ω.nm
M4(H)	48nm	24nm	24nm	36.9Ω.nm
M5(V)	48nm	24nm	24nm	36.9Ω.nm
M6(H)	64nm	32nm	32nm	32.0Ω.nm
M7(V)	64nm	32nm	32nm	32.0Ω.nm
M8(H)	80nm	40nm	40nm	28.8Ω.nm
M9(V)	80nm	40nm	40nm	28.8Ω.nm

Table B.3: Specification of vias in ASAP7 [7, 8]

Via	R_V	Via Size	Minimum Spacing
V12 (M1 and M2)	17Ω	18nm×18nm	18nm
V23 (M2 and M3)	17Ω	18nm×18nm	18nm
V34 (M3 and M4)	17Ω	18nm×18nm	18nm
V45 (M4 and M5)	12Ω	24nm×24nm	33nm
V56 (M5 and M6)	12Ω	24nm×24nm	33nm
V67 (M6 and M7)	8Ω	32nm×32nm	45nm
V78 (M7 and M8)	8Ω	32nm×32nm	45nm
V89 (M8 and M9)	6Ω	40nm×40nm	57nm

Appendix C

Thevenin Model for TMVM in 3D XPoint

We derive recursive expressions for calculating R_{th} and V_{th} of a $(N_{row} \times N_{column})$ subarray of 3D XPoint.

Within the footprint area of a cell $(W_{cell} \times L_{cell})$, we define G_y (representing the conductance for WLT and WLB segments) and G_x (representing the conductance of BL segment). Fig. C.1 shows the equivalent simplified circuit model for the implementation TMVM in the corner case. Row i is separated from its predecessor by conductance G_y at each end. The input cell is connected to the output cell, N_{column} columns away. The conductances in the last row are rearranged to create a two-port structure consisting of the PCM conductances so that the rest of the network can be modeled using Thevenin Equivalents (R_{th} and V_{th}).

For configuration 1 (listed in Table 5.1), $G_y = G_{M_1} = G_{M_3}$ (assuming similar wire conductance for WLTs and WLBs) and $G_x = G_{M_2}$ in which the conductance, G_{M_k}, is given by: $G_{M_k}^{-1} = \frac{\rho_{M_k} L_{M_k}}{t_{M_k} W_{M_k}}$, where ρ_{M_k}, L_{M_k}, t_{M_k}, and W_{M_k} are, respectively, the resistivity, length, thickness, and width in metal layer k (see the Supplementary

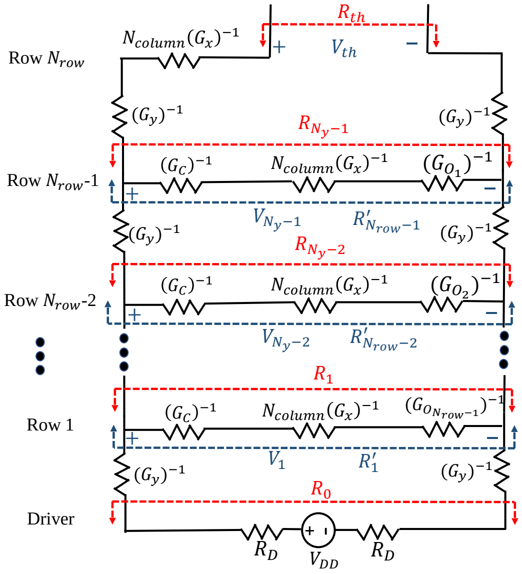


Figure C.1: Notations used for calculating Thevenin resistance (R_{th}) and Thevenin voltage (V_{th}) shown on the circuit model for the implementation of TMVM in the worst case scenario.

Material). For configurations 2 and 3, the equivalent conductance of the wire segment must be calculated based on the multi-metal layer configuration of a given segment. For example, in configuration 2, G_y (representing a segment conductance of WLT) is obtained by $G_y = G_{M_3} + G_{M_6} + G_{M_8}$.

To calculate R_{th} and V_{th}, we derive recursive expressions. For conciseness, we define the resistance, R_{row_i}, of row i as:

$$R_{row_i} = N_{column}\left(G_x\right)^{-1} + \left(G_C\right)^{-1} + \left(G_{O_{N_{row}-i}}\right)^{-1} \tag{C.1}$$

We can obtain R_{th}, using the notations in Fig. C.1, as:

$$R_{th} = 2\left(G_y\right)^{-1} + N_{column}\left(G_x\right)^{-1} + R_{N_{row}-1} \tag{C.2}$$

where $R_{N_{row}-1}$ is calculated using the recursive expression:

$$R_i = \left(R_{row_i}\right) || \left(R_{i-1} + 2\left(G_y\right)^{-1}\right) \tag{C.3}$$

The base case corresponds to the driver row that precedes the first row, and is $R_0 = 2R_D$, as seen in Fig. C.1.

To compute V_{th}, as illustrated in Fig. C.1, we first compute the intermediate variable R'_j, which corresponds to the effective downstream resistance (away from the source) seen from node j. The computation proceeds in a recursive fashion from the last row towards the first as:

$$R'_{j-1} = \left(R_{row_{j-1}}\right) || \left(R'_j + 2\left(G_y\right)^{-1}\right) \tag{C.4}$$

with the base case $R'_{N_{row}-1} = R_{row_{N_{row}-1}}$. Having computed R'_j, we may now compute $V_{th} = V_{N_{row}}$, using a recursive computation on V_i:

$$V_j = \frac{R'_j}{2\left(G_y\right)^{-1} + R'_j} V_{j-1} \tag{C.5}$$

in which $2 \leq j \leq N_{row} - 1$ and the base case is:

$$V_1 = \frac{R'_1}{R'_1 + 2\left(G_y\right)^{-1} + 2R_D} V_b \tag{C.6}$$

Appendix D

TMVM in 3D XPoint

PCM parameters

The conductance values of parameters in a PCM cell listed in Table D.1. In this work, we adopt the RESET current (I_{RESET}) of 100μA with RESET Time (t_{RESET}) of 15ns, and SET time (t_{SET}) of 80ns with the assumption of SET current (I_{SET}) of 50μA ($= \frac{I_{RESET}}{2}$) [9, 10].

Table D.1: PCM cell parameters and values [9, 10]

Parameters	Description	Value
G_A	PCM conductance in the amorphous state	660 nΩ^{-1}
G_C	PCM conductance in the crystalline state	160$\mu\Omega^{-1}$
S_1	Voltage control switch for OTS	100nΩ^{-1}(<0V) and 10Ω^{-1}(>0.3V)
S_2	Voltage control switch for PCM in crystalline state	10Ω^{-1}(<0.8V) and 100nΩ^{-1}(>1V)

Interconnect specifications for ASAP7

The interconnect specifications are listed in Table D.2, which shows the metal thickness (t_M) and resistivity (ρ_M), the minimum line spacing (S_{min}), minimum line width (W_{min}), and Table D.3, which shows the via parameters [7, 8].

Table D.2: Specification of metal layers in ASAP7 [7, 8]

Metal	t_M	S_{min}	W_{min}	ρ_M
M1(V)	36nm	18nm	18nm	43.2Ω.nm
M2(H)	36nm	18nm	18nm	43.2Ω.nm
M3(V)	36nm	18nm	18nm	43.2Ω.nm
M4(H)	48nm	24nm	24nm	36.9Ω.nm
M5(V)	48nm	24nm	24nm	36.9Ω.nm
M6(H)	64nm	32nm	32nm	32.0Ω.nm
M7(V)	64nm	32nm	32nm	32.0Ω.nm
M8(H)	80nm	40nm	40nm	28.8Ω.nm
M9(V)	80nm	40nm	40nm	28.8Ω.nm

Table D.3: Specification of vias in ASAP7 [7, 8]

Via	R_V	Via Size	Minimum Spacing
V12 (M1 and M2)	17Ω	18nm×18nm	18nm
V23 (M2 and M3)	17Ω	18nm×18nm	18nm
V34 (M3 and M4)	17Ω	18nm×18nm	18nm
V45 (M4 and M5)	12Ω	24nm×24nm	33nm
V56 (M5 and M6)	12Ω	24nm×24nm	33nm
V67 (M6 and M7)	8Ω	32nm×32nm	45nm
V78 (M7 and M8)	8Ω	32nm×32nm	45nm
V89 (M8 and M9)	6Ω	40nm×40nm	57nm

Status of lines during communications between subarrays

The status of 3D XPoint lines during the communications with each other is listed in Table D.4.

Table D.4: Status of 3D XPoint lines for two different configurations.

Line	Subarray	Configuration	
		BL-to-BL	BL-to-WLT
WLTs	1	V_is are applied	V_i are applied
	2	all float	all active
BLs	1	all active	all active
	2	all active	all float -{output row connect to the ground}
WLBs	1	all float	all float
	2	all float-{output column connect to the ground}	all float

Corner case circuit

The corner case circuit is shown in Fig. D.1. In the Appendix C, we simplified the circuit even further and calculate the Thevenine equvalents observed from the last row. In Fig. D.2, we show the reconfiguration and simplification of circuit shown in Fig. D.1.

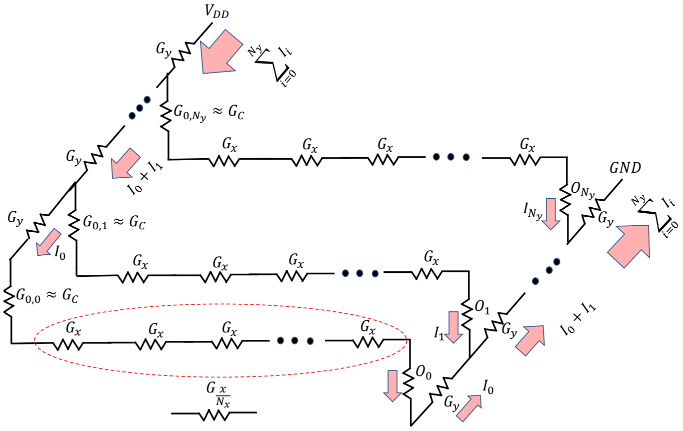


Figure D.1: The equivalent circuit model for the worst case

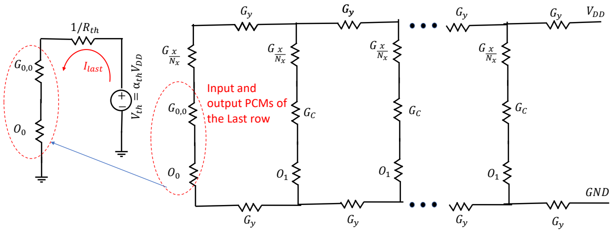


Figure D.2: Reconfiguration and simplification of equivalent circuit model.

Multi-bit operations

Thus far, we have discussed the implementations of operations with binary digits. We introduce two methods that enable us to implement operations with multi-bit digits. For brevity, we explain the principle using two-bit digits, where each digit consists an MSB (most significant bit) and an LSB (least significant bit). Let us assume that we want to perform TMVM of $G_{2bit}V$ where G_{2bit} is a $(N_x + 1) \times (N_y + 1)$ matrix with two-bit elements, meaning the element located in row i and column j of the matrix G_{2bit} is $G_{i,j}^{MSB} G_{i,j}^{LSB}$.

Fig. D.3 illustrates two ways to implement 2-bit operations on a 3D XPoint subarray. Fig. D.3(a) shows an area-efficient approach. For example, to calculate O_0, we need to calculate $(G_{0,0}^{LSB} + 2G_{0,0}^{MSB})V_0 + (G_{1,0}^{LSB} + 2G_{1,0}^{MSB})V_1 + ... + (G_{N_x,0}^{LSB} + 2G_{N_x,0}^{MSB})V_{N_x}$. To do so, we can apply V_0 to the WLT_0 (connected to the PCM storing $G_{0,0}^{LSB}$ bit), and we can apply $2V_0$ to WLT_1 (connected to the PCM storing $G_{0,0}^{MSB}$ bit). Therefore, the current flowing through the MSB cell is two times larger than that of the LSB cell. Another more area-intensive approach, which does not require multiple voltage levels, is shown in Fig. D.3(b) where we copy the MSB in pair of adjacent cells, and we apply the same voltage to their corresponding WLTs. The current through the MSB cell is weighted to be twice that of the current through the LSB cell.

We analyze the energy and area for the implementation a multi-bit TMVM using two schemes that we introduced in Section 5.4. We listed the results in Table D.5. As

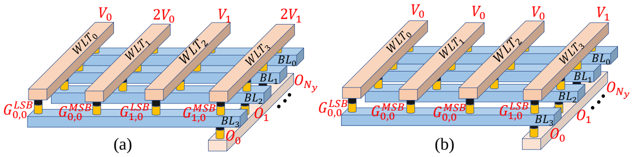


Figure D.3: Two implementations with multi-bit operations: (a) area-efficient implementation, (b) low-power implementation.

Table D.5: Evaluation of implementation energy and area for multi-bit TMVM using area efficient and low power schemes.

Parameters	Implementation Scheme	Number of Bits for each *G* element					
		1	2	3	4	5	6
TMVM Energy (pJ)	Area Efficient	2.0	5.0	13.1	▨	▨	▨
	Low Power	2.0	2.2	2.4	2.5	2.6	2.6
TMVM Area (μm^2)	Area Efficient	0.2	0.4	0.6	▨	▨	▨
	Low Power	0.2	0.6	1.3	2.8	5.7	11.6

we increase the number of bits for the $G_{i,j}$s, the allocated area for both implementations increases. However, while for area-efficient scheme, the area increases linearly, for the low-power scheme, the area increases exponentially. The implementation energy in the low-power scheme slightly increases with increasing the number of bits, while for the area-efficient schemes, energy increases rapidly. For the area-efficient scheme, we do not list the energy and area values beyond 3 bits, because it requires applying a large voltage level (>5V) within the subarray, making the implementation infeasible and unrealistic.