GPA: A General-Purpose In-Memory Computing Accelerator

Xiaoyu Zhang[1], Zerun Li[1], Rui Liu[1], Libo Shen[3], Boyu Long[1,2], Xueqi Li[1], Yinhe Han[1], Xiaoming Chen[1*]
[1]Institute of Computing Technology, Chinese Academy of Sciences
[2]School of Computer Science and Technology, University of Chinese Academy of Sciences
[3]The Chinese University of Hong Kong
[*]Corresponding author (e-mail: chenxiaoming@ict.ac.cn)

2026 IEEE International Symposium on Circuits and Systems (ISCAS) | 979-8-3315-7769-8/26/$31.00 ©2026 IEEE | DOI: 10.1109/ISCAS66217.2026.11562974

***Abstract*—In-memory computing (IMC) is considered as a promising technique to alleviate the memory wall bottleneck. Various specialized IMC-based accelerators have been proposed recently to accelerate applications in different fields. In this paper, we propose GPA, a general-purpose accelerator based on the IMC technique. GPA is a massively parallel architecture that works in single-instruction-multiple-data (SIMD) manner. It is composed of a three-level hierarchy of computation and storage units (PE-core-array), where the IMC arrays are elaborated to support various operations on both integers and floating-points. We propose four working phases to avoid the potential structural conflict problem in IMC accelerators. An instruction set is also developed for GPA as a programming interface. Compared with an NVIDIA V100 GPU, evaluation results show that GPA achieves 161.28×, 169.66×, 73.18×, and 579.16× speedups and 6.23×, 6.66×, 24.16×, and 412.37× energy savings in four different applications.**

***Index Terms*—In-memory computing, Processing-in-memory, FeFET**

I. Introduction

Processing-in-memory (PIM) is considered as a promising technique to alleviate the memory wall bottleneck. As a type of PIM techniques, the in-memory computing (IMC) technique performs in-situ computations directly within the memory array and has been used in many fields to accelerate different applications, such as convolutional neural network (CNN) inference and training (e.g. [1], [2]), scientific computing (e.g. [3], [4]) and graph processing (e.g. [5], [6]).

Taking advantages of the reduction of the data movement, IMC accelerators achieve better performance and higher energy efficiency in these applications. However, for most existing IMC architectures, since the hardware and architectural design are specialized to accelerate a single class of applications, it is difficult for them to accelerate different applications, which greatly lowers the generality of these accelerators. Previous researchers have also proposed some multi-functional IMC accelerators, such as Liquid Silicon [7], [8], Merging Everything [9], and Re-FeMAT [10]. However, these accelerators focused on accelerating a set of similar applications. Due to the lack of instruction sets, it is difficult for them to accelerate general-purpose tasks.

Constructing general-purpose IMC accelerators faces severe challenges in the array and architecture designs. The biggest challenge is how to build IMC arrays such that they can enable general-purpose computing. Existing IMC arrays are usually for analog matrix-vector multiplications (e.g., [11], [12]). Some other works implement logic operations in IMC arrays (e.g., [13]–[15]). Different operations in IMC arrays are based on different mechanisms and array designs. It is challenging for an IMC array to support various operations on various data types (e.g., integers and floating-points).

Another issue is the potential structural conflict when multiple IMC arrays work together. In each work cycle, an IMC array can only perform one operation, which means that it either works as a memory array (performing data read or write) or as an computing array (performing in-memory computations). A single IMC array needs to request data from other arrays and/or write the results to other arrays. At the same time, other arrays are executing their own operations on their own arrays and the external data access requests will interrupt their executions, leading to a structure conflict problem. This is a special problem of IMC. Without an apposite workflow, this problem will significantly reduce the performance.

In view of the poor generality of current IMC accelerators and the above-mentioned challenges, this paper proposes an IMC-based general-purpose accelerator, GPA, which is built using ferroelectric field-effect transistors (FeFETs). The contributions of this paper are summarized as follows.

- We propose a General-Purpose in-memory computing Accelerator named GPA. We further propose a three-level storage and computing structure and a four-stage workflow to enable the arrays of GPA to work efficiently.
- We design an IMC array that can support logic and arithmetic operations on different data types, which is the basic storage and computing component of GPA.
- Compared with an NVIDIA V100 GPU, GPA achieves significant speedups and energy savings in different applications. Specifically, GPA achieves 161.28×, 169.66×, 73.18× and 579.16× speedups and 6.23×, 6.66×, 24.16×, and 412.37× energy savings in four floating-point and integer applications.

II. GPA Architecture Overview

Fig. 1 shows the architecture overview of GPA. A three-level storage and computing hierarchy is used in GPA to support

This work was supported in part by National Natural Science Foundation of China under Grant 62488101, Grant 62495104, and Grant 62025404, and in part by Youth Innovation Promotion Association CAS.

Authorized licensed use limited to: University of Chinese Academy of SciencesCAS. Downloaded on July 25,2026 at 12:23:50 UTC from IEEE Xplore. Restrictions apply.

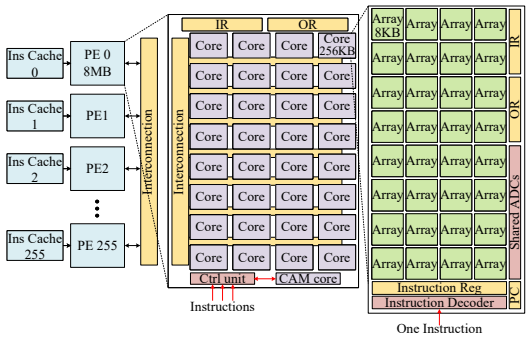

Fig. 1: **Architecture hierarchy of GPA.**

TABLE I: **GPA instruction set architecture.**

Instruction Type	Operations	Data Type
Boolean logic	AND	Integer
	OR	
	XOR	
	XNOR	
	NOT	
Arithmetic	Rem	Integer
	Add(i,f)	Integer and floating-point
	Sub(i,f)	
	Mul(i,f)	
	Div(i,f)	
	MAC(i,f)	
	Abs(i,f)	
	Cmpset(i,f)	
Data movement	MOV(i,f)	Integer and floating-point
	Read(i,f)	
	Write(i,f)	
Branch	C Branch	-
	UC Branch	

the SIMD working manner, which can be divided into PE-level, core-level and array-level from top to bottom. Different computing components are integrated in each level.

Array-level: Each IMC array is composed of a FeFET-based crossbar array (whose size is 256×256) and other peripheral circuits. In each cycle, each IMC array performs a specified operation according to the its instruction.

Core-level: Thirty-two IMC arrays are integrated in a core and other hardware resources in this core are shared among these arrays. Each core works in a SIMD fashion, which means that when an instruction is fed to a core, all arrays in that core will execute the same instruction, but on different data stored in different arrays. Different arrays within a core work in parallel. They share the same data input and output paths, and there are no connection paths between them.

PE-level: A PE is composed of thirty-two cores. GPA integrates 256 PEs and can store up to 2GB data. Input and output registers are integrated in each PE for inter-PE data transmission. Different cores execute different instructions and work in parallel. To handle data transfer conflict problem, each PE integrates a control unit and a content-addressable memory (CAM) core. A set of instructions will be compared in the CAM core to detect conflicts.

Table I shows the instructions supported by GPA. GPA supports a total of 28 instructions which can be divided into four types, including Boolean logic instruction, arithmetic instruction, data movement instruction and branch instruction.

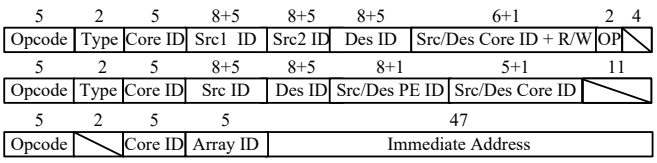

Fig. 2: **Instruction formats of GPA. From top to bottom are the Boolean logic and arithmetic instruction, data movement instruction, and branch instruction.**

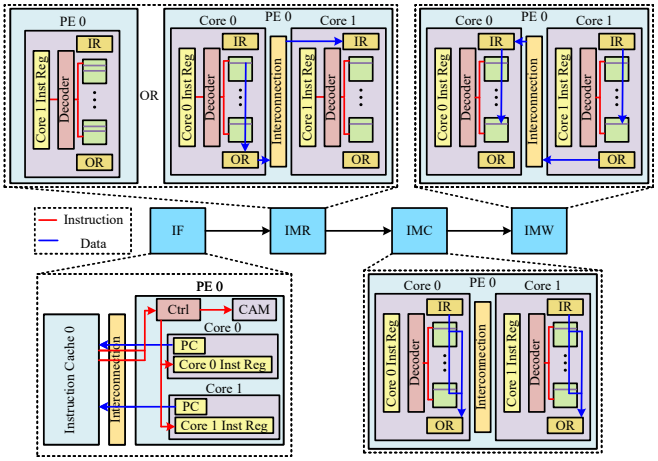

Fig. 3: **Working phases of GPA.**

Fig. 2 shows the format of instructions. Boolean logic instructions and arithmetic instructions use the same instruction format, which contains the addresses of three operands in the same core, including two source operands and a destination location. Each operand address includes an 8-bit row ID and a 5-bit position ID on that row. To transfer data between cores, the next 7 bits specify the destination core and whether to perform a read or write operation on that core. Data transfer between PEs can be completed using MOV instruction. The read and write instructions are respectively responsible for reading (writing) data at (to) the specified location to (from) the PE's output (input) registers.

III. Four Working Phases of GPA

Each core in each PE of GPA works in a four-stage approach and the four phases include the instruction fetch phase (IF), in-memory reading phase (IMR), in-memory computation phase (IMC) and in-memory writing phase (IMW). Different cores run each phase synchronously. This working mechanism is designed to resolve the potential structural conflict issues and make sure that each core not only has time to process its own computing tasks but also has time to respond to requests from other cores. Fig. 3 shows the working phases of the GPA.

IF phase: During this phase, different PEs read instructions from their instruction caches according to the PC of each core in this PE. A PE's instructions will be transmitted to the control unit and then to the CAM core for comparison and detection of transmission conflicts. When more than two instructions have the same write address or source data address, a few more cycles need to be reserved in the corresponding work phase, that is, IMR phase for same source address situation

Authorized licensed use limited to: University of Chinese Academy of SciencesCAS. Downloaded on July 25,2026 at 12:23:50 UTC from IEEE Xplore. Restrictions apply.

and IMW phase for same write address situation, to ensure that the data transfer can be completed. When the comparison is completed, a computation instruction and read and write requests from other instructions will be distributed to each core. A core receives its own complete instruction as well as a read address and a write address from other instructions.

IMR phase: During this phase, each core of a PE read the data in each array based on the read addresses which have been stored in the instruction register of this core. The data from the same location in different arrays will be stored in the output registers of each core and then transmitted to the input registers of the destination cores via the inter-core interconnection within the same PE. As shown in Fig. 3, data is read from core 0 and transferred to core 1 in the same PE. Core 1 can also perform read operations and transfer data to other cores at the same time. Considering that we perform data transmission conflict detection in the IF phase, there will only be one-to-one data transmission between cores. Additional data transfer requests will be completed using several cycles reserved in the IF phase. When the two operand addresses specified by one instruction to participate in the computation are in the same core (the example on the left in Fig. 3), inter-core data transmission is no longer needed.

IMC phase: Different cores perform their IMC operations in the IMC phase. One operand involved in the computation will be passed to each array from the input register of each core, while the other operand has been initially stored inside each array. Both operands may initially be stored on the same array, in which case there is no need to get the data from input register. All arrays within a core share the same instruction and decoder, which guarantees the synchronization of these arrays. The computation results of each core will be saved in their output registers. The IMC phase ends when different instructions running on all cores in a PE have been executed.

IMW phase: In the last phase, the computation results that need to be written back will be transferred from the output registers of the computing core to the input registers of the writing core. Depending on whether the data will be used by the subsequent instructions, the data will be temporarily stored in the registers or written back to the array. Specifically, due to the limited capacity of the input registers, the intermediate result is only allowed to be stored in the registers if it will be used as an operand in the next instruction, otherwise it should be written to the array. Using registers to temporarily store intermediate results can reduce the delay and energy consumption caused by write back.

IV. Array Design of GPA

Fig. 4a shows a GPA array. The different operands which need to be computed are aligned and stored on different rows. Since binary cells are used, multiple adjacent cells on a row store different bits of the same operand. The ADCs for MAC operations are integrated at the core level and shared by multiple arrays. The two types of SAs are precharge-based SAs (PSAs) and current-sensing-based SAs (CSAs) [10].

Since the operands involved in computation may come from the array itself or other arrays, the IMC array should support

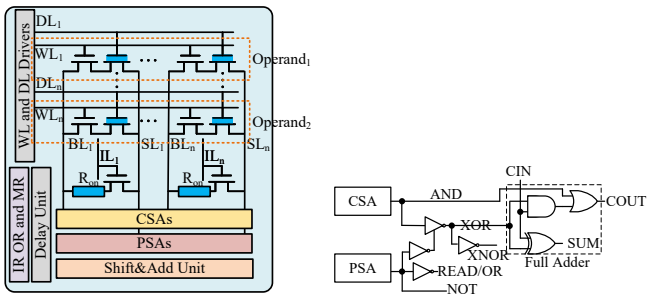

Fig. 4: **Array design of GPA. (a) Array. (b)Peripheral circuits for logic operations.**

TABLE II: **Control signals of basic operations of GPA.**

	WL	BL	DL	SL	IL
Write	$V_{DD}{}^a/0^b$	Data ($\pm V_{DD}$)	0	0	0
Read/OR	$V_{DD}{}^a/0^b$	0	$V_{DD}{}^a/0^b$	Sensing(P)	0
AND	$V_{DD}{}^a/0^b$	Sensing(C)	$V_{DD}{}^a/0^b$	V_R	0
ORI	$V_{DD}{}^a/0^b$	0	$V_{DD}{}^a/0^b$	Sensing(P)	Data (V_{DD}/0)
ANDI	$V_{DD}{}^a/0^b$	Sensing(C)	$V_{DD}{}^a/0^b$	V_R	Data (V_{DD}/0
Search	V_{DD}	0	Data (V_{DD}/0)	Sensing(P)	0
MAC	$V_{DD}{}^a/0^b$	Sensing(A)	Data (V_{DD}/0)^a/0^b	V_R	0

^{a} For selected row; ^{b} For unselected rows.
V_{DD} = 1V; V_R = 0.1V; P: PSA; C: CSA; A: ADC

computations between two stored data and computations between a stored data and an immediate data. A row of cells which are composed of NMOSs and resistors are integrated in each array. When an operand is an immediate data, the NMOS cells will replace the function of FeFETs and serve as the immediate input ports. The resistors in these cells have the same resistance as the on state FeFETs. The design philosophy of GPA supporting multiple operations is to combine basic operations to implement complex ones. The basic functions supported by the array are shown in Table II. Other functions can be implemented using these basic functions.

The Boolean logic and addition operations can be obtained through the peripheral circuits shown in Fig. 4b. To perform subtraction, the NOT result of one operand will be output and added with 1. This result will be added to the other operand to get the result of the subtraction. Performing multiplication on integer data is similar to performing a single-row MAC. The division and remainder operations are implemented through multi-cycle subtraction and comparison with zero.

When performing floating-point addition or subtraction, subtraction is first performed on the exponents of the two operands to find the larger one. Then the mantissa of the smaller operand is read and shifted. The shifted mantissa is then input into the array to perform addition or subtraction with the mantissa of the other operand. The output result will be cut according to the maximum number of bits of the mantissa. The number of extra bits will be input to the array and added with the exponent of the larger operand to get the exponent of the final result. When performing floating-point multiplication, the exponents of the two operands are added. The mantissa of an operand is then read and multiplied by the other mantissa using the same approach as the integer

Authorized licensed use limited to: University of Chinese Academy of SciencesCAS. Downloaded on July 25,2026 at 12:23:50 UTC from IEEE Xplore. Restrictions apply.

TABLE III: **GPA Specification.**

Number of PEs	256	Core per PE	32
Array per core	32	ADC per core	4
ADC Resolution	9 bits	ADC Power	10.8mW
ADC Area	0.038mm^2	Array Size	256×256
Frequency	1GHz	Storage Capacity	2GB

multiplication. Then, the mantissa will be cut and the exponent will be adjusted. The calculation process of floating-point division is similar to that of floating-point multiplication, with the difference that subtraction is performed on the exponents.

When performing MAC operations on floating-points, the mantissas of the operands involved in the calculation need to be aligned according to the exponents. Serial subtraction is performed on each exponent to find the largest exponent and the difference between each exponent and the largest exponent. The differences are then transmitted to the delay unit, which generates different delay values based on the differences. According to the delay value, data will be input through DLs in the corresponding cycle. Then the subsequent MAC process is the same as integer MAC.

V. Evaluation

A. Evaluation Setup

We implement and evaluate the analog and digital components in GPA using HSPICE and Synopsys Design Compiler, respectively. The Preisach FeFET model [16], [17] and 45nm predictive technology model [18] are used for FeFETs and MOSFETs, respectively. The Preisach model has been calibrated based on fabricated FeFET devices [19]. The ADC parameters from [20] are used in the evaluation. Table. III shows the specification of GPA. We compare the performance and energy consumption of GPA with an NVIDIA Tesla V100 GPU in various applications, including general matrix-vector multiplication (GEMV), general matrix-matrix multiplication (GEMM), Jacobi iteration algorithm, and MD5 algorithm. Since the operations of GPA are implemented based on logical operations. We also compare GPA with three logic-in-memory (LIM) based IMC accelerators [8], [21], [22]. The operations supported by these accelerators are a subset of GPA's. For fair comparisons, the same data type and the same data amount are used in the comparisons.

B. Comparison with GPU

Fig. 5 shows the speedups and energy savings of GPA compared with GPU in different applications. On average, GPA achieves 161.28×, 169.66×, 73.18×, and 579.16× speedups and 6.23×, 6.66×, 24.16×, and 412.37× energy savings in GEMV (floating-point), jacobian iterative solver (floating-point), GEMM (floating-point), and MD5 algorithm (integer), respectively, compared with GPU. GPA achieves significant improvements because it can not only take advantage of IMC technique, but also work efficiently in a SIMD manner.

C. Comparison with Other IMC Accelerators

Fig. 6 shows the comparison results between GPA and other IMC-based accelerators. These accelerators include Plim (MAJ-logic-based) [22], FloatPIM (MAGIC-logic-based) [21]

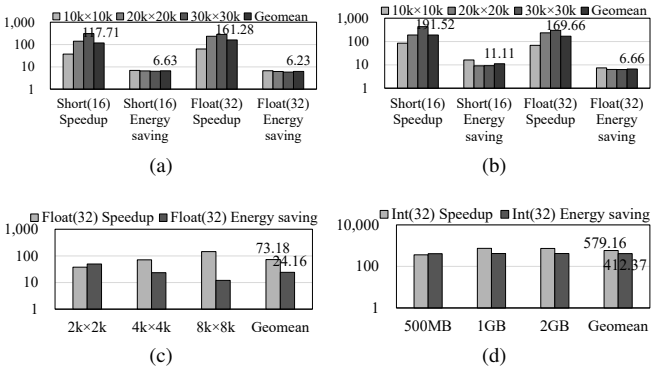

Fig. 5: **Speedups and energy savings compared with GPU. (a) GEMV. (b) Jacobian solver. (c) GEMM. (d) MD5.**

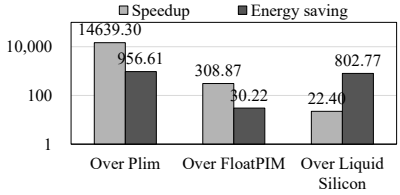

Fig. 6: **Speedups and energy savings of GPA compared with other IMC accelerators.**

and Liquid Silicon (FPGA-like) [8]. GPA and these accelerators are compared on the operations supported by them. The speedup and energy savings across different instructions are averaged and shown in Fig. 6. On average, GPA achieves 14639.30×, 308.87×, and 22.40× speedups and 956.61×, 30.22×, and 802.77× energy savings over Plim, FloatPIM, and Liquid Silicon, respectively. Serial single-bit logic is used in Plim, resulting in the worst performance. Although FloatPIM has achieved support for floating-point operations, its computation process relies on write-back. In contrast, GPA separates computation from write-back and can handle immediate values, thereby reducing the number of intermediate result write-backs and enhancing performance. Liquid Silicon's storage and computation modes can't coexist. It avoids structural conflicts but needs double resources, degrading performance. With structural conflicts resolved and SIMD employed, GPA outperforms other IMC accelerators in floating-point and integer operations.

VI. Conclusion

Many existing IMC studies are focused on designing domain-specific accelerators. This work aims to construct an IMC-based general-purpose accelerator. By designing general-purpose IMC arrays and a three-level hierarchical parallel architecture, together with an instruction set architecture, GPA achieves 161.28×, 169.66×, 73.18×, and 579.16× speedups and 6.23×, 6.66×, 24.16×, and 412.37× energy savings compared with GPU in four applications. As a general-purpose accelerator, by applying SIMD and resolving potential structural conflict issues, GPA also demonstrates higher performance and energy efficiency compared to other LIM-based and FPGA-like IMC accelerators.

Authorized licensed use limited to: University of Chinese Academy of SciencesCAS. Downloaded on July 25,2026 at 12:23:50 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. Pipelayer: A pipelined reram-based accelerator for deep learning. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 541–552, 2017.

[2] Peng Yao, Huaqiang Wu, Bin Gao, Jianshi Tang, Qingtian Zhang, Wenqiang Zhang, J Joshua Yang, and He Qian. Fully hardware-implemented memristor convolutional neural network. *Nature*, 577(7792):641–646, 2020.

[3] Ben Feinberg, Uday Kumar Reddy Vengalam, Nathan Whitehair, Shibo Wang, and Engin Ipek. Enabling scientific computing on memristive accelerators. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 367–382, 2018.

[4] Xiaoyu Zhang, Zerun Li, Rui Liu, Xiaoming Chen, and Yinhe Han. Fspa: An fefet-based sparse matrix-dense vector multiplication accelerator. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2023.

[5] Nagadastagiri Challapalle, Sahithi Rampalli, Linghao Song, Nandhini Chandramoorthy, Karthik Swaminathan, John Sampson, Yiran Chen, and Vijaykrishnan Narayanan. Gaas-x: Graph analytics accelerator supporting sparse data representation using crossbar architectures. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 433–445, 2020.

[6] Zerun Li, Xiaoming Chen, and Yinhe Han. Graphring: an hmc-ring based graph processing framework with optimized data movement. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 1063–1068, 2022.

[7] Yue Zha and Jing Li. Liquid silicon: A data-centric reconfigurable architecture enabled by rram technology. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, page 51–60, 2018.

[8] Yue Zha, Etienne Nowak, and Jing Li. Liquid silicon: A nonvolatile fully programmable processing-in-memory processor with monolithically integrated reram. *IEEE Journal of Solid-State Circuits*, 55(4):908–919, 2020.

[9] Xiaoming Chen, Longxiang Yin, Bosheng Liu, and Yinhe Han. Merging everything (me): A unified fpga architecture based on logic-in-memory techniques. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–2, 2019.

[10] Xiaoyu Zhang, Rui Liu, Tao Song, Yuxin Yang, Yinhe Han, and Xiaoming Chen. Re-femat: A reconfigurable multifunctional fefet-based memory architecture. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(11):5071–5084, 2022.

[11] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 27–39, 2016.

[12] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 14–26, 2016.

[13] Nishil Talati, Saransh Gupta, Pravin Mane, and Shahar Kvatinsky. Logic design within memristive memories using memristor-aided logic (magic). *IEEE Transactions on Nanotechnology*, 15(4):635–650, 2016.

[14] Saeideh Shirinzadeh, Mathias Soeken, Pierre-Emmanuel Gaillardon, and Rolf Drechsler. Fast logic synthesis for rram-based in-memory computing using majority-inverter graphs. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 948–953, 2016.

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

[16] Kai Ni, Matthew Jerry, Jeffrey A. Smith, and Suman Datta. A circuit compatible accurate compact model for ferroelectric-fets. In *2018 IEEE Symposium on VLSI Technology*, pages 131–132, 2018.

[17] Kai Ni, Pankaj Sharma, Jianchi Zhang, Matthew Jerry, Jeffery A. Smith, Kandabara Tapily, Robert Clark, Souvik Mahapatra, and Suman Datta. Critical role of interlayer in hf0.5zr0.5o2 ferroelectric fet nonvolatile memory performance. *IEEE Transactions on Electron Devices*, 65(6):2461–2469, 2018.

[18] Wei Zhao and Yu Cao. New generation of predictive technology model for sub-45 nm early design exploration. *IEEE Transactions on electron Devices*, 53(11):2816–2823, 2006.

[19] P. Sharma, K. Tapily, A. K. Saha, J. Zhang, A. Shaughnessy, A. Aziz, G. L. Snider, S. Gupta, R. D. Clark, and S. Datta. Impact of total and partial dipole switching on the switching slope of gate-last negative capacitance fets with ferroelectric hafnium zirconium oxide gate stack. In *2017 Symposium on VLSI Technology*, pages T154–T155, 2017.

[20] Hyeok-Ki Hong, Hyun-Wook Kang, Barosaim Sung, Choong-Hoon Lee, Michael Choi, Ho-Jin Park, and Seung-Tak Ryu. An 8.6 enob 900ms/s time-interleaved 2b/cycle sar adc with a 1b/cycle reconfiguration for resolution enhancement. In *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pages 470–471, 2013.

[21] Mohsen Imani, Saransh Gupta, Yeseong Kim, and Tajana Rosing. Floatpim: In-memory acceleration of deep neural network training with high precision. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 802–815, 2019.

[22] Pierre-Emmanuel Gaillardon, Luca Amarú, Anne Siemon, Eike Linn, Rainer Waser, Anupam Chattopadhyay, and Giovanni De Micheli. The programmable logic-in-memory (plim) computer. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 427–432, 2016.

Authorized licensed use limited to: University of Chinese Academy of SciencesCAS. Downloaded on July 25,2026 at 12:23:50 UTC from IEEE Xplore. Restrictions apply.