Racetrack memory based logic design for in-memory computing

Luo, Tao

2018

Luo, T. (2018). Racetrack memory based logic design for in-memory computing. Doctoral thesis, Nanyang Technological University, Singapore.

<http://hdl.handle.net/10356/73359>

<https://doi.org/10.32657/10356/73359>

RACETRACK MEMORY BASED LOGIC DESIGN FOR IN-MEMORY COMPUTING

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy

LUO TAO

August 2017

Abstract

In-memory computing has been demonstrated to be an efficient computing infrastructure in the big data era for many applications such as graph processing and encryption. The area and power overhead of CMOS technology based memory design is growing rapidly because of the increasing data capacity and leakage power along with the shrinking technology node. Thus, a newly introduced emerging memory technology, racetrack memory, is proposed to increase the data capacity and power efficiency of modern memory systems. As the design requirements of the conventional logic are different from that of the emerging memory based logic for in-memory computing, the conventional well-developed CMOS technology based logic designs are less relevant to the emerging memory based in-memory computing. Therefore, novel logic designs for racetrack memory are required.

Traditional logic design with separate chips is focusing on high speed, which causes large area and power consumption. Implementing efficient logic design for in-memory computing is challenging due to the demanding requirement for area and power. Firstly, as the computing logic for in-memory computing is built in memory, the available area budget is limited, otherwise the data density of the memory system would be affected. Secondly, due to the thermal constraint of the memory chip, the available energy budget for computing logic design is limited. Large energy consumption may cause malfunction and even permanent damage to the memory chip because of high temperature. Finally, the adoption of emerging memory technologies makes the logic design more challenging due to their unique characteristics such as the sequential access mechanism of racetrack memory.

This thesis addresses the above challenges in racetrack memory based in-memory logic design as follows. First, for general computing operations, we first propose racetrack memory based half and full adders. The proposed magnetic full adder is implemented with pre-charged sense amplifiers (PCSA) and magnetic tunnel junctions (MTJ). By reusing parts of the logic design, the magnetic full adder significantly improves the area and energy efficiency compared with CMOS-based full adder and the state-of-the-art magnetic full adder.

Second, based on the proposed magnetic full adder, we propose a pipelined Booth multiplier by exploring the inherent sequential access mechanism of racetrack memory, which achieves high area and energy efficiency. In order to increase the throughput of proposed Booth multiplier, we further parallelize the generation and addition of the partial products of the proposed Booth multiplier. Unlike the area- and energy-consuming adder array architecture in conventional CMOS technology based designs, the proposed multiplier utilizes a weight-based parallel architecture. In order to ensure the high energy efficiency, we propose an optimization that transforms the energy-demanding write operations to shift operations. With this optimization, the weight-based parallel multiplier achieves high throughput while maintaining high area and energy efficiency.

Third, for specific applications, we propose an efficient racetrack memory based design to accelerate modular multiplication. Modular multiplication is widely used in various applications such as cryptography, number theory, group theory, ring theory, knot theory, abstract algebra, computer algebra, computer science, chemistry and the visual and musical arts. In order to implement modular multiplication efficiently, a novel two-stage scalable modular multiplication algorithm is proposed to significantly reduce the delay. An efficient architecture based on racetrack memory is further developed to reduce the number of required adders. Racetrack memory based application specific design for modular multiplication shows significant improvement compared with the state-of-the-art CMOS technology based implementation in area, energy, and performance.

Overall, this thesis has made contributions to address the challenges in racetrack memory based in-memory logic design, and we demonstrate significant im-

provements in terms of area overhead and energy consumption in comparison with the state-of-the-art CMOS technology based logic design.

Acknowledgments

Firstly, I would like to give my great gratitude to my supervisors, Dr. Zhang Wei at Hong Kong University of Science and Technology, Dr. He Bingsheng at National University of Singapore, and Dr. Douglas L. Maskell at Nanyang Technological University. Their guidance and support truly trained and helped me in my research work. I learned a lot from both life and study in the past four years. Without their great effort, I cannot finish this dissertation. Secondly, I want to thank all the people who collaborated with me in my Ph.D. study, including Dr. Cui Yingnan, Dr. Liang Hao, Dr. Zhou Bin, Dr. Li Hai, Dr. Chen Yi-Chung, Dr. Thambipillai Srikanthan, Dr. Jason Teo Kian Jin, Dr. Vivek Chaturvedi and Dr. Liu Cheng. Their enlightening recommendations and discussion have been a great boon to my research. Thirdly, I would like to give my thanks to my friends and colleagues, Dr. Cui Jin, Dr. Chen Yupeng, Dr. Yang Liwei, Dr. Jiang Lianlian, Dr. Gu Xiaozhe, Dr. Li Xiangwei, Dr. Chen Hao, Dr. Mao Fubing, Dr. Gao Yiyi, Dr. Lin zhe, Dr. Feng Liang, Dr. He Wenjian, Dr. Zhao Jieru, Dr. Zhu Zuoming, Dr. Mu Jiandong, Dr. Niu Zhaojie, Dr. Johns Paul and many others, who helped and cheered me in the past four years. Finally, I specially thank my parents for their unconditional love and support, and my love, Chen Yanyi, who always believes in me and gives me continuous support.

Contents

Abstract	i
Acknowledgments	v
List of Figures	xi
List of Tables	xiii
Abbreviations	xv
1 Introduction	1
1.1 Emerging Memory Technologies	3
1.2 Computing Logic Design	6
1.3 Motivations	7
1.4 Contributions	8
1.5 Organization	10
2 Background and Literature Review	11
2.1 Background	11
2.1.1 Racetrack Memory	11
2.1.2 Adder and Booth Multiplier	13
2.1.3 Montgomery Multiplication	14
2.2 Literature Review	15
2.2.1 Applications of Racetrack Memory	15
2.2.2 Logic Design for In-memory Computing	20
2.2.3 Hardware Implementations of Modular Multiplication	23
2.3 Summary	27

3	Racetrack Memory Based Magnetic Half Adder and Full adder	31
3.1	Introduction	32
3.2	Racetrack Memory Based Magnetic Half Adder	33
3.3	Racetrack Memory Based Magnetic Full Adder	36
3.4	Experiment Results	39
3.5	Summary	41
4	Racetrack Memory Based Booth Multiplier	43
4.1	Introduction	44
4.2	Pipelined Booth Multiplier Based on Racetrack Memory	45
4.2.1	Partial Products Generation	47
4.2.2	Addition of Partial Productions	49
4.3	Booth Multiplier with Weight-based Parallel Architecture	51
4.3.1	Weight-based Parallel Architecture	51
4.3.2	Write Optimization	55
4.4	Experimental Results	55
4.4.1	Experiment Setup	56
4.4.2	Adders with Write Operation	56
4.4.3	Multipliers	57
4.5	Summary	63
5	Two-stage Modular Multiplier Based on Racetrack Memory	65
5.1	Introduction	66
5.2	Word-based Montgomery Multiplication	67
5.3	Proposed Modular Multiplication Algorithm	69
5.3.1	Independence of MSB Operation	70
5.3.2	Two-stage Algorithm	72
5.4	Architecture Design and Optimization	74
5.4.1	Processing Element Design and Optimization	74
5.4.2	Architecture of the Proposed Modular Multiplication	78
5.5	Experimental Results	79
5.5.1	Experiment Setup	79

5.5.2	Evaluation of Basic Logic	80
5.5.3	Evaluation of Modular Multiplication	81
5.6	Summary	84
6	Conclusion and Future Work	85
6.1	Conclusion	85
6.2	Future Work	87
6.2.1	Racetrack Memory Based Reconfigurable Modular Multiplier for Both RSA and ECC	87
6.2.2	In-memory CGRA Design for Homomorphic Encryption with Montgomery Multiplication	87
6.2.3	In-memory Accelerator for Encrypted Database	88
Author's Publication		
References		93

List of Figures

1.1	Spin-transfer-torque magnetic random access memory	4
1.2	Racetrack memory	5
2.1	Basic structure of vertical magnetic tunnel junction and the race-track memory	12
3.1	The schematic of proposed magnetic half adder	35
3.2	The schematic of proposed magnetic full adder	37
4.1	A 8-bit x 8-bit Booth multiplication with radix-4	46
4.2	Data organization for generation of partial products	49
4.3	The pipelined addition based on racetrack memory	50
4.4	Data organization for weight-based generation of partial products	53
4.5	Weight-based parallel addition of the partial products	54
4.6	Transformation of write operations to shift operations	56
4.7	Throughput of Booth multipliers with different radix base, architectures, and input length	58
4.8	Area overhead of Booth multipliers with different radix base, architectures, and input length	60
4.9	Energy of Booth multipliers with different architecture and configurations	62
5.1	The data dependency graph for Alg. 1	69
5.2	The data dependency graph for Alg. 2	73
5.3	An example of how the optimized adder operates	77
5.4	The structure of basic logic units and proposed adder	77

5.5	Architecture of the proposed modular multiplication	78
5.6	Performance comparison of different schemes	81
5.7	Energy per bit with different word length	83
5.8	Throughput with different word length	84

List of Tables

3.1	Truth table of the carry function for the half adder	35
3.2	Truth table of the sum function for the half adder	36
3.3	Truth table of the carry function of the full adder	37
3.4	Truth table of the sum function of the full adder	38
3.5	Main parameters in the RM model	39
3.6	Comparison of the proposed half adder and three full adders . .	40
4.1	Radix-4 based encoding and decoding regarding to multiplicand	47
4.2	Comparison of the number of required write operations for dif- ferent adders	57
5.1	Main parameters in the RM model	80
5.2	Results of the basic logic	80

Abbreviations

Abbreviations	Meaning
FA	Full Adder
FM	Ferromagnetic
HA	Half Adder
LSB	Least Significant Bit
MFA	Magnetic Full Adder
MHA	Magnetic Half Adder
MSB	Most Significant Bit
MTJ	Magnetic Tunnel Junction
PCM	Phase Change Memory
PCSA	Pre-Charge Sensing Amplifier
PMA	Perpendicular Magnetic Anisotropic
RM	Racetrack Memory
STT-MRAM	Spin-Transfer Torque Magnetic Random-Access Memory

Chapter 1

Introduction

According to Moore's law [1], hardware becomes cheaper and higher-performing along with the technology development. The continuous effort on improving cost and capacity of memory opens up the possibility of in-memory computing [2, 3, 4]. Nowadays, the explosion of data and the ever-growing requirement for fast data analysis in the big data era make in-memory computing an efficient computing infrastructure for many applications such as graph processing and encryption [5]. In-memory computing bridges the gap between memory and processor, alleviates the I/O pressure and reduces the overhead of data movement. However, it poses challenges on area, energy, and performance because of the shrinking of feature sizes and integration of computing logic. For memory cells in in-memory computing system, emerging memory technologies such as racetrack memory are proposed to improve the data density and power efficiency. However, developing computing logic that efficiently handle the computing tasks still remains an open question.

The logic design for emerging memory based in-memory computing has become a crucial issue because of the rapid progress of semiconductor technologies and adoption of emerging memory technologies. The area and power overhead of CMOS technology based memory design is growing in recent years because

of the increasing data capacity and leakage power along with the shrinking technology node. In addition, DRAM cannot scale easily below $40nm$ according to international technology roadmap for semiconductors (ITRS) [6]. In order to further increase the memory capacity and avoid the thermal emergencies due to the end of Dennard scaling, emerging memory technologies such as racetrack memory are adopted. However, with the application of in-memory computing, new concerns are raised. Firstly, the computing logic occupies large area overhead, which decreases the data density of the memory. In addition, large area overhead of the computing logic incurs the increase of fabrication cost. Secondly, the energy efficiency is a critical concern in the design of in-memory computing logic. With the ever-decreasing feature size, the static power of CMOS-based integrated circuits has increased dramatically, which results in the end of Dennard scaling. For example, in integrated circuits fabricated with $45nm$ or smaller processes, static power contributes to over 50% of the total power. High power density causes high temperature in memory chips which severely affects the reliability and performance of the memory [7], which result in the so-called power-wall problem [8]. Finally, the emerging memory technologies have unique features. For instance, racetrack memory has sequential access mechanism. Computing logic design without considering it would result in overhead in terms of performance, area, and power. In order to address such problems, logic designs for emerging memory based in-memory computing are required.

In the remainder of this chapter, we first introduce emerging memory technologies especially racetrack memory. Second, we briefly summarize the logic design with traditional CMOS technology and magnetic technology. Third, we discuss the motivation of this thesis. Fourth, we list the contributions we made in the logic design for racetrack memory based in-memory computing that are

included in this thesis. Finally, we demonstrate the organization of the whole thesis.

1.1 Emerging Memory Technologies

Understanding the adoption of emerging memory technologies is critical to design efficient emerging memory based in-memory logic. With the shrinking of technology node, the traditional CMOS technology based memory suffers from the scaling and power-wall problem [6]. Researchers shift their efforts and interests to emerging, resistive memory technologies to realize memory system with lower cost, higher capacity and lower energy. In addition, non-volatility of the resistive memory also makes it an ideal candidate for future memory systems. Resistive memory technologies store digital information by the high/low resistance of its memory cell. According to the mechanism, resistive memory technologies are categorized as phase change memory (PCM), ReRAM (memristor), spin-transfer-torque magnetic random access memory (STT-RAM or STT-MRAM), *etc.*

Resistance of PCM is determined by the phase of phase change material [9]. The phase change material has two states which are amorphous (high resistance) and crystalline (low resistance), respectively. Performance of PCM is determined by the set process, *i.e.*, the transition from amorphous to crystalline phase, while its power efficiency is limited by the reset process *i.e.*, the transition from crystalline to amorphous phase. Memory cell design and phase change materials determine the performance of PCM. Careful designs should be applied to simplify processing, optimize power efficiency, and reduce reset current. Overall, phase change materials demonstrate desirable scaling behaviors, but suffer from high cost and high crystallization temperature. Resistance of ReRAM (memristor) is

determined by atom distance [10]. ReRAM is promising due to its large on/off ratio, fast speed, and long endurance. Recently, a 16 GB Conductive-Bridge RAM (CBRAM), a type of ReRAM, test chip has been reported [11]. However, ReRAM still suffers from reliability, variability, and failure mechanisms.

For STT-MRAM, the resistance is determined by magnetic polarity [12]. Injection current changes the magnetic polarity to switch the state between high and low resistance. The STT-MRAM cell and its peripheral circuits are shown in figure 1.1. With STT-MRAM, a memory cell is mainly comprised of a magnetic

Figure 1.1: Spin-transfer-torque magnetic random access memory

tunnel junction (MTJ).

Like STT-MRAM, racetrack memory is a spin-based memory but with much higher data density [13]. With STT-MRAM, a memory cell is mainly comprised of a magnetic tunnel junction (MTJ). Instead of storing data in MTJs, racetrack memory stores data in a magnetic nanowire and only uses MTJs to write and read data. The racetrack memories and the peripheral circuits [13] are shown in figure 1.2. A single nanowire can contain 256 bits demonstrated by the racetrack memory manufactured so far [14], which results in high data density. Racetrack memories with vertical track and horizontal track are shown in figure 1.2.A and

Figure 1.2: Racetrack memory

figure 1.2.B, respectively. With reading and writing heads shown in figure 1.2.C and figure 1.2.D, the vertical track structure enables the racetrack memory to further utilize the chip space in 3D manner. Figure 1.2.E shows the racetrack storage array with vertical tracks, which can be applied as an external storage with ultra-high data density. Using as shift registers, its intrinsic shift property can improve the area and energy efficiency considerably. In addition, racetrack memory scales well below $10nm$, and leverages existing CMOS manufacturing techniques and

processes, which makes it an ideal memory technology for memory intensive applications.

1.2 Computing Logic Design

Computing logic can be categorized as general computing logic and application specific computing logic. The fundamental logic circuits for general computing mainly contains the half adder, the full adder, the multiplier, *etc.* The general computing logic usually serves as the arithmetic circuits in general computing unit such as central processing unit (CPU). The application specific computing logic is dedicated according to different applications. For traditional CMOS technology based design, the general computing logic is well-developed. Both adders and multipliers are thoroughly explored [15, 16, 17, 18]. Traditional CMOS technology based ASIC for specific applications is also well-studied. However, the traditional logic design for CMOS only technology is not applicable to in-memory computing scenario. This is because traditional CMOS technology based designs have large area overhead, which does not fit into in-memory style.

The rapid progress of semiconductor technology on deep sub-micro and nanometer scale announces the end of Dennard Scaling. The power consumption of the CMOS technology based design becomes more and more important. In order to alleviate the power-wall problem, magnetic device comes in to the view of researchers. A number of magnetic adders are developed and optimized for energy and area efficiency [19, 20, 21]. Due to the non-volatility of the magnetic device, the magnetic adders have non-volatile property too. The magnetic adders can be further divided into two categories according to the non-volatility. Partially non-volatile adders use the MTJ as one of their operand, which results in

the need to charge volatile data from CMOS registers [21, 20, 22, 23]. Fully non-volatile adders store all the operand in the non-volatile states, which leads to safe turn-off and power-on of the logic circuits [24, 25]. For the magnetic multiplier, to the best of our knowledge, there is no previous work. Magnetic device is also adopted to implement application specific design for low power consumption and area overhead.

1.3 Motivations

With wide adoption of emerging memory technologies and in-memory computing infrastructure, new challenges have been posed to in-memory logic design. Firstly, in order to ensure the large capacity and high data density of memory, the area budget of in-memory logic is limited. For instance, in Micron’s Hybrid Memory Cube (HMC) system [2], for one memory stack, only about $50mm^2$ of logic layer can be used for compute units. Due to the limited area budget, only simple logic operations such as “AND” and “OR” can be realized in memory for current in-memory computing systems. Therefore, computing logic, specifically, arithmetic logic with high area efficiency is required. In this thesis, we propose in-memory arithmetic designs with ultra-high area efficiency to address this problem. In addition, the mismatch between the low compute throughput and the available high bandwidth has negative influence on the overall system performance. To address this problem, in-memory computing logic with high throughput is required. In this thesis, we propose in-memory arithmetic designs with high throughput per area to bridge this gap. Secondly, for in-memory computing, the large energy consumption incurs thermal issues, which could cause malfunctioning of the memory chip. Thus, the computing logic design for in-memory

computing has rigorous energy constraint to avoid causing thermal issues. Therefore, energy efficiency plays an important role for in-memory computing logic design. However, conventional logic design cannot fit into the in-memory computing scenario because of the large energy consumption. In this thesis, we propose highly energy efficient in-memory computing logic to alleviate the stringent energy constraint. Finally, the application of emerging memory technologies requires customized design to take advantage of the unique features while avoiding its adverse impact. For instance, racetrack memory has magnetic nanowires as its storage media, which achieves ultra-high data density but leads to the sequential access mechanism. Therefore, dedicated designs should be applied. In this thesis, we propose in-memory computing logic that takes advantages of the memory features to realize high efficiency.

1.4 Contributions

In this thesis, we have three contributions in logic design for racetrack memory based in-memory computing.

- Firstly, we propose **racetrack memory based half and full adders** as the fundamental in-memory arithmetic circuit cells, which significantly improve the area overhead and energy consumption compared with CMOS technology only adder designs and the state-of-the-art magnetic adders. In this work, we first propose a resistive half adder with MTJs and Pre-Charge Sensing Amplifier (PCSA), which achieves high energy and area efficiency. Then by reusing parts of the logic circuit, we propose a magnetic full adder, which significantly reduces the area overhead and energy consumption compared with the CMOS-based full adder and the state-of-the-art magnetic full adder. This work is introduced in Chapter 3.

- Secondly, we propose **racetrack memory based Booth multipliers** as the fundamental in-memory arithmetic circuit cells. In this work, we first propose a pipelined Booth multiplier by exploring the inherent sequential access mechanism of racetrack memory, which achieves high area and energy efficiency. Then, we propose a weight-based parallel Booth multiplier to enhance the throughput performance of the proposed Booth multiplier while maintaining high area and energy efficiency. Unlike the area- and energy-consuming adder array architecture in conventional CMOS multipliers, the proposed weight-based parallel architecture parallelize the multiplication according to the bit weight in the partial products. We explore and analyze the Booth multipliers with different architectures, radix base, and input length. In addition, we propose an optimization on the Booth multiplier to transform the required write operations to shift operations, which reduces the energy consumption dramatically. This work is introduced in Chapter 4.
- Finally, we propose **a racetrack memory based two-stage modular multiplier** for specific applications, which accelerates the computation-intensive modular multiplication in various applications such as cryptography, number theory, abstract algebra, chemistry and the visual and musical arts. Our design shows the potential of the racetrack memory based computing logic for application specific designs. In this work, a novel two-stage scalable modular multiplication algorithm is proposed to significantly reduce the delay. In addition, an efficient architecture based on racetrack memory is further developed to reduce the number of required adders. Racetrack memory based application specific design for the modular multiplication shows

significant improvement compared with the state-of-the-art CMOS technology based implementation in terms of area, energy, and performance. This work is introduced in Chapter 5.

1.5 Organization

This dissertation is organized as follows. Chapter 2 reviews the previous studies. Chapters 3, 4, 5 introduce the three contributions of racetrack memory based magnetic adders, racetrack memory based magnetic multiplier and racetrack memory based application specific modular multiplier. Chapter 6 concludes this dissertation and introduces the future work.

Chapter 2

Background and Literature Review

In this chapter, we first introduce the background knowledge related to this thesis, including racetrack memory technology, adder and Booth multiplier, and Montgomery multiplication. Then we review the related work to this thesis, including applications of emerging memory technologies, logic designs for in-memory computing, and hardware implementations of modular multiplication.

2.1 Background

2.1.1 Racetrack Memory

Racetrack memory is a newly introduced non-volatile memory developed by a team led by Stuart Parkin in early 2008 [13]. The storage media of racetrack memory are the nanowires, which are like the racetracks of data. A nanowire consists of a nano-magnetic stripe, on which multiple magnetic domains are separated by non-magnetic regions called domain walls (DW). Therefore, the racetrack memory is also called domain wall memory.

Fig. 2.1 shows the structure of the racetrack memory which stores data on a magnetic stripe. Along with the stripe, each storage unit has a very short width (e.g., 50-100 nm) divided by the magnetic DW. With the shifting current at the end of magnetic stripe, the domain wall between each unit can be shifted along

Figure 2.1: Basic structure of vertical magnetic tunnel junction and the racetrack memory

with the direction of the current. The magnetic stripe can be very long, which can contain 256 bits demonstrated by the racetrack memory manufactured so far [14].

Two vertical magnetic tunnel junctions (MTJ) are used for DW nucleation and detection as shown in Fig. 2.1. The MTJ consists of two ferromagnetic (FM) layers with an oxide barrier between them. One of the FM layers has fixed magnetization direction while the other one has a free one. When the magnetization direction of the two layers is the same, the resistance of the MTJ is low, otherwise, the resistance is high. Once we have the magnetization direction of the fixed FM layer in the MTJ as a reference, we can use the magnetization direction in the free FM layer to denote two different states. Thus, the “0”, “1” binary state can be perfectly represented and stored.

According to the figure, the MTJ_W is used to write data into the racetrack memory, while the MTJ_R is used to read the data out. Such a structure makes

the racetrack memory extremely high data density and low read/write energy. However, when the data needs to be accessed, it has to be shifted to the access ports, which will cause overhead in terms of energy and access time. Therefore, a careful design should be applied to avoid the adverse effect of the sequential access mechanism of the racetrack.

2.1.2 Adder and Booth Multiplier

Adder is one of the most basic building units of any kind of multiplier and other arithmetic units. Usually, in CMOS-based computing system, the adder is implemented by CMOS transistors, which have large area overhead and energy consumption. In addition to conventional CMOS adders, there are many magnetic full adders proposed [19], [20]. Among these magnetic adders, some of them are partially non-volatile due to use of the MTJ as only one of their operands. Some of them use MTJs as both operands, however, their designs require a large number of MTJs, resulting in large overhead in terms of area and power, especially the writing power.

Multiplication is the fundamental arithmetic operation in various kinds of data intensive applications such as asymmetric encryption, image processing, *etc.* The processing of a multiplication can be divided into two stages, which are generation and addition of the partial products. According to the generation and addition of partial products, multipliers can be mainly classified into serial multipliers and parallel multipliers. The serial multiplier uses *shift and add* algorithm to realize multiplication with limited area and power while having relatively large delay. The parallel multiplier computes the partial products in parallel first, then sums them together to obtain the final result. The parallel multiplier has shorter delay but larger area and power overhead compared with the serial multiplier. In

conventional CMOS-based multiplier design, the throughput is usually regarded as the first design target because of the high area and energy budget. An adder array architecture is applied to make the best of parallelism in addition of partial products, which incurs large area overhead and energy consumption.

In order to improve the efficiency of the multiplier, many optimized multiplication algorithms are proposed. Among various kinds of optimization algorithms, Booth optimization algorithm is one of the most efficient multiplication algorithms for binary digit multiplication [26]. Booth algorithm focuses on the optimization of generating partial products. With different radix, the Booth algorithm is modified into radix-2 modified Booth algorithm, radix-4 modified Booth algorithm and so on. Among these modified Booth algorithms, modified Booth algorithm with radix-4 is the most suitable one for binary implementation because efficient shift operations are mainly involved. It also has a fixed number of partial products and thus eases the implementation of related Booth encoder and decoder.

2.1.3 Montgomery Multiplication

We use RSA as an example asymmetric cryptography algorithm to illustrate our multiplication design since RSA algorithm is widely used with high reliability [27]. In the encryption and decryption operations, the modular exponentiations are computed using equations Eq. 2.1 and Eq. 2.2, where C and P are cipher and plain text, respectively. E and D are encryption key and decryption key, respectively, while M is the modulus.

$$C = P^E \bmod M \quad (\text{Eq. 2.1})$$

$$P = C^D \bmod M \quad (\text{Eq. 2.2})$$

The encryption and decryption operations of RSA algorithm are very costly in computing time and resource. In order to accelerate the modular multiplication, Montgomery multiplication algorithm is proposed. It replaces the costly division operation in modular operation with the efficient shift operation [28].

The Montgomery modular multiplication works as follows. For an n -bit modulus M , given two integers x and y , the M residues of x and y with respect to r can be expressed as: $X = x \times r \pmod{M}$, $Y = y \times r \pmod{M}$, where r is an integer that can ease the division. The Montgomery product of X and Y is equal to the M residues of xy with respect to r , which is shown as:

$$Z = X \times Y \times r^{-1} = x \times y \times r \pmod{M} \quad (\text{Eq. 2.3})$$

Assuming x equals to y and with X precomputed, the M residues of x^n with respect to r can be recursively computed using Equation Eq. 2.3. Then it is convenient to obtain M residues of x^n . r is usually set as 2^n to replace division operations with shift operations. Since the core operation of encryption and decryption is Montgomery multiplication, the efficiency of Montgomery multiplication determines the overall efficiency of the whole cryptography algorithm.

Because Montgomery multiplication algorithm is a very efficient method for modular multiplication, many researchers have implemented it for general-purpose processor and application specific integrated circuit (ASIC) [29, 30, 31, 32]. However, none of them has proposed emerging memory based design of Montgomery multiplication.

2.2 Literature Review

2.2.1 Applications of Racetrack Memory

In this section, we review the related work to emerging memory technologies, especially the racetrack memory and its applications in CPUs, GPUs, and application-

specific processors.

2.2.1.1 Emerging Memory Technologies

Emerging memory technologies are the foundation of the emerging memory based logic. According to the data storage medium, the traditional memory technologies such as DRAM and flash are categorized as charge-based memories which store data by charging on capacitance. The emerging memory technologies are categorized as resistive memories which store data by adjusting the resistance of its storage medium. The resistive memories are non-volatile, which protect the valuable data against the loss of electrical power.

Phase change memory (PCM) [33] is one of the most commonly adopted non-volatile emerging memories. The resistance of PCM is determined by the phase of phase change materials [9, 34]. A lot of studies exploit the possibility of using PCM as the memory for the future computer systems [35, 36, 37, 38, 39, 40]. However, PCM suffers from high programming current density and limited write endurance [41, 42, 43].

The resistance of ReRAM (memristor) is determined by atom distance [10]. ReRAM typically refers to the electrical switching between different resistance states observed in numerous metal oxides (*e.g.*, NiOx, HfOx, TiOx, TaOx, PrxCa1-xMnO3), although similar phenomenon has also been reported in non-oxide materials (*e.g.*, silicon, sulfides, chalcogenides). Due to its promising scalability, CMOS compatibility, and high performance, it attracts great attention [44]. Research interest on ReRAM materials is lasting, for example, the research on resistive switching in ALD HfO_x and TaO_x [45, 46]. However, the theory and related technology of the memristors are far from mature [47, 48].

The resistance of Spin-transfer torque magnetic random-access memory (STT-MRAM) is determined by magnetic polarity [12, 49]. STT-MRAM has advan-

tages of high performance, low power, and unlimited endurance. Due to its advantages, it is widely explored as the main memory in modern computer systems [50, 51]. Especially for its high performance, it is also regarded as one of the candidates for future cache memory [52, 53, 54, 55].

Racetrack memory is a recently introduced emerging non-volatile memory which stores the data based on the magnetic polarity too [13]. Unlike STT-MRAM using MTJ as the main storage element, racetrack memory stores data in ferromagnetic nanowires [56, 57]. A racetrack memory nanowire can store multiple bits, which achieves high data density. According to the racetrack memory manufactured so far [14], a single nanowire can contain 256 bits. Compared with STT-MRAM, it has much higher data density due to the advantages of multiple-bit store and access ports share. [56]. However, access ports share causes the sequential access mechanism, which has negative influence over the performance [58]. There are many studies proposed to utilize the advantages of racetrack memory [59, 60, 61, 62]. Because of its high data density, high speed, and low power consumption, racetrack memory is promising to be applied in all memory hierarchy from register to external storage [63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 62]. Existing studies of using racetrack memory in CPUs mainly focus on building cache systems. Some studies proposed racetrack memory based designs for traditional L1 and L2 caches, while others proposed designs for scratchpad memory.

2.2.1.2 Racetrack Memory in CPUs

For cache memory in CPUs, Venkatesan *et al.* [65] explore the use of racetrack memory within the on-chip L1 and L2 cache of general purpose computing platforms. In order to address the impact of sequential access, several management

schemes at the device, circuit, and architecture levels are proposed, including data migration policies, bit-interleaving scheme and so on. Due to the higher data density, their cache design achieves large area overhead and energy consumption reduction compared to both STT-MRAM and SRAM caches. Venkatesan *et al.* [67] also present another L2 cache design, where a pre-shifting approach is adopted to hide shift latency. When the cache is accessed, a prediction is performed to predict the next block to be accessed. Then the expected block is pre-shifted to its statically assigned port. If the prediction is accurate, then the shift operation is avoided otherwise the accessed block is shifted to the nearest read port. Compared with SRAM and STT-MRAM caches, their design improves the efficiency in terms of performance and energy consumption. Zhang *et al.* [73] propose a nanowire organization that can reduce the access energy. Usually, in racetrack memory designs, multiple heads of a track share the same bitline and sourceline, which leads to low heads utilization. In their design, multiple heads of a track share the sourceline but have different bitlines, which alleviates contention on shared bitline and enables the parallel access. By virtue of their technique, the number of required shift operation is reduced and energy efficiency is improved.

For scratchpad memory, Mao *et al.* [74] explores three data placement strategies to reduce the shift overhead including *first come first store*, *most access in middle* and *most access first*. In order to cope with the dynamics of the working sets, a genetic algorithm is adopted to perform data allocation in racetrack memory. For large capacity storage applications, Park *et al.* [75] propose a racetrack memory based graph processing approach. In their design, racetrack memory provides byte addressability with low read and write latency. With pointer-assisted graph representation, the performance of graph algorithms is enhanced. Compared with graph processing approaches based on SSD, the performance of

their approach is improved because of the lower access latency of racetrack memory.

2.2.1.3 Racetrack Memory in GPUs

Racetrack memory is proposed as various components in GPUs, such as shared memory, register file, and GPU cache. Venkatesan *et al.* [66] design the entire cache hierarchy for GPUs with racetrack memory, including L1 cache, L2 cache, instruction cache, shared memory, and constant cache. They propose architecture management policies to reduce the shift latency incurred by the sharing of access ports. A racetrack memory based fully associative buffer is designed to alleviate the large shift penalties. Compared with SRAM and STT-MRAM based design, their designs show higher performance and energy saving. Atoofian and Saghir [72] present an address predictor based technique to hide shift penalties in racetrack memory based L2 cache in GPUs. Three address predictors, including a stride predictor, a context predictor, and a hybrid predictor, are proposed. They show that the hybrid predictor which combines the stride and context predictor has the highest performance. The racetrack memory based cache with the hybrid predictor has comparable performance with the SRAM based caches.

Due to the advantages of high data density, racetrack memory is also promising in the register file (RF) design of GPUs. The capacity requirement of RF in GPUs is much larger than in CPU, it makes the racetrack memory an attractive candidate for the RF in GPUs [76]. Many researchers use racetrack memory to implement high capacity GPU RF. Various techniques are proposed to reduce the shifting overhead of racetrack memory based RF, such as pre-shifting [77, 71, 69], warp scheduling [63], and intelligent register mapping [77, 63].

2.2.1.4 Racetrack Memory in Application-Specific Processors

The holistic efficiency in area, power, and performance makes the racetrack memory a suitable memory material for application-specific processors, where different memory structures show different access properties and capacity requirements. Venkatesan *et al.* [64] explore the application of racetrack memory in recognition and mining processors. In such a processor, racetrack memory is used to build the first memory level, a 64KB FIFO, where streaming read operations dominate. They show that their spin-based design is more efficient in both performance and energy consumption compared with the CMOS-only design. Chung *et al.* [78] propose a racetrack memory design to exploit serial access for area- and energy-efficient digital signal processor (DSP). Because many DSP applications are featured by sequential memory access, they present racetrack memory based embedded memories for DSP, such as the LIFO in the Viterbi decoder, an input register of FIR filter, and FIFO RFs in the FFT processor. Compared with SRAM and STT-MRAM based designs, racetrack memory based design is more efficient in area overhead and power consumption.

2.2.2 Logic Design for In-memory Computing

2.2.2.1 CMOS Technology Based Logic Design

As in-memory computing shows advantages in both the performance and energy efficiency, it attracts a lot of research interests. A lot of in-memory computing architectures are proposed in the last decades, such as integrating processing units and main memory in a single chip [79, 80, 81, 82] as well as the emerging 3D integration technology [83, 84].

A key issue for in-memory computing systems is the type of the processing elements. Conventional CMOS technology based logics are well-developed [85,

86, 17, 87, 88]. 1-bit half adder and 1-bit full adder are included in the standard cell library for the convenience of designers [89, 90].

Usually, a multiplier is built using adders as basic elements. The operation of a multiplier can be divided into two stages: partial products generation and partial products addition. According to the generation of partial products, multipliers can be mainly classified into serial multipliers and parallel multipliers. The serial multiplier uses “shift and add” algorithm to realize multiplication with limited area and power cost but having relatively large delay. The parallel multiplier computes the partial products in parallel first, and then sums them together to obtain the final result. The parallel multiplier has shorter delay but larger area and power overhead compared with the serial multiplier. There also exist many algorithms to optimize the multiplication such as Karatsuba’s algorithm [91], Toom-Cook algorithm [92] and Booth algorithm [93]. Among various kinds of optimization algorithms, Booth optimization algorithm is one of the most efficient multiplication algorithms for binary digit multiplication, while other optimization algorithms are more efficient in scenario where operands are not binary digits [26]. However, conventional CMOS based arithmetic circuit designs usually consume large area and power. Therefore, they are not feasible for in-memory computing systems due to the tight area and power budgets.

2.2.2.2 Magnetic Logic Design

With the shrinking of technology node, the power efficiency of CMOS technology based logic is dominated increasingly by the leakage power [94]. Magnetic logic has ultra-low static power and attract the interest of researchers. Various kinds of magnetic logic designs are proposed in order to improve power efficiency of in-memory computing systems. Chabi *et al.* [95] propose a magnetic

flip-flop to reduce the static power by the combination of checkpointing operation, power gating and self-enable mechanisms. They show that their design has lower power consumption compared to conventional CMOS technology based flip-flop. Onkaraiah *et al.* [96] present a non-volatile flip-flop based on Bipolar ReRAMs (Bi-RNVFF). The proposed flip-flop achieves high power efficiency and safe “power-on, power-off” property. Zhao *et al.* [97] propose magnetic flip-flop and programmable logic function look-up tables (LUTs). The proposed magnetic flip-flop has much smaller area overhead and power consumption than any existing CMOS technology based flip-flop. In addition, because of the non-volatility of the magnetic device, magnetic LUT (spin-LUT) enables non-volatile and programmable implementation of logic functions. Thus, the magnetic programmable circuits based on the magnetic LUTs achieve both real “instant-on” and dynamical reconfiguration.

As the basic element of arithmetic circuits, magnetic adders are also proposed to alleviate the severe power concern. According to the size of addends, adders are categorized into 1-bit adders and multi-bit adders. Meng *et al.* [19] propose a 1-bit magnetic full adder in 2005 based on programmable spintronic logic devices for future magnetic CPUs. Matsunaga *et al.* [20] have fabricated a 1-bit non-volatile full adder based on logic-in-memory architecture using MTJs in combination with MOS transistors, which verifies the feasibility of the magnetic adder. Ren *et al.* [23] conduct a comprehensive energy and performance analysis on a 1-bit magnetic full adder and compare it with the CMOS full adder. Gang *et al.* [22] make effort for the reliability of the magnetic adder by presenting a 1-bit magnetic full adder with thermally assisted switching MTJ cell. Deng *et al.* [21] present a 1-bit magnetic full adder based on STT-MRAM. By the virtue of hybrid MTJ/CMOS design, it provides advantageous power and area efficiency compared with conventional CMOS-only full adder.

In addition to 1-bit magnetic full adder, a few studies have been reported for multi-bit ones. Trinh *et al.* [25] propose a magnetic adder based on racetrack memory in 2013. According to their design, a 1-bit non-volatile magnetic full adder is proposed. Their magnetic full adder is a MTJ/CMOS hybrid design composed of MTJ logic tree and PCSA circuits, which occupy 14 transistors and 16 MTJs. With the proposed adder as the building element, a serial multiple-bit adder is implemented by shifting the operands to the adder inputs to perform serial addition. In their design, an 8-bit serial adder is realized and analyzed. Compared with CMOS technology based serial adder, their design significantly reduced area overhead and energy consumption.

Deng *et al.* [24] present a synchronous 8-bit non-volatile full adder based on MTJs. Instead of shifting the operands to the adder inputs to perform serial addition, they implement the 8-bit adder in the carry-ripple manner. Three synchronous 8-bit non-volatile full adder structures are proposed considering the trade-offs between performance and energy consumption. However, eight cycles are required to read or write the eight bits data in the proposed 8-bit full adder. Compared with CMOS technology based carry ripple adders, it achieves full non-volatility and nearly zero leakage power. Yuhao Wang *et al.* propose an in-memory Advanced Encryption Standard (AES) encryption design based on racetrack memory [98]. However, their design is for AES, a symmetric encryption scheme which needs many unique logic operations. Thus, the generality of the design is limited.

2.2.3 Hardware Implementations of Modular Multiplication

Modular multiplication is widely used in various applications such as cryptography, number theory, group theory, ring theory, knot theory, abstract algebra,

computer algebra, computer science, chemistry and the visual and musical arts. It is the most important operation in many public-key cryptosystems such as RSA (RivestShamirAdleman) and Elliptic-curve cryptography (ECC) cryptosystems [27, 99, 100, 101, 102]. However, the key size of the contemporary cryptosystem is very large because of the security requirement along with the ever-increasing computing speed. For instance, it is claimed that 2048-bit keys are sufficient until 2030 and an RSA key length of 3072 bits should be used if security is required beyond 2030 [103]. Due to the large key size, the operands of the modular multiplication are large integers, which incurs long computation time and large resources overhead. Thus, numerous algorithms and hardware implementations have been presented to perform the modular multiplication with large integers more efficiently. Montgomery algorithm is one of the most widely used one to accelerate the modular multiplication by replacing the costly division operations with a series of shifting operations [28, 104, 105]. Because Montgomery multiplication algorithm is a very efficient method for modular multiplication, many researchers have proposed specific hardware designs to enhance its performance. According to the input precision, hardware implementations of Montgomery modular multiplication can be categorized into fixed-precision implementations and variable-precision implementations.

2.2.3.1 Fixed-precision Implementations

In fixed-precision implementations, multiplicand and modulus are processed in full-precision while the multiplier is processed bit by bit [106, 107, 108, 109, 110, 111, 112]. In order to improve performance, high-radix implementations with fixed-precision have also been proposed [107, 113]. However, the high-radix implementations are usually complex. The performance overhead caused

by the implementation complexity offsets the desired performance improvement. Walter *et al.* [114] conduct an exhaustive theoretical investigation of the high-radix modular multipliers for the design trade-offs between performance and area overhead. Compared with high-radix designs, low-radix designs attract more research interests because it is more suitable for hardware implementations [115].

Fixed-precision implementations lack scalability when considering the possible increase of the operand size. Because the digits in multiplicand and modulus are handled as a whole, the operand size cannot be altered, otherwise the hardware needs to be redesigned. Thus, the fixed-precision implementations are prone to obsolescence. In addition, the fixed-precision implementations have problems of large fan-out of signals, complex routing, and large wire delays due to the large operand size. Systolic architectures can help to improve these problems at the cost of extra hardware resources [116, 117].

2.2.3.2 Variable-precision Implementations

In order to address the problems of the fixed-precision implementation, variable-precision implementations with scalable architecture are proposed [118, 119, 32, 120, 30, 121, 122].

Tenca and Koc [118, 32] propose the first variable-precision implementation of the Montgomery modular multiplication. With their implementation, a word-based data path is presented that can perform Montgomery modular multiplication with any precision as long as the input operands are expressed in the word manner. In their design, the multiplicand and modulus are processed word by word while the multiplier is processed bit by bit. This scalable architecture has two major advantages. Firstly, it addresses the concern of scalability. Since this architecture processes the multiplicand and modulus word by word, it can process

operands with larger sizes after the hardware fabrication. Secondly, the high fan-out problem is also alleviated as the word size is much smaller than the operand size.

Tenca *et al.* [119] propose a high-radix variation of the scalable architecture of the Montgomery modular multiplication. In their design, radix-8 is used as an example to illustrate the high-radix design. They adopt Booth-recoding techniques to improve the efficiency of the multiplication. However, it also suffers from the large overhead caused by the complexity of high radix. Harris *et al.* [120] improve the performance of the scalable architecture by processing the operand in the reverse manner and implement their design on a field-programmable gate array (FPGA). However, the extra performance overhead of the proposed architecture offsets the desired improvement. Erkey *et al.* [123] and Akashi *et al.* [124] implement Montgomery modular multiplication with scalable architecture with CMOS technology based ASIC. They analyze the design trade-offs between different design parameters such as the word length, the number of the pipeline stages. Their analysis shows that a pipeline consisting of several stages is more efficient than a single unit processing very long words. Working with relatively short words diminishes data paths in the final circuit, which reduces the required bandwidth.

Chen *et al.* [125] design a microcode-based architecture with a reconfigurable data path controlled by the micro-instruction for Montgomery modular multiplications in various public-key cryptosystems. Huang *et al.* [121] propose two new hardware architectures to reduce the latency by pre-computing partial results using two possible assumptions regarding the most significant bit of the previous word. However, their design lacks scalability and needs extra hardware resources. When the word size is small, the area overhead of their design is very

large. Shieh *et al.* [30, 31] propose scalable Montgomery modular multiplication architectures to reduce the delay and memory access by manipulating the intermediate variables and dedicated task scheduling. However, the complexity of the proposed scheme degrades the performance and area efficiency.

2.3 Summary

This section introduces the background knowledge related to this thesis, including racetrack memory technology, adder and Booth multiplier, and Montgomery multiplication. This section also reviews the relevant studies in emerging memory technologies, especially the racetrack memory technology. The logic designs of general computing cells with CMOS-only technology and CMOS/magnetic hybrid technology are reviewed too. For the application specific design, hardware implementations of modular multiplication for asymmetric cryptography are analyzed. Based on the previous studies, we propose three new studies for racetrack memory based logic design. Our work shows innovation in the following aspects.

Previous studies in 1-bit magnetic full adders are mostly partially non-volatile [20, 19, 22]. Because these studies of magnetic full adders only use MTJ as one of their operands. Thus, the advantages of spintronic devices are not fully exploited. Moreover, it is difficult to extend these 1-bit magnetic full adders to multi-bit structure. Magnetic full adders like [25, 24] are fully non-volatile, however, their designs require a large number of MTJs, resulting in large overhead in terms of area and power, especially the writing power. For example, the magnetic full adder in [25] costs 14 transistors and 16 MTJs, which means that 16 bits data need to be written and shifted to the adder, consuming considerable amount of power. When compared with previous designs, our work has two innovative points. Firstly, our proposed adders are fully non-volatile. The logic circuits

can be safely turned off. Secondly, by reusing parts of the logic design, the proposed full adder significantly reduces the area overhead and energy consumption compared with the state-of-the-art magnetic full adder [25].

Previous studies in CMOS technology based multipliers adopt adder array architecture to maximize the speed [17, 88]. However, the adder array occupies large area and consumes considerable amounts of energy. In order to improve the area and energy efficiency, our work develops a pipelined low-power Booth multiplier by exploring the inherent sequential access mechanism. The multiplier is deeply pipelined to exploit the advantages of racetrack memory while avoiding the adverse impact of its sequential access mechanism. In order to increase the throughput, our work proposed a weight-based parallel architecture. The proposed weight-based parallel architecture parallelizes the multiplication according to the bit weight in the partial products. In order to ensure high energy efficiency, we propose an optimization that transforms the energy-demanding write operations to shift operations. With the proposed optimization, the weight-based parallel multiplier achieves high area and energy efficiency while maintaining high performance in throughput.

Previous studies in scalable Montgomery modular multiplication are word-based parallel implementations with hardware pipelining, in which the operations in different iterations of the outer loop are executed in parallel to reduce the latency at the cost of more hardware overhead [32]. However, the data dependency between two successive iterations in the algorithm limits the throughput. Various techniques are proposed to solve this problem at the cost of high design complexity and large overhead [30, 31]. Compared with previous works, our work has three innovative points. Firstly, our work proposes a novel two-stage scalable modular multiplication algorithm to address the data dependency problem

at nearly no extra cost. Secondly, our implementation is based on the racetrack memory, which is very efficient in terms of area overhead and power consumption. Finally, our work has an optimized architecture for the proposed algorithm.

Chapter 3

Racetrack Memory Based Magnetic Half Adder and Full adder

In-memory computing has been demonstrated to be an efficient computing infrastructure in big data era for many applications such as graph processing, encryption, *etc.* Racetrack memory is a newly introduced memory technology. It allows high data density fabrication and thus is a good fit for in-memory computing. However, building in-memory arithmetic circuits together with the memory cells for in-memory computing affects both the memory density and power efficiency. And the conventional well-developed CMOS-based arithmetic circuits don't fulfil the more stringent area and energy constraints. As a result, it remains challenging to build efficient in-memory arithmetic circuits on racetrack memory for general in-memory computing. To that end, we present a series of fundamental in-memory arithmetic circuits as in-memory computing cells including adders and multipliers. The presented circuits perform the logical operations based on the memory cell and reuse part of the peripheral circuits and can be constructed efficiently using racetrack memory technique. The resulting design realizes high area and energy efficiency while maintaining considerable performance.

To build the arithmetic circuits, in this chapter, we propose highly area and energy efficient racetrack memory based magnetic half adder and full adder. Ex-

perimental results show that the proposed magnetic full adder saves 66% area and 56.3% energy compared with the state-of-the-art magnetic full adder.

3.1 Introduction

With the development of information technology, we are entering the big data era, where large amount of data needs to be created, processed and transferred in the cloud. Owing to the increase of memory capacity, in-memory computing has become a popular computing infrastructure for many applications in big data, such as graph processing and encryption.

With in-memory computing, computation is executed in memory, which bridges the gap between memory and processor. In addition, the data transfer between the memory and the processor for simple arithmetic operations can be avoided. Thus, the power consumption of the data transfer in many applications can be dramatically reduced. As in-memory computing can be beneficial to applications on both performance and energy efficiency, it attracts a lot of research interests. A lot of in-memory computing architectures are proposed in the last decades, such as integrating processing units and main memory in a single chip [79, 80, 81, 82] as well as emerging 3D integration technology [83, 84].

As the storage and logic media, the memory technology and the embedded computing logic units are critical for an in-memory computing system. Racetrack memory is a newly introduced memory technology that has the advantages of high data density, non-volatility, low power, and high speed [13]. It is a promising technology that not only can be used in all hierarchy of memory systems from external storage to register files, but also can be applied to logic designs for in-memory computing. Despite the great potential of using racetrack memory for in-memory computing, there is still a lack of efficient general arithmetic circuits.

In this chapter, we present racetrack memory based in-memory arithmetic circuits including a magnetic half adder and a magnetic full adder.

Since addition is one of fundamental arithmetic operations, which is also one of fundamental arithmetic operations in various kinds of data intensive applications such as compression, image processing, *etc*, the proposed racetrack memory based arithmetic designs are general and applicable to many in-memory computing applications. We first present a racetrack memory based magnetic half adder. Then we present a racetrack memory based magnetic full adder on top of the half adder. Compared with the state-of-the-art magnetic full adder design, our proposed magnetic full adder saves 66% area and 56.3% power [25].

The rest of the chapter is organized as follows. In Section 3.2, the racetrack memory based half adder is presented. Section 3.3 presents our design of the racetrack memory based full adder on top of the half adder design. In Section 3.4, experimental results are presented and analyzed. Section 3.5 concludes this chapter.

3.2 Racetrack Memory Based Magnetic Half Adder

Addition is the fundamental building components of the arithmetic operators. Its area and energy efficiency is thus of vital importance to the in-memory computing cells. The half adder is the simplest arithmetic circuit, which is more efficient than the full adder in the scenario with no carry-in signal. Thus, half adder design is indispensable in arithmetic circuit designs. Half adder is to add two digits A and B and to produce two outputs i.e. sum and carry-out as given by the following equations. The carry signal represents an overflow into the next digit of a multi-digit addition. It is clear that eventually XOR and AND logic need to

be implemented for a half adder. The logic functions of a half full adder (HA) are given by Eq. 3.1 and Eq. 3.2:

$$Sum = A \oplus B \quad (\text{Eq. 3.1})$$

$$C_o = A \cdot B \quad (\text{Eq. 3.2})$$

where A and B are two addends, Sum is the result bit while C_o is carry-out signal to the next stage.

In order to save the energy and area, we use two MTJs to implement the AND logic and one MTJ to implement the XOR logic respectively. Then the MTJ is seated along with the pre-charge sense amplifier (PCSA) and the computation can be performed at the pre-charge phase. Fig 3.1 shows the schematic of our proposed half adder. The left side circuit produces Sum of input A and B while the right side generates the C_o . The processing is similar and here we just take C_o generation as an example to detail the computation process. At the pre-charge phase, CLK is set to “0”, MP5 and MP8 are turned on, and both C_o and $\overline{C_o}$ are charged to “1” in this phase. When CLK is changed to “1” in the evaluation phase, MP5 and MP8 are turned off while MN6 is turned on. The two bitlines start to discharge. The bitline with lower resistance, taking bitline of with MTJs as an example, would reach the threshold voltage of PMOS transistor in its opposite branch, namely MP7, faster than the branch with higher resistance. Once the gate voltage of the PMOS reaches the threshold, it is turned on and force the $\overline{C_o}$ to “1”, which would in turn force the C_o to “0”. Thus, we get two complementary outputs based on the resistance information stored in the MTJs and the resistor.

A two-input AND logic can be implemented with two serial MTJs. Given different input of A and B , resistance of this bitline can be calculated as shown in Table 3.1. Note that R_L and R_H stands for resistance of a low resistance

Figure 3.1: The schematic of proposed magnetic half adder

Table 3.1: Truth table of the carry function for the half adder

A	B	R_{left}	R_{right}	C_o
0	0	$2R_L$	$4R_L$	0
0	1	$R_L + R_H$	$4R_L$	0
1	0	$R_H + R_L$	$4R_L$	0
1	1	$2R_H$	$4R_L$	1

state and high resistance state of a MTJ, respectively. In our model, R_H equals to $2.5R_L$. To obtain the AND logic with the PCSA circuit, the resistor on the complementary bitline can be set to be $4R_L$.

For a two-input XOR function, the output would be “1” if and only if the two inputs are different. It is very efficient to be realized by MTJ because it is consistent with the intrinsic property of the MTJ. We stack the A/B together with oxide barrier between them to realize the XOR operation. Previous papers and industrial works show that this structure has the same high/low resistance behaviour with the MTJ which has a fixed reference layer [126, 98, 127]. The

Table 3.2: Truth table of the sum function for the half adder

A	B	R_{left}	R_{right}	sum
0	0	R_L	$2R_L$	0
0	1	R_H	$2R_L$	1
1	0	R_H	$2R_L$	1
1	1	R_L	$2R_L$	0

schematic of the Sum logic part is also shown in Fig. 3.1. The resistance of the resistor is set to $2R_L$. With R_H equals to $2.5R_L$ in our model, we can get the truth table of the Sum function, which is shown in Table 3.2. After realizing these two logic functions, we get a half adder.

3.3 Racetrack Memory Based Magnetic Full Adder

In this section, we illustrate our full adder design. The logic functions of a full adder (FA) are given by the following equations:

$$Sum = A \oplus B \oplus C_i \quad (\text{Eq. 3.3})$$

$$C_o = A \cdot B + A \cdot C_i + B \cdot C_i \quad (\text{Eq. 3.4})$$

where A and B are two addends, Sum is the result bit while C_i and C_o are carry-in from previous stage and carry-out to the next stage respectively.

Fig. 3.2 shows the schematic of our proposed full adder. According to the figure, the FA mainly consists of two parts, one is for Sum generation and the other one is for C_o generation. As shown in Fig. 3.2, in the similar way with the half adder design, we use pre-charge sense amplifier (PCSA) to read the information stored in the racetrack memory according to their resistance. As shown in equation Eq. 3.4, the logic function of C_o is in fact a majority function. As long as there are more than one “1” in the three input data, the output of the function is “1”. Hence, we use serial connection of 3 MTJs to form the left branch and a

Figure 3.2: The schematic of proposed magnetic full adder

resistor to form the right branch. The resistance of the resistor is $2R_H$. With R_H equals to $2.5R_L$ in our model, we can get the truth table of the carry function, as shown in Table 3.3.

Sum logic part is the critical part of our design, which leads to considerable amount of power saving compared to the previous MFA design [25]. As shown

Table 3.3: Truth table of the carry function of the full adder

A	B	C_i	R_{left}	R_{right}	C_o
0	0	0	$3R_L$	$2R_H$	0
0	0	1	$2R_L + R_H$	$2R_H$	0
0	1	0	$2R_L + R_H$	$2R_H$	0
0	1	1	$R_L + 2R_H$	$2R_H$	1
1	0	0	$2R_L + R_H$	$2R_H$	0
1	0	1	$R_L + 2R_H$	$2R_H$	1
1	1	0	$R_L + 2R_H$	$2R_H$	1
1	1	1	$3R_H$	$2R_H$	1

Table 3.4: Truth table of the sum function of the full adder

A	B	C_i	R_{left}	R_{right}	C_o	Sum_{in}	Sum
0	0	0	$2R_L$	R_H	0	0	0
0	0	1	$R_L + R_H$	R_H	0	1	1
0	1	0	$2R_H$	R_H	0	1	1
0	1	1	$R_L + R_H$	R_H	1	1	0
1	0	0	$R_L + R_H$	R_H	0	1	1
1	0	1	$2R_H$	R_H	1	1	0
1	1	0	$R_L + R_H$	R_H	1	1	0
1	1	1	$2R_L$	R_H	1	0	1

in Eq. 3.3, the Sum equals to the three inputs doing XOR operation together. It is a complex logic operation that needs many resources to realize in a conventional manner. Eq. 3.5 shows the logic function presented using basic logic operations.

$$A \oplus B \oplus C_i = A \cdot B \cdot C_i + A \cdot \overline{B} \cdot \overline{C_i} + \overline{A} \cdot B \cdot \overline{C_i} + \overline{A} \cdot \overline{B} \cdot C_i \quad (\text{Eq. 3.5})$$

Instead of using a large number of MTJs to form the logic tree like the previous design, we propose a new structure to realize the logic function for area and power efficiency, which is shown in the left half of Fig. 3.2. The idea lies in the fact that since the logic function of the carry out is a majority function, we can make use of the information carried within C_o . We stack the A/B and B/C_i together with oxide barrier between them to realize XOR operation. Previous papers and industrial works show that this structure has the same high/low resistance behaviour with the MTJ which has a fixed reference layer [126, 98, 127]. A , B and C_i are shifted into this structure from other parts of the racetrack memory.

As shown in Fig. 3.2, for the ease of explanation, we introduce the interim Sum signal and its complementary signal which are denoted as Sum_{in} and $\overline{Sum_{in}}$ respectively. As shown in Table II, the R_{left} column shows the resistance of left branch in different input patterns. We use a resistor as the right branch with the resistance of R_H whose value equals to $2.5R_L$. A 2-to-2 MUX is added in the

Table 3.5: Main parameters in the RM model

Parameter	Description	Default value
W_{RT}	Width of racetrack	$1F$
G_{RT}	Gap distance between two racetracks	$1F$
L_D	Length of the domain in a racetrack	$2F$
L_{RT}	Length of racetrack	$128F$
T_{RT}	Thickness of racetrack	$6nm$
p	PMA racetrack nanowire resistivity	$4.8 \times 10^7 \Omega m$
$TMR(0)$	TMR with $0 V_{bias}$	150%
W_{EN}	Write energy	$1pJ$
$J_{nucleation}$	Critical current density for write	$5.7 \times 10^6 A/cm^2$
W_{DE}	Write latency	$5ns$
S_{EN}	Shift energy	$0.051pJ$
J_{shift}	Critical current density for shift	$6.2 \times 10^7 A/cm^2$
S_{DE}	Shift latency	$500ps$

adder to select the correct output. The MUX takes Sum_{in} and $\overline{Sum_{in}}$ as inputs, Sum and $\overline{Sum}$ as outputs while C_o acts as a select signal. The truth table of Sum regarding to A , B , C_i and Sum_{in} is shown in Table 3.4.

3.4 Experiment Results

CMOS 45 nm design kit [89] and a model of perpendicular magnetic anisotropy (PMA) racetrack memory based on CoFeB/MgO structure [128] have been used to perform SPICE simulations for the proposed adders. The main parameters of this PMA racetrack memory model are described in the Table 3.5.

Based on the parameters in the Table 3.5, we simulate our adder with HSPICE. Table 3.6 shows the results of the magnetic half and full adders and the CMOS-based full adder. Note that the delay shown in the table is the worst case delay while the energy is the average energy obtained by the activity set as 0.5 for all the input ports. The ‘‘Previous MFA’’ is the MFA proposed in [25]. As shown in the Table 3.6, our proposed magnetic half adder denoted by ‘‘Proposed MHA’’

Table 3.6: Comparison of the proposed half adder and three full adders

	Delay	Energy	Write operation	Area
CMOS FA	$100ps$	$15fJ$	NA	$11.04\mu m^2$
Previous MFA	$180ps$	$7.6fJ$	16	$3.36\mu m^2$
Proposed MFA	$240ps$	$19fJ$	7	$1.142\mu m^2$
Proposed MHA	$153ps$	$16.1fJ$	4	$0.992\mu m^2$

has a shorter delay and lesser energy consumption compared with the proposed magnetic full adder denoted by “Proposed MFA”. The delay of the half adder is $153ps$ while the energy consumed is $16.1fJ$. The area overhead of the half adder is slightly smaller than the full adder, which is $0.992\mu m^2$. It is reasonable because the half adder has no need to deal with the *carry-in* signal.

As shown in the Table 3.6, our proposed full adder has a longer delay than the other two FAs. This is because a MUX is added in the adder to trade for power and area. However, the longer delay would make nearly no difference between previous MFA and our proposed MFA, because the write latency of RM is $5ns$, which dominates the latency of the adder. According to the Table 3.5, even the shift delay is much larger than the delay of the adder. Therefore, when the adder runs with the input data, the total delay would be limited by the shift and write delay of the operands. Although our proposed MFA has a slightly larger computing energy than the previous MFA, the number of writes needed by our proposed MFA is much smaller than the previous MFA. As shown in the Table 3.5, the write energy of the RM is at pJ level, thus, the computing energy can be ignored when considering writing energy. In this point of view, our proposed MFA saves 56.3% write energy compared with the previous MFA.

3.5 Summary

In this chapter, we propose racetrack memory based arithmetic circuits for general computing operations. The proposed arithmetic circuits include a magnetic half adder and a magnetic full adder. By reusing parts of the logic circuit, the proposed magnetic full adder significantly reduces the area overhead and energy consumption compared with the CMOS-based full adder and the previous magnetic full adders. The experiment results show that our proposed magnetic full adder can save 56.3% write energy compared with the previous state-of-the-art magnetic full adder.

Chapter 4

Racetrack Memory Based Booth Multiplier

In the previous chapter, we proposed racetrack memory based magnetic half and full adders as in-memory computing cells for general computing operations. The proposed adders achieve high area and energy efficiency. In addition to addition, multiplication is also one of the fundamental computing operations in arithmetic computing. High efficiency multipliers are critical for high performance computing systems. In this chapter, with the adder proposed in the previous chapter as the base element, we propose a pipelined Booth multiplier by exploiting the inherent sequential access mechanism of racetrack memory, which achieves high area and energy efficiency.

In order to enhance the throughput, we develop a weight-based in-memory Booth multiplier with different radix base and different input bit length. Unlike the area- and energy-consuming adder array architecture in conventional CMOS-based designs, the proposed multiplier parallelizes the addition of partial products according to the bit weight in the final result. In order to maintain high energy efficiency, we propose an optimization that can transform the energy-consuming write operations to shift operations. By transferring write operations to shift operations, the energy efficiency of the arithmetic circuits can be greatly improved.

In combination with the write optimization, the weight-based parallel Booth multiplier achieves $3.1Pj/bit$ for 16-bit x 16-bit multiplication size.

4.1 Introduction

Many in-memory magnetic arithmetic circuits are proposed to enhance the area and energy efficiency of the in-memory computing systems. However, the performance of the magnetic arithmetic circuits is either affected by the energy- and time-demanding write operations or the performance-inefficient architecture. In order to address this challenge, in this chapter, we present racetrack memory based in-memory Booth multipliers. Since the multiplication is a fundamental arithmetic operation, which is widely used in various kinds of data intensive applications such as compression, image processing, *etc*, the proposed racetrack memory based Booth multiplier is general and applicable to many in-memory computing applications. In this chapter, we first present a pipelined Booth multiplier by exploring the inherent sequential access mechanism of racetrack memory, which achieves high area and energy efficiency. Second, in order to enhance the throughput, we propose weight-based Booth multipliers with different radix base and bit length. At last, with the observation that the write operation consumes much larger energy than the shift operation, we optimize the arithmetic circuits to improve the energy efficiency by transforming the required write operations to shift operations. Venkatesan *et al.* proposed a shift based write to improve the efficiency of racetrack memory [67]. However, their design is a device level design, which is different from our approach. The key contributions of this work can be summarized as:

- (i) A pipelined Booth multiplier is proposed by exploiting the inherent sequential access mechanism of racetrack memory, which achieves high area and

energy efficiency.

- (ii) A weight-based parallel structure is proposed to improve the throughput of the Booth multiplier while maintaining high area and energy efficiency.
- (iii) An optimization is applied on the arithmetic circuits to transform the required write operations to shift operations for performance and energy efficiency.

The rest of the chapter is organized as follows. Section 4.2 presents the pipelined multiplier based on racetrack memory. Section 4.3 discusses the design of the racetrack memory based Booth multiplier with weight-based parallel architecture. In Section 4.4, experimental results are presented and analyzed. Section 4.5 concludes this chapter.

4.2 Pipelined Booth Multiplier Based on Racetrack Memory

Multiplier is a basic arithmetic operator and a large number of multipliers have been explored. Among the various multipliers, Booth multiplier is adopted because it is one of the most efficient multiplication algorithms for binary digit multiplications. Booth multiplier can be typically classified based on the radix such as radix-2, radix-4 and radix-8. While radix-4 stands out because of the hardware implementation efficiency and hence it is thoroughly explored in this work. In radix-4 Booth algorithm, we consider the bits in blocks of three, such that each block overlaps the previous block by one bit. Before grouping the block, a zero is added to the LSB of the multiplier. If there are not enough bits to obtain a MSB of the last block, we sign extend the multiplier by one bit.

Figure 4.1: A 8-bit x 8-bit Booth multiplication with radix-4

For example, if we have a multiplier and a multiplicand which are “01101011” and “00110100” respectively, then we can get four blocks, which are “110”, “101”, “101” and “011”. According to the bits in the four blocks, denoted by Y_{2i-1} , Y_{2i} and Y_{2i+1} , we can get four partial products by the corresponding re-coding rule shown in Table 4.1. In our example case, the four partial products are multiplication of multiplicand with “-1”, “-1”, “-1” and “2” consecutively. Then the sum up of the four partial products gives the multiplication result. The operation of the Booth multiplication is illustrated in Fig. 4.1. The procedure of the Booth multiplier can be classified into two steps: partial products generation and partial products addition. First, we present our implementation of the partial products generation.

Table 4.1: Radix-4 based encoding and decoding regarding to multiplicand

Y_{2i-1}	Y_{2i}	Y_{2i+1}	Partial Product
0	0	0	0* Multiplicand
0	0	1	1* Multiplicand
0	1	0	1* Multiplicand
0	1	1	2* Multiplicand
1	0	0	-2* Multiplicand
1	0	1	-1* Multiplicand
1	1	0	-1* Multiplicand
1	1	1	0* Multiplicand

4.2.1 Partial Products Generation

For the performance concern, we implement partial products generation in parallel. Since we choose radix-4 Booth algorithm, we need to implement radix-4 based Booth encoder and decoder. Table 4.1 shows the encoding and decoding regarding to different block inputs.

As shown in the table, Y_{2i-1} , Y_{2i} and Y_{2i+1} are digits of the input blocks of the multiplier. “0*” means the partial product equals to zero multiplying the multiplicand. It is the same for “1*”, “2*”, “-1*” and “-2*” respectively. First, we need to decode the block inputs to decide which partial products is needed for the multiplication. Then control signals are computed in the encoding step according to the following equations Eq. 4.1-Eq. 4.5 to control the generation of corresponding partial products.

$$zero = Y_{2i-1} \cdot Y_{2i} \cdot Y_{2i+1} + \overline{Y_{2i-1}} \cdot \overline{Y_{2i}} \cdot \overline{Y_{2i+1}} \quad (\text{Eq. 4.1})$$

$$one = \overline{Y_{2i-1}} \cdot \overline{Y_{2i}} \cdot Y_{2i+1} + \overline{Y_{2i-1}} \cdot Y_{2i} \cdot \overline{Y_{2i+1}} \quad (\text{Eq. 4.2})$$

$$two = \overline{Y_{2i-1}} \cdot Y_{2i} \cdot Y_{2i+1} \quad (\text{Eq. 4.3})$$

$$ne_two = Y_{2i-1} \cdot \overline{Y_{2i}} \cdot \overline{Y_{2i+1}} \quad (\text{Eq. 4.4})$$

$$ne_one = Y_{2i-1} \cdot \overline{Y_{2i}} \cdot Y_{2i+1} + Y_{2i-1} \cdot Y_{2i} \cdot \overline{Y_{2i+1}} \quad (\text{Eq. 4.5})$$

Note that these logic functions are implemented by CMOS logic. Since we need to generate the partial products in parallel, the data stored in the racetrack memory need to be re-organized in a certain fashion to enable the parallel operation.

Fig. 4.2 shows the data organization of the multiplier and the multiplicand. Although our design has great scalability and can be applied to 64-bit multiplier, for simplicity, we use 8-bit data as an example to illustrate the data organization and the data flow in the multiplier. As shown in Fig. 4.2, the multiplicand X is stored in the memory stripe in series, while the multiplier Y is stored in separate memory stripes. This data organization ensures that the bits of multiplier can be accessed concurrently, so that the partial products can be generated in parallel. As we can see from Fig. 4.2, if the multiplier has 8-bit, then there are four partial products to be generated based on the four 3-bit groups in the multiplier. According to different partial products, there are different transformations to be applied.

For radix-4 Booth algorithm, there are five kinds of transformations which are *remain*, *negation*, *left-shifting*, *plusing-one*, and *setting-to-zero*. Among the five transformations, *remain* would not cause any change, and can be ignored. *Negation* can be realized by applying a 2-to-2 MUX controlled by control signals in front of writing circuits, which is shown in Fig. 4.2. The selecting signal of the MUX can be the result of OR function of ne_one and ne_two , which means either signal is valid, negation would be applied to the multiplicand to generate the required partial products. *Left-shifting* and *setting-to-zero* can be realized by controlling the shifting circuit, which is shared with the racetrack memory itself. Since the initial state of the racetrack memory is zero, *setting-to-zero* means doing nothing. *Plusing-one* can be realized by setting the initial C_i to “1” when conducting the addition operation.

Figure 4.2: Data organization for generation of partial products

4.2.2 Addition of Partial Productions

After obtaining all the required partial products, we need to sum them together. In order to exploit the inherent advantages of racetrack memory, we pipeline the addition operation. Since the racetrack memory itself can be used as the stage register, the pipelined addition can be very deep, which is very efficient for data intensive applications.

Fig. 4.3 shows the pipelined addition based on racetrack memory. For simplicity, we take 8-bit multiplication as an example, which means the final result has the length of 16 bits. For the ease of illustration, we use a single stripe in the figure to represent the stripe set of the corresponding operand. For 8-bit multiplication, there are four partial products, which should be added together. Instead of feeding the four partial products to four different stripe sets, we feed them to two

Figure 4.3: The pipelined addition based on racetrack memory

stripes sets with multiple access ports. We build the proposed adder right next to (or a few units away from) the access ports. We feed two partial products to the left adder and the other two partial products to the right adder respectively. Then we use the adder in the middle to sum up the results generated by the left and right adders, which is shown with blue arrows in Fig. 4.3. The final result is written into a stripe used to store the result of the multiplication, which is demonstrated by the green arrow in the figure. With such a structure, we can implement the multiplier with much less resource. As an example, in the figure, we show how the four generated partial products in Fig. 4.1 are feed into the proposed pipelined Booth multiplier. With cases having longer bit length, this structure can save more resource in terms of racetrack memory units.

4.3 Booth Multiplier with Weight-based Parallel Architecture

In the previous section, we propose a pipelined Booth multiplier with high area and energy efficiency. However, the throughput of the proposed Booth multiplier is limited due to the architecture of the multiplier. In order to enhance the throughput, in this section we develop a weight-based in-memory Booth multiplier.

4.3.1 Weight-based Parallel Architecture

In this section, we use the Booth multiplier proposed in the previous section as the baseline design. The baseline Booth multiplier design has two major problems. Firstly, the throughput of the multiplier is limited due to its architecture. With the baseline architecture, although the partial products are generated in parallel, for an individual partial product, it is generated bit by bit. The addition of the partial products is pipelined taking the generated partial products as input. Therefore, the number of applied adders is limited, which limits the throughput. For $n - bit \times n - bit$ multiplication, the number of applied adders is $(n/2) - 1$. For example, for $32 - bit \times 32 - bit$ multiplication, the number of applied adders is 15, which limits the throughput of the baseline Booth multiplier. Secondly, although the number of write operations required by the adders and multiplier is minimized, there are still a large number of write operations required in the multiplier design, which consumes considerable amount of energy.

In order to address the first problem, a weight-based parallel architecture is proposed to improve the throughput. In traditional CMOS multiplier design, in order to realize the addition of partial products in parallel, the addition is usually realized by an array of adders, which is fast at the cost of large area overhead and

energy consumption. However, for arithmetic circuits in in-memory computing, large area and energy consumption are not favoured. Weight-based parallel architecture generates the digits in an individual partial product in parallel and add them based on the weight of each digit. Unlike adder array architecture in conventional CMOS-based design, our proposed weight-based parallel architecture improves the throughput of the design while maintaining high area and energy efficiency. With weight-based parallel architecture, $2n - 2$ adders are utilized for n bit X n bit Booth multiplication.

For the second problem, in order to maintain the high energy efficiency of our proposed Booth multiplier when we enhance its throughput, a novel write optimization that transforms write operations to shift operations are proposed to further improve the energy efficiency.

4.3.1.1 Parallel digits generation for the partial products

In baseline Booth multiplier, although the partial products are generated in parallel, for an individual partial product, it is generated bit by bit. In weight-based parallel architecture, addition of the partial products needs to be further parallelized, which requires further parallelization of the partial products generation. In order to realize parallel partial products generation required by the weight-based parallel architecture, the data organization is adjusted. As shown in Fig. 4.4, the multiplicand X is stored in the memory stripe in parallel, while the multiplier Y is stored in series. The multiplier Y is processed three bits a time and the following bits are shifted to the read port accordingly. This data organization ensures that the bits of multiplicand can be accessed concurrently, so that each bit in partial products can be generated in parallel.

Figure 4.4: Data organization for weight-based generation of partial products

4.3.1.2 Weight based parallel addition of the partial products

The optimized architecture of the addition of the partial products is shown in Figure. 4.5. As we state above, the traditional adder array based addition architecture incurs large area overhead and energy consumption, which is not favoured in in-memory design. For area and energy efficiency, we present an architecture that causes less area overhead and energy consumption while achieving high performance. We use an 8-bit X 8-bit multiplication as an example to illustrate the optimized architecture. As mentioned in previous section, for 8-bit X 8-bit multiplication, there are four 16-bit partial products, which are denoted by P1, P2, P3, and P4. Binary digits in the four partial products can be grouped into 16 sets, which contain data with same weight. For example, *set 1* contains data with weight 2^0 , and *set 2* contains data with weight 2^1 . For digits in each set, there is an adder equipped to add them up.

The architecture for 8-bit X 8-bit radix-4 Booth multiplication is shown in

fig. 4.5. As shown in the figure, there are 14 adders in total including 13 full adders and a half adder. For data set with weight 2^3 to weight 2^{15} , bits in each set are fed into a full adder. In these full adders, output signal *sum* is fed into one input of itself and the bits in the corresponding set are fed into the other one. The *carry-out* signal of the full adder with lower weight is fed into the *carry-in* input of the full adder with higher weight, which is shown in the figure. For data set with weight 2^2 , the bits are fed into a half adder because there is no carry signal delivered from previous data set. The bits in data set 2^0 and 2^1 are sent to output directly as the final result because all bits in sets with weight 2^0 and 2^1 are zero except bits in partial product P1, thus, there is no need to perform addition. With such an architecture, the addition of the partial products can be pipelined with enough parallelism.

Figure 4.5: Weight-based parallel addition of the partial products

4.3.2 Write Optimization

As shown in Table 3.5, a write operation consumes 20 times energy as a shift operation. Therefore, if the write operations could be transformed to shift operations, the amount of energy consumed would be reduced considerably. In racetrack memory system design, the write operation is inevitable, because the value in the nanowire needs to be stored. However, in racetrack memory based logic design, the storage unit is used to perform the logic computation, which allows the possibility to transform the write operations to shift operations.

Figure 4.6 shows our proposed method to transform the write operation to the shift operation. In previous design, the data is written into a stripe and shifted to the MTJ to perform computation. However, in our proposed structure, each input MTJ is equipped with a stripe, on which there are “1”, “0” stored in advance. When a “1” signal needs to be written and shifted into the input MTJ, a “1” would be shifted into the input MTJ through the equipped stripe controlled by the shifting circuits. It is the same when a “0” signal needs to be written into the input MTJ. With such a structure, the energy-consuming write operation is transformed into the energy-efficient shift operation at the cost of area overhead. Since write operations have the largest delay, this optimization would also benefit the overall delay. The trade-off between the area overhead and energy and performance efficiency is illustrated in the experimental results section.

4.4 Experimental Results

In this section, we present the experimental results of our proposed designs in terms of area overhead, energy consumption, and throughput. Different design trade-offs are explored by comparing Booth multipliers with different radix base, architectures, and input length.

Figure 4.6: Transformation of write operations to shift operations

4.4.1 Experiment Setup

A CMOS 45 nm design kit [89] and a validated model of perpendicular magnetic anisotropy (PMA) racetrack memory based on CoFeB/MgO structure [129] have been used to perform SPICE simulations for the proposed half and full adders. The main parameters of this PMA racetrack memory model are described in the Table 3.5. We adopt a circuit level racetrack memory model from [128] and the non-volatile memory modelling tool NVSim [130] to estimate the performance, power consumption, and area overhead for our proposed Booth multipliers.

4.4.2 Adders with Write Operation

Table 4.2 shows the results of the proposed half and full adders with and without write optimization. As shown in the table, with write optimization, all the related write operations of the adder are transformed into shift operations. In this sense, the MFA with write optimization eliminates almost all the energy consumed com-

Table 4.2: Comparison of the number of required write operations for different adders

	Delay	Energy	Write operation
CMOS FA [25]	100ps	15fJ	NA
Previous MFA [25]	180ps	7.6fJ	16
Proposed MFA w/o write-opt	240ps	19fJ	7
Proposed MHA w/o write-opt	153ps	16.1fJ	4
Proposed MFA w/ write-opt	240ps	19fJ	0
Proposed MHA w/ write-opt	153ps	16.1fJ	0

pared with MFAs without write optimization. Note that the area overhead is not shown in this table. The area overhead is discussed in the following section.

4.4.3 Multipliers

In this subsection, the experimental results of the proposed Booth multipliers with different radix base, architectures, and input length are presented.

4.4.3.1 Throughput

Fig. 4.7 shows the throughput of proposed Booth multiplier with different architectures, radix base, and input length. The effect of the write optimization for the throughput is also shown in the figure. From the figure, we make the following observations.

First, with weight-based architecture, the throughput of the Booth multipliers is much larger than the ones with baseline architecture. Without write optimization, for radix-2 base, the throughput of Booth multiplier with weight-based parallel architecture is about 2x of the original architecture for same input length. This is because the applied adders in weight-based parallel architecture is about

Figure 4.7: Throughput of Booth multipliers with different radix base, architectures, and input length

two times of the baseline architecture due to the weight-based parallel addition of the partial products. For radix-4 base, the throughput of Booth multiplier with weight-based parallel architecture is about 4x of the one with baseline architecture for same input length. Because with radix-4 base, the number of partial products is reduced to half, which leads to the reduction of the number of applied adders to half in the baseline architecture. However, for the weight-based parallel architecture, the number of applied adders remains the same for different radix base, which leads to four times throughput compared with the baseline architecture.

Second, the write optimization plays a significant role in the energy consumption. With write optimization, for the same architecture, radix base, and input length, the throughput of the multiplier is increased dramatically. This is because the most time-consuming operation in the pipelined multiplier is the write operation, which requires $5ns$, while the shift operation and the logic processing are

much faster than the write operation. Therefore, once write operations are transformed to shift operations, the delay of the multiplier is reduced dramatically, which improves the throughput of the multiplier considerably.

Third, the throughput of Booth multipliers with baseline architecture is fixed for different radix base and input length, while the throughput of Booth multipliers with weight-based parallel architecture is fixed for different input length but increases with different radix base. This is due to the scheme of the addition of partial products in baseline architecture, which ensures the throughput of Booth multiplier with lower radix base and larger input length at the cost of more applied adders and energy consumption compared with the ones with higher radix base and shorter input length. For weight-based parallel architecture, radix-4 based Booth multiplier has about two times the throughput of the radix-2 based one, due to the half number of partial products and unaltered number of applied adders.

4.4.3.2 Area overhead

Area overhead of proposed Booth multipliers with different architectures, radix base, and input length are shown in Fig. 4.8. According to the figure, we make the following observations. First, the weight-based parallel architecture incurs more area overhead than the baseline architecture. For example, as shown in the figure, with radix-4 base and input length as 32-bits, area overhead of Booth multiplier with weight based parallel architecture is 1.28x as that with baseline architecture. Due to the weight based addition of partial products in weight based parallel architecture, the number of applied adders is larger than that of baseline architecture, which leads to the larger area overhead. In the same example, with radix-4 base and input length as 32-bits, there are 15 adders applied in the baseline architecture, while there are 61 adders applied in the weight based parallel

Figure 4.8: Area overhead of Booth multipliers with different radix base, architectures, and input length

architecture. Thus, the number of applied adders in the weight based parallel architecture is nearly four times as that of original architecture. However, the area overhead increment is not that much. This is because the area overhead of partial products generation circuits and other peripheral circuits makes up a high proportion in the overall area overhead. In this example, the area overhead of adders in the baseline architecture is only about 10% of the total area overhead. While in the weight based parallel architecture, the area overhead of adders is about 31% of the total area overhead.

Second, higher radix base of the Booth multipliers causes more area overhead. As shown in the figure, for baseline architecture, with fixed input length, the area overhead of radix-4 based Booth multiplier is larger than the radix-2 one. Let us use 32-bit input length as an example, the area overhead of radix-4 based Booth multiplier is 1.24x as the radix-2 based one. Although the number of applied adders for radix-4 based Booth multiplier is less than that of radix-2 based

one, the partial products generation circuits for radix-4 base is much more complicated than that for radix-2 base, which incurs much larger area overhead. For Booth multipliers with weight based parallel architecture, the situation is similar. Taking 32-bit input length as an example, the area overhead of radix-4 based Booth multiplier is 1.29x as the radix-2 based one. With same number of applied adders, more complicated partial products generation circuits make the area overhead of radix-4 based Booth multiplier larger than the radix-2 based one.

Third, the area overhead increases with the increase of input length. It is reasonable because with the increase of input length, the number of required adders and partial products generation circuits increases, which leads to the increase of area overhead. With identical architecture, radix base, and input length, Booth multipliers with write optimization has slightly larger area overhead than the ones without write optimization due to the peripheral circuits of the write optimization.

4.4.3.3 energy consumption

Fig. 4.9 shows the energy per bit of the proposed Booth multipliers with different architectures, radix base, and input length. We make the following observations. First, in addition to throughput, write optimization contributes dramatically to the energy efficiency. As shown in the figure, the energy consumption of the Booth multiplier with write optimization is negligible compared with that without write optimization. This is because the write operation is the most energy-consuming operation which consumes 1 pJ per operation as shown in Table 3.5. While energy consumption of other operations and logic processing are at fJ level.

Second, without write optimization, the number of write operations dominates the energy consumption of the multipliers. From the figure, we can see that without write optimization, for same input length and radix base, the energy per

Figure 4.9: Energy of Booth multipliers with different architecture and configurations

bit of the Booth multiplier with baseline architecture is almost the same as the one with the weight based parallel architecture. The number of applied adders in the multiplier with the weight based parallel architecture is about two times that with the baseline architecture, which means the number of write operations also have this “two times” correspondence. Therefore, the energy consumed in the multiplier with the weight based parallel architecture ought to be two times the one with the baseline architecture. However, the throughput of the multiplier with the weight based parallel architecture is about two times the baseline one, which leads to similar energy per bit for the two multipliers with different architectures.

Third, without write optimization, for same architecture and input length, the energy per bit of radix-4 based Booth multiplier is about a half of the radix-2 based one. There are two cases: (1) for radix-4 based Booth multiplier with the baseline architecture, it has about half number of applied adders as the radix-2 based one, which leads to the half number of write operations resulting the half

energy per bit; (2) for the radix-4 based Booth multiplier with the weight based parallel architecture, it has same number of applied adders as the radix-2 based one. However, in this case, the throughput of the radix-4 based multiplier is about double the radix-2 based one, which results in the half the energy per bit. With write optimization, for the same input length, the radix-4 based Booth multiplier is more efficient than the radix-2 based one in terms of energy per bit due to the same reason.

4.5 Summary

In this chapter, with the half and full adder as building blocks, we develop in-memory Booth multipliers with different radix base and different input bit length. A pipelined Booth multiplier is proposed by exploiting the inherent sequential access mechanism of racetrack memory, which achieves high area and energy efficiency. In order to enhance the throughput, we develop a weight-based in-memory Booth multiplier. With weight-based parallel architecture, the throughput of the Booth multipliers can be doubled or even quadrupled compared with the baseline design.

In order to maintain the high energy efficiency, we propose an optimization that can transform the energy-consuming write operations to shift operations. By transferring write operations to shift operations, the energy efficiency of the arithmetic circuits can be greatly improved.

Chapter 5

Two-stage Modular Multiplier Based on Racetrack Memory

Asymmetric cryptography algorithms such as RSA are widely used in applications such as blockchain technology and cloud computing to ensure the security and privacy of data. However, the encryption and decryption operations of asymmetric cryptography algorithms involve many computation-intensive multiplications, which require high memory bandwidth and involve large performance and resource overheads. Emerging non-volatile memory technologies such as racetrack memory are regarded to be promising for all levels of memory hierarchy to reduce the area and power overhead due to their high data density and nearly zero leakage. In this chapter, we propose an efficient racetrack memory based in-memory design to accelerate the modular multiplication for asymmetric cryptography algorithms. A novel two-stage scalable modular multiplication algorithm is proposed to significantly improve the delay. An efficient architecture is further developed to reduce the number of required adders by half. Experimental results show that our proposed scheme improves the energy efficiency by 45.9%, the area efficiency by 93.6% and achieves 8x the throughput per area compared with the state-of-the-art CMOS-based implementation.

5.1 Introduction

The area and power overhead of CMOS technology based memory is growing dramatically because of the increasing leakage power along with the shrinking technology node. In order to reduce the area and power overhead while meeting the requirement of large data storage, emerging non-volatile memories have been proposed to implement all levels of the memory hierarchy due to their near-zero leakage power and small area. Among various non-volatile memories, the racetrack memory is regarded as a good candidate to implement all levels of the memory system [13], even to perform logic computations [98] because of its advantages of ultra-high data density, low power, and high speed. Particularly, the high energy and area efficiency of racetrack memory make it an ideal platform for main data storage.

In order to protect the security of stored data, asymmetric cryptography algorithms such as the RSA algorithm and elliptic curve cryptography are widely used in various applications such as blockchain technology and cloud computing. However, transferring data between memory and processor simply for data encryption and decryption would incur significant overhead. In order to alleviate the memory bandwidth limitation and I/O pressure, in-memory hardware accelerated encryption and decryption have been proposed. However, there is still another unsolved challenge. The massive modular multiplications involved in the encryption and decryption operations of the asymmetric cryptography algorithms require a large amount of energy and area in CMOS technology.

To address the challenges, we propose an efficient racetrack memory based in-memory computing scheme to accelerate the modular multiplication. It satisfies the ultra-high data density requirement and realizes energy-efficient in-memory encryption/decryption. To the best of our knowledge, this is the first work that

focuses on racetrack memory based modular multiplication. We first propose a novel two-stage scalable modular multiplication algorithm to double the performance of the baseline algorithm. Second, in order to further reduce the overhead, we present an efficient architecture of the modular multiplier to reduce the number of required adders by half. Our proposed scheme improves the energy efficiency by 45.9%, the area efficiency by 93.6% and achieves 8x of throughput per area compared with the state-of-the-art CMOS-based implementation.

The key contributions of this work can be summarized as follows:

- (i) The first work focusing on racetrack memory based modular multiplier design to accelerate the in-memory data encryption and decryption for asymmetric cryptography.
- (ii) A novel two-stage scalable modular multiplication algorithm presented to double the performance of the baseline modular multiplication.
- (iii) An efficient architecture proposed to reduce the number of required adders by half, achieving remarkable reduction of the area and energy overhead.

The rest of the chapter is organized as follows. In Section 5.2, basic design of word-based Montgomery multiplication is introduced. Section 5.3 presents the two-stage scalable modular multiplication algorithm. An efficient architecture based on racetrack memory is detailed in Section 5.4. In Section 5.5, experimental results are presented and analyzed. Section 5.6 concludes this chapter.

5.2 Word-based Montgomery Multiplication

Tenca and Koc et al. proposed a word-based parallel implementation with hardware pipelining, in which the operations in different iterations of the outer loop

Algorithm 1 Word-Based Radix-2 Montgomery Multiplication Algorithm

Input: X, Y, M **Output:** $Z = X \cdot Y \cdot 2^{-n} \bmod M$
$$\begin{aligned} X &= \{x_{n-1}, x_{n-2}, \dots, x_0\} & Y &= \{y_{n-1}, y_{n-2}, \dots, y_0\} &= \\ &\{Y^{e-1}, \dots, Y^1, Y^0\} \\ M &= \{m_{n-1}, m_{n-2}, \dots, m_0\} = \{M^{e-1}, \dots, M^1, M^0\} \end{aligned}$$

```
1:  $Z \leftarrow 0$ 
2: for  $i = 0; i < n; i++$  do
3:    $(carry|Z^0) = Z^0 + x_i \cdot Y^0$ 
4:    $q = Z_0^0$ 
5:    $(carry|Z^0) = (carry|Z^0) + q \cdot M^0$ 
6:   for  $j = 1; j < e; j++$  do
7:      $(carry|Z^j) = x_i \cdot Y^j + Z^j + carry$ 
8:      $q = Z_0^j$ 
9:      $(carry|Z^j) = (carry|Z^j) + q \cdot M^j$ 
10:     $Z^{j-1} = (Z_0^j|Z_{w-1}^{j-1}, \dots, Z_1^{j-1})$ 
11:   end for
12:    $Z^{e-1} = (carry|Z_{w-1}^{e-1}, \dots, Z_1^{e-1})$ 
13: end for
```

are executed in parallel to reduce the latency at the cost of more hardware overhead [32], as shown in Alg. 1. The algorithm addresses the scalability problem by processing multiplicand Y and modulus M word by word while processing the multiplicand X bit by bit. For a modulus M with n -bit precision, there are $e = \lceil n + 1/w \rceil$ words with w bits.

In Alg. 1, the word index is marked with superscript while bit index is marked with subscript and the concatenation of two binary digits is expressed with “|”. With each x bit, the algorithm uses two loops to process Y and M word by word. Each word of Z is obtained by additions followed by a shift operation. The word-based processing is efficient, however, the data dependency between the two successive iterations in the algorithm limits the throughput. The reason is that after computation of word Z^j and Z^{j+1} in one iteration, the least significant bit (LSB) of Z^{j+1} would be shifted to be the most significant bit (MSB) of Z^j . With parallel processing, the data dependency of the algorithm is shown in Fig. 5.1.

Figure 5.1: The data dependency graph for Alg. 1

Here, one processing element is used to process one x bit, and generate one Z word in one clock cycle. All bits in x are processed in parallel. Due to data dependency, the MSB of Z^j word computed using x_i cannot be obtained until the Z^{j+1} word computed using x_{i-1} is obtained, which leads to an initial delay of two clock cycles for each processing element. With the precision of the multiplicand equal to n , the total delay can be as long as $2(n - 1) + e$ clock cycles.

5.3 Proposed Modular Multiplication Algorithm

Since the Montgomery multiplication involves addition and shift operations, and the shift operation is an intrinsic operation of racetrack memory, it is very efficient to realize Montgomery multiplication based on racetrack memory. However, the two cycles delay and a large amount of processing resources required are challenging for efficient in-memory implementation. In this section, we propose a

novel two-stage scalable modular multiplication algorithm, which doubles the performance of the baseline word-based modular multiplication.

5.3.1 Independence of MSB Operation

The disadvantage of the previous algorithm lies in the fact that there exists data dependency between two words of the intermediate results because of the right shift operation. We observe that the dependency is only related to the MSB of word Z^j . Moreover, in multi-bit additions, the operation on MSB does not affect the operation on the lower bits. Since the words of Z are generated by additions, we can process the lower bits, the vast majority, in advance without obtaining the MSB.

For example, given two 8-bit binary numbers $a = "11111111"$ and $b = "10001001"$, the sum of the two numbers is $c = "110001000"$. If the MSB of b is not known in the first place, we can assume that the MSB of b is "0". We denote the assumed b by b' , which equals "00001001". We now calculate the intermediate sum of a and b , namely the sum of a and b' , which is denoted by c' . By adding a and b' , c' is obtained as "100001000" without knowing the MSB of b . When the MSB of b is determined, if the MSB is "1", then we only need to add "1" in the eighth bit of c' , resulting in $c = "110001000"$. Otherwise, the result is unaltered as $c = "100001000"$. From this example, we can see that the last addition is very efficient and independent of the bits whose weight are lower than 2^7 , which leads to very small overhead. Therefore, in our algorithm, the MSB of Z is assumed as 0 in the first step, and the final result is updated when the MSB of Z is determined.

As the foundation of our proposed two-stage algorithm, the correctness of the independence of MSB operation is ensured by lemma 5.1.

Algorithm 2 Optimized Word Based Radix-2 Montgomery Multiplication Algorithm

Input: X, Y, M
Output: $Z = X \cdot Y \cdot 2^{-n} \bmod M$

```

1:  $Z \leftarrow 0$ 
2: for  $i = 0; i < n; i++$  do
3:    $(carry|Z^0) := Z^0 + x_i \cdot Y^0$ 
4:    $q = Z_0^0$ 
5:   if  $i > 0$  then
6:      $CM = Z_0^1$ 
7:   end if
8:    $(carry|Z^0) = (carry|Z^0) + q \cdot M^0 + CM \cdot 2^{w-1}$ 
9:    $Z^0 = (0|Z_{w-1}^{(0)}, \dots, Z_1^{(0)})$ 
10:  for  $j = 1; j < (e - 1); j++$  do
11:     $(carry|Z^j) = Z^j + x_i \cdot Y^j + carry$ 
12:     $q = Z_0^j$ 
13:    if  $i > 0$  then
14:       $CM = Z_0^{(j+1)}$ 
15:    end if
16:     $(carry|Z^j) = (carry|Z^j) + q \cdot M^j + CM \cdot 2^{w-1}$ 
17:     $Z^j = (0|Z_{w-1}^j, \dots, Z_1^j)$ 
18:  end for
19:   $(carry|Z^{e-1}) = Z^{e-1} + x_i \cdot Y^{e-1} + carry$ 
20:   $q = Z_0^{e-1}$ 
21:   $(carry|Z^{e-1}) = (carry|Z^{e-1}) + q \cdot M^{e-1}$ 
22:   $Z^{e-1} = (carry|Z_{w-1}^{e-1}, \dots, Z_1^{e-1})$ 
23: end for

```

Lemma 5.1 *Given two binary numbers a and b doing addition, the addition on bits with higher weight does not affect the bits with lower weight in the final addition result.*

Proof: Assume the number a has more bits than b and is expressed as “ $a_{n-1} a_{n-2} \dots a_1 a_0$ ” while the binary number b is expressed as “ $b_{m-1} b_{m-2} \dots b_1 b_0$ ”. Both n and m are positive integers and $n > m$. For any integer i , where $i \leq m$, in the final result, the bit with weight 2^i is decided by a_i , b_i , and carry-in bit from weight 2^{i-1} . There are four cases: (1) all of the three bits are “1”, then the bit with weight 2^i is “1” and the carry-in bit to weight 2^{i+1} is “1”; (2) two of the three bits are “1”, then the bit with weight 2^i is “0” and the carry-in bit to weight 2^{i+1} is “1”; (3) one of the three bits is “1”, then the bit with weight 2^i is “1” and the carry-in bit to weight 2^{i+1} is “0”; (4) none of the three bits is “1”, then the bit with weight 2^i is “0” and the carry-in bit to weight 2^{i+1} is “0”. None of the four cases affects the bit with weight lower than 2^i . For any integer i , where $m < i \leq n$, the situation is similar. Thus, Lemma 1 holds. We can similarly prove when a has no more bits than b .

5.3.2 Two-stage Algorithm

Based on this observation, we optimize the algorithm for $n > 1$ as shown in Alg. 2, where X , Y , M are expressed in the same manner as in Alg. 1. We divide each processing node in Fig. 5.1 into two stages as shown in Fig. 5.2, where the semi-circles labeled as S1 and S2 represent the stage 1 and stage 2 in each processing node, respectively. In the first stage of processing element i , $(carry|Z^j) = x_i \cdot Y^j + Z^j + carry$ is computed assuming the MSB of Z^j is 0. While in the second stage, $(carry|Z^j) = (carry|Z^j) + q \cdot M^j + CM \cdot 2^{w-1}$ is computed using corrected MSB denoted by CM delivered from processing

Figure 5.2: The data dependency graph for Alg. 2

element $i - 1$. The data dependency of the two-stage algorithm is illustrated in Fig. 5.2. Therefore, with the proposed two-stage algorithm, the two clock cycles initial delay of each processing element in the previous algorithm has been reduced to one stage delay. The total delay of the algorithm could be reduced by nearly half.

First stage (line 11-15): In stage 1 of processing element i , we assume the MSB of Z^j is 0 and start to compute the addition on the lower bits immediately with $x_i \cdot Y^j$ and *carry*. It saves one stage waiting time for each processing element. With the computation in the first stage, Z_0^j in the first stage denoted as q is determined, which would be used in the second stage. We note that since the addition of CM in the second stage does not affect the result related to the lower bits, q obtained from the first stage is correct. Therefore, Z_0^j in the second stage can be obtained ahead of time by q and M_0^j in the first stage. Similarly, in the

first stage of processing element $i - 1$, the LSB of word Z^{j+1} is also computed and then assigned to CM . Then CM , q , and the resulted Z^j are delivered to the second stage of processing element i .

Second stage (line 16-17): In stage 2 of processing element i , with the data from first stage, $(carry|Z^j) = (carry|Z^j) + q \cdot M^j + CM \cdot 2^{w-1}$ can be computed. After the final result of Z^j in the second stage is obtained, a right shift operation is applied on it before it is delivered to the following processing element. The right shift operation is a one-bit shift operation assuming the shift-in MSB is “0”.

5.4 Architecture Design and Optimization

5.4.1 Processing Element Design and Optimization

In order to implement the proposed design in Alg. 2, we present an efficient pipelined processing element, which performs the key operation in two stages. The first stage is to compute $Z^j + x_i \cdot Y^j + carry$ while determining its LSB denoted by q . The second stage is to compute $Z^j + q \cdot M^j + CM \cdot 2^{w-1}$. If simply counting the additions in two stages, it requires four adders. As one processing element is used to process one x bit, it requires a large amount of area and power consumption, if the length of X is large. In order to reduce the overhead, some optimizations have to be applied to the processing element design.

5.4.1.1 Basic Operation

Due to the property of the MTJ in racetrack memory, the XOR function and majority function are very efficient. With these two logic functions, an adder can be built. The circuits that perform the XOR and the majority functions are shown in Fig. 5.4(a) and Fig. 5.4(b), respectively.

For the XOR logic function in the figure, R_1 equals $(R_L + R_H)/2$. We stack the A/B together with an oxide barrier between them to realize the XOR logic function. Previous papers and industrial products show that this structure has the same high/low resistance behavior as the MTJ which has a fixed reference layer [126, 98]. When “ A ” and “ B ” are different, the output is “1” (“0” otherwise), which realizes the XOR function.

For the majority function, R_2 equals $2R_H$, and “ A ”, “ B ” and “ C_{in} ” are shifted into the MTJs with one fixed “0” reference layer. With R_H equals $2.5R_L$ in our model, if the number of “1” signal among the three inputs is larger than the number of “0” signal, the output would be “1” (“0” otherwise). Therefore, it realizes the majority function. Based on these two efficient logic functions, we design our adder, the fundamental processing unit for our design.

5.4.1.2 Adder Design and Optimization

In our design, for a standard full adder, two separate XOR logic gates are used to compute $sum = A \oplus B \oplus C_{in}$. A majority logic gate is used to compute the *carry* logic. With the current design, operations in one processing element require four full adders. However, we observe that the two additions in each stage have three operands, among which there is an operand containing at most one “1” bit. Particularly, for addend $CM \cdot 2^{w-1}$, only its MSB is possible to be “1”. Similarly, only one bit in addend *carry* is possible to be “1”. Note that the *carry* signal has two bits.

Based on this observation, we propose a new adder design to reduce the number of required adders by half. Fig. 5.4(c) shows the architecture of our proposed design. Compared with a standard full adder, our adder has an extra XOR logic gate and an extra input S . This adder can be used to perform operations in both

the stages. When it is used in stage 1, the S input is connected to the *carry* signal and when used in stage 2, the S input is connected to the CM signal. As the racetrack memory shifts the input bit by bit into the MTJ, we use one copy of the proposed adder to perform the pipelined n -bit ripple-carry addition. It means the adder first computes the LSB addition of addend A , B , and S . Then the carry out bit, C_o is written into MTJ as C_{in} of the next bit addition. Depending on the bit value of S , the mechanism of the logic is as follows.

Case 1: If the bit “ S_i ” is “0”, then “ C_{in} ” is the output of the XOR gate. In this case, “ C_{in} ” is used as the carry in signal of the full adder. The behaviour of the full adder is unaltered.

Case 2: If the bit “ S_i ” is “1”, depending on the value of C_{in} , the behavior of the adder changes. If C_{in} is “0”, then “ S_i ” used as the carry in signal of the full adder, which is similar with Case 1. If C_{in} is “1”, it means the four input bits could all be “1”. Since the adder only has two output bits, which are not enough to represent the result, we have to postpone the carry computation to the next clock cycle. For example, if A_i, B_i, C_{in} are “1, 1, 1”, respectively, then in the current clock cycle, the output of the three XOR gates are all “0”. Therefore, Sum is “0”, and C_o is “1”. Then C_o is fed to C_{in} for the next clock, and S_i remains without being shifted to the next bit. The postponement continues until the cycle when C_{in} turns to “0”. With *carry* and CM having independent shifting stripes, the pipeline timing is not affected. Fig. 5.3 shows an example of how the adder operates. In this example, the three addends are “119”, “123”, and “32”, respectively. S' indicates the actual digit in the S port.

Note that such mechanism is only efficient for S with only one “1” bit. With such a logic structure, the number of required full adders is reduced to half at the cost of one XOR logic gate for each adder and one transmission gate for the S_i shift control.

A (119)	0	0	1	1	1	0	1	1	1
B (123)	0	0	1	1	1	1	0	1	1
C _{in}	0	1	1	1	1	1	1	1	0
S (32)	0	0	0	1	0	0	0	0	0
S'	1	1	1	0	0	0	0	0	0
Result (274)	1	0	0	0	1	0	0	1	0

Figure 5.3: An example of how the optimized adder operates

Figure 5.4: The structure of basic logic units and proposed adder

Figure 5.5: Architecture of the proposed modular multiplication

5.4.1.3 Processing Element Design

Fig. 5.5 shows the design of the processing element, which consists of two cascaded adders proposed above. Each adder in the processing element is used to implement one stage in the proposed two-stage modular multiplication. The AND operation shown in the figure is realized by using the bit signal to control the writing of word signal. The two adders in the processing element are pipelined to increase the throughput and hardware utilization.

5.4.2 Architecture of the Proposed Modular Multiplication

Fig. 5.5 shows the architecture of the proposed modular multiplication. The architecture of the multiplication consists of a buffer to store the data fetched from the memory, pipelined PEs, and shift registers for the multiplication. The shift registers are used to shift the data such as Y , M , and X to the corresponding PEs. Note that all the registers and control logic in the architecture are realized by race-track memory based registers and logic, which are very efficient in area overhead and energy consumption compared with CMOS-based designs. According to the

figure, PEs in the architecture are pipelined to parallelize the operations of the outer loop in the proposed algorithm. Since the racetrack memory itself can be used as the stage register, this pipelined structure achieves high efficiency in energy consumption and area overhead.

5.5 Experimental Results

In this section, we present the experimental results of the proposed design in area overhead, energy consumption, and throughput. We compare our design with the state-of-the-art Montgomery multiplication designs which are implemented using CMOS-based ASIC [30, 31].

5.5.1 Experiment Setup

To evaluate our proposed scheme, the following experiment platform has been set up. At the device level, a CMOS 45 nm design kit[89] and a validated model of perpendicular magnetic anisotropy (PMA) racetrack memory based on CoFeB/MgO structure[129] have been used to perform SPICE simulation for the evaluation of the proposed adder. Table 5.1 shows the key parameters of the device model. We adopt a circuit level racetrack memory model from [128] to estimate the performance, power consumption, and area overhead for the buffer and registers. We modify the non-volatile memory modelling tool NVSim [130] together with the adopted racetrack memory model [128] to evaluate the design of the modular multiplier. NVSim is a circuit-level estimator which evaluates the circuit area, performance, and power by estimating properties of cells and their peripheral circuitry. It is widely used in academic studies [59, 63, 62].

Table 5.1: Main parameters in the RM model

Parameter	Description	Default value
W_{RT}	Width of racetrack	$1F$
G_{RT}	Gap distance between two racetracks	$1F$
L_D	Length of the domain in a racetrack	$2F$
L_{RT}	Length of racetrack	$128F$
T_{RT}	Thickness of racetrack	$6nm$
p	PMA racetrack nanowire resistivity	$4.8 \times 10^7 \Omega m$
$TMR(0)$	TMR with $0 V_{bias}$	150%
W_{EN}	Write energy	$1pJ$
$J_{nucleation}$	Critical current density for write	$5.7 \times 10^6 A/cm^2$
W_{DE}	Write latency	$5ns$
S_{EN}	Shift energy	$0.051pJ$
J_{shift}	Critical current density for shift	$6.2 \times 10^7 A/cm^2$
S_{DE}	Shift latency	$500ps$

Table 5.2: Results of the basic logic

	XOR	Maj	Proposed pipelined adder	Two cascaded CMOS adders
Delay	$100ps$	$110ps$	$212.9 ps$	$200 ps$
Energy	$2.88fJ$	$2.43fJ$	$13.47fJ$	$30 fJ$
Area	$0.679um^2$	$0.852um^2$	$2.889um^2$	$20.08um^2$

5.5.2 Evaluation of Basic Logic

Table 5.2 shows the performance in terms of delay, energy, and area of the XOR logic, the majority logic (Maj), and the presented pipelined adder. The results of the two cascaded CMOS adders which can fulfill the same function as our proposed pipelined adder are also listed in the table for comparison. As shown in the table, the MTJ based basic logic are slower than the CMOS-based logic, however, they are very efficient in terms of energy consumption, and area overhead. The proposed pipelined adder has an energy efficiency of $13.47fJ$ per operation which is 45% of the two cascaded CMOS adders. For the area overhead, the proposed pipelined adder is 14.4% of the two cascaded CMOS adders.

Figure 5.6: Performance comparison of different schemes

5.5.3 Evaluation of Modular Multiplication

In our experiment, the memory is set as RAM with a capacity of 1MB and block size of 512 bits. The result data of the CMOS-based design are extracted from the reported results in [30] and [31] with corresponding technology scaling. Comparison of the three designs is shown in Fig. 5.6. Note that the comparison is conducted with the precision n set as 2048 bits and the word length w set as 16 bits.

Energy per bit: The results of energy per bit of the three schemes are shown in Fig. 5.6(a). Our proposed scheme has an energy efficiency of $68.2pJ/b$ which improves the energy efficiency by 75.2% and 45.8% compared with the two CMOS-based ASIC designs in [30] and [31], respectively. It is because that non-volatile memory itself can be used as the stage and shift registers. Furthermore,

the logic gates based on MTJ are energy efficient. With shifting racetrack memory based register, our design saves a considerable amount of energy compared with designs using CMOS-based ones.

Area: Fig. 5.6(b) shows the area of our proposed design and the two CMOS-based designs for comparison. Our proposed scheme improves the area efficiency by 92% and 93.6% compared with the two CMOS-based ASIC designs in [30] and [31], respectively. Our proposed design has an area of $5231 \mu m^2$, which is 1.7% of the racetrack memory area. The low percentage of computing logic in memory is critical such that the in-memory logic does not affect the data density of the memory.

Throughput: The throughput results are shown in Fig. 5.6(c). The throughput of our proposed design is 50.1% of the CMOS ones [30, 31] because that the speed of racetrack memory based logic is slower than the CMOS-based one. Besides, the shift delay of racetrack memory is up to 500 ps per bit.

However, as shown in Fig. 5.6(d), the throughput per area of our proposed design is 6.4x and 8x compared with the two CMOS-based ASIC designs in [30] and [31], respectively.

Word length exploration: As a scalable modular multiplication scheme, one of the multiplicands is processed word by word. Hence the length of the word is a critical parameter that affects the performance of the scheme. The results of energy per bit and throughput with fixed area budget are shown in Fig. 5.7 and Fig. 5.8. The results are shown for two precision n settings, which are 1024 bits and 2048 bits respectively. For each precision, we compare the results with word length, w set as 8 bits, 16 bits, 32 bits, and 64 bits.

As shown in the figure, with the increase of word length, the energy per bit increases considerably, while the throughput increases by a small amount. In

Figure 5.7: Energy per bit with different word length

our design, the length of shifting register is set as the same length as word length. With the fixed area budget, the transistor drive strength is fixed. For the energy per bit, when the length of racetrack memory based register increases, the number of domains on the racetrack increases, which leads to the increase of shifting energy.

For the throughput, when the word length decreases, the number of iterations in inner loop increases, which leads to more processing time on the carry logic, resulting in a decrease of throughput. Therefore, with a fixed area budget, the design becomes more energy efficient if the word length is small. However, if the word length is too small, the energy consumption for processing carry logic becomes large, which leads to worse energy efficiency.

Precision exploration: Fig. 5.7 and Fig. 5.8 also show the energy per bit and throughput of the design with two precision n settings, which are 1024 bits and 2048 bits, respectively. With a fixed area budget, as the precision increases, the energy per bit increases while the throughput decreases. For applications with increasing precision, if the throughput needs to be guaranteed, more computing resources are required, which leads to increasing area overhead.

Figure 5.8: Throughput with different word length

5.6 Summary

In this chapter, we propose an efficient two-stage modular multiplier based on racetrack memory. It achieves the ultra-high data density requirement while realizing efficient in-memory modular multiplication. Our proposed scheme improves the energy efficiency by 45.9%, the area efficiency by 93.6% and achieves 8x of throughput per area compared with the state-of-the-art CMOS-based implementation.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this dissertation, we explore the racetrack memory based logic design for in-memory computing systems. To solve the challenges posed by the demanding area and power requirements, we made the following contributions:

- Racetrack memory based in-memory half and full adders are proposed. Our proposed adders significantly reduce the area overhead and energy consumption compared with CMOS technology-only adder designs and the previous magnetic adder designs. Experiment results show that our proposed full adder can save 66% area and 56.3% energy compared with the previous state-of-the-art magnetic full adder.
- A pipelined Booth multiplier is proposed by exploiting the inherent sequential access mechanism of racetrack memory, which achieves high area and energy efficiency. Weight-based parallel architecture is proposed to increase the throughput of the magnetic Booth multiplier. An optimization is proposed to transform the required energy-consuming write operations to shift operations to ensure the high energy efficiency. Experimental results

show that in combination with the write optimization, the racetrack memory based Booth multiplier with weight-based parallel architecture achieves $3.1Pj/\text{bit}$ for 16-bit x 16-bit multiplication size.

- A racetrack memory based two-stage modular multiplier is proposed to accelerate the computation-intensive modular multiplication for specific applications while minimizing the required area overhead and energy consumption. Experimental results show that our design improves the energy efficiency by 45.9%, the area efficiency by 93.6% and achieves 8x of throughput per area compared with the state-of-the-art CMOS-based implementation.

Overall, in this dissertation, we propose high efficiency in-memory computing logic for racetrack memory based in-memory computing system. For the limited area budget problem, we improve the area efficiency dramatically to realize complex computing in memory. For the throughput and bandwidth gap, we bridge it by increasing the throughput per area of the in-memory computing logic. For the stringent energy constraint, we alleviate it by minimizing the energy consumption of the in-memory computing logic.

The proposed computing logic includes arithmetic computing cells for general purpose computing and specialized computing logic for modular multiplication. The in-memory computing infrastructure with specialized computing logic is attractive for many important applications such as graph computing and so on. The application of racetrack memory based in-memory infrastructure to these applications needs further exploration.

6.2 Future Work

Racetrack memory based in-memory computing has shown great potential for many pivotal applications, such as security in big data era. In this section, we discuss some new directions for our future work.

6.2.1 Racetrack Memory Based Reconfigurable Modular Multiplier for Both RSA and ECC

RSA and ECC are two of the most popular public key cryptosystems in modern computer security domain. However, these two cryptosystems require modular multiplications over different Galois Fields (GF). The two-stage modular multiplier we proposed aims at the modular multiplication over GF (p), which is the core operation in the encryption and decryption operations in RSA. Regarding ECC, modular multiplications over GF (p) and GF (2^m) are required in its point multiplication [131]. Thus, in order to accelerate both RSA and ECC with the same accelerator, a hardware modular multiplier that can accelerate modular multiplication over both GF (p) and GF (2^m) is required. In the future, we are going to design a reconfigurable dual-field modular multiplier. With the reconfigurability, it can be configured for either RSA-based application or ECC-based application.

6.2.2 In-memory CGRA Design for Homomorphic Encryption with Montgomery Multiplication

A fully homomorphic encryption (FHE) scheme is an encryption scheme that would enable arbitrary computation on encrypted data. Its delicate feature opens up many new possibilities in the cloud-centric, data-driven world. It is regarded as a critical cryptographic tool in building a secure and reliable cloud computing environment. The first FHE scheme proposed by Gentry *et al.* [132] attracts

interests of many researchers. However, existing software FHE implementations remain impractical due to their limited performance. Therefore, an efficient hardware accelerator is required.

Coarse-grained reconfigurable architecture (CGRAs) consist of an array of a large number of function units (FUs) interconnected by a mesh style network [133]. The FUs in the CGRA can execute common or domain-specific word-level operations, including addition, subtraction, and multiplication. In contrast to fine-grained reconfigurable architecture, CGRAs have short reconfiguration times, low delay characteristics, and low power consumption. Thus, CGRAs are suitable for the acceleration of homomorphic encryption scheme, which has regular computing pattern. With our proposed reconfigurable dual-field modular multiplier and adders as the FUs, in-memory CGRAs have potential to accelerate the homomorphic encryption efficiently.

6.2.3 In-memory Accelerator for Encrypted Database

With the development of cloud computing, confidentiality of user data is becoming an important concern [134]. However, online applications are vulnerable to theft of sensitive information because adversaries can exploit software bugs to gain access to private data. In addition to the adversaries, curious or malicious administrators may capture and leak data too. Encrypted database, such as CryptDB [135], is a kind of database management system that can provide practicality and provable confidentiality in the face of these attacks.

The confidentiality of the encrypted database is ensured by various kinds of encryption schemes, including symmetric encryption schemes, such as AES, and asymmetric encryption schemes, such as RSA and ECC. The capability of race-track memory based logic for asymmetric schemes and symmetric schemes is

shown in our work and previous works. In addition, the in-memory database system is explored and analyzed by many researchers due to the increasing memory capacity [136]. Therefore, we are interested in exploring the possibility to accelerate the encrypted database with our proposed racetrack memory based in-memory computing design.

Author's Publications

- **Tao Luo**, Wei Zhang, Bingsheng He, and Douglas L. Maskell, “A Novel Two-stage Modular Multiplier Based on Racetrack Memory for Asymmetric Cryptography”, *2017 International Conference On Computer Aided Design (ICCAD)*, 2017, pp.276-282.
- **Tao Luo**, Wei Zhang, Bingsheng He, and Douglas L. Maskell, “A race-track memory based in-memory modular multiplication for cryptography application”, *In work-in-progress session of Design Automation Conference (DAC)*, 2017
- **Tao Luo**, Hao Liang, Wei Zhang, Bingsheng He, and Douglas Maskell, “A Hybrid Logic Block Architecture in FPGA for Holistic Efficiency”, *Circuits and Systems II, IEEE Transactions on (TCAS-II)* 64.1 (2017): 71-75.
- **Tao Luo**, Wei Zhang, Bingsheng He, and Douglas L. Maskell, “A race-track memory based in-memory booth multiplier for cryptography application”, *Design Automation Conference (ASP-DAC), 2016 IEEE 21st Asia and South Pacific*, 2016, pp.286-291.
- Bin Zhou, Wei Zhang, ambipillai Srikanthan, Jason Teo Kian Jin, Vivek Chaturvedi, and **Tao Luo**, “Cost- efficient Acceleration of Hardware Trojan Detection through Fan-out Cone Analysis and Weighted Random Pattern

Technique”, *Computer-Aided Design of Integrated Circuits and Systems (TCAD), IEEE Transactions on* 35.5 (2016): 792-805.

- Hao Liang, Yi-Chung Chen, **Tao Luo**, Wei Zhang, Hai Li, and Bingsheng He, “Hierarchical library based power estimator for versatile FPGAs”, *Embedded Multicore/Many-core Systems-on-Chip (MCSoc), 2015 IEEE 9th International Symposium on*, 2015, pp.25-32.

References

- [1] R. R. Schaller, “Moore’s law: past, present and future,” *IEEE spectrum*, vol. 34, no. 6, pp. 52–59, 1997.
- [2] J. T. Pawlowski, “Hybrid memory cube (HMC),” in *Hot Chips 23 Symposium (HCS), 2011 IEEE*, pp. 1–24, IEEE, 2011.
- [3] J. Kim and Y. Kim, “HBM: Memory solution for bandwidth-hungry processors,” in *Hot Chips 26 Symposium (HCS), 2014 IEEE*, pp. 1–24, IEEE, 2014.
- [4] H. Zhang, G. Chen, B. C. Ooi, K.-L. Tan, and M. Zhang, “In-memory big data management and processing: A survey,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 7, pp. 1920–1948, 2015.
- [5] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, “A scalable processing-in-memory accelerator for parallel graph processing,” in *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on*, pp. 105–117, IEEE, 2015.
- [6] L. Wilson, “International technology roadmap for semiconductors (ITRS),” *Semiconductor Industry Association*, 2013.
- [7] H. Zheng, J. Lin, Z. Zhang, E. Gorbato, H. David, and Z. Zhu, “Mini-rank: Adaptive DRAM architecture for improving memory power efficiency,” in *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, pp. 210–221, IEEE Computer Society, 2008.

- [8] T. Kuroda, “CMOS design challenges to power wall,” in *Microprocesses and Nanotechnology Conference, 2001 International*, pp. 6–7, IEEE, 2001.
- [9] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, “Scalable high performance main memory system using phase-change memory technology,” *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 24–33, 2009.
- [10] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, “The missing memristor found,” *nature*, vol. 453, no. 7191, pp. 80–83, 2008.
- [11] R. Fackenthal, M. Kitagawa, W. Otsuka, K. Prall, D. Mills, K. Tsutsui, J. Javanifard, K. Tedrow, T. Tsushima, Y. Shibahara, *et al.*, “19.7 a 16Gb ReRAM with 200MB/s write and 1GB/s read in 27nm technology,” in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pp. 338–339, IEEE, 2014.
- [12] C. Lin, S. Kang, Y. Wang, K. Lee, X. Zhu, W. Chen, X. Li, W. Hsu, Y. Kao, M. Liu, *et al.*, “45nm low power CMOS logic compatible embedded STT MRAM utilizing a reverse-connection 1T/1MTJ cell,” in *Electron Devices Meeting (IEDM), 2009 IEEE International*, pp. 1–4, IEEE, 2009.
- [13] S. S. Parkin, M. Hayashi, and L. Thomas, “Magnetic domain-wall race-track memory,” *Science*, vol. 320, no. 5873, pp. 190–194, 2008.
- [14] R. Venkatesan, V. Kozhikkottu, C. Augustine, A. Raychowdhury, K. Roy, and A. Raghunathan, “TapeCache: a high density, energy efficient cache based on domain wall memory,” in *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, pp. 185–190, ACM, 2012.
- [15] O. J. Bedrij, “Carry-select adder,” *IRE Transactions on Electronic Computers*, no. 3, pp. 340–346, 1962.

- [16] T.-Y. Chang and M.-J. Hsiao, "Carry-select adder using single ripple-carry adder," *Electronics letters*, vol. 34, no. 22, pp. 2101–2103, 1998.
- [17] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on electronic Computers*, no. 1, pp. 14–17, 1964.
- [18] K.-J. Cho, K.-C. Lee, J.-G. Chung, and K. K. Parhi, "Design of low-error fixed-width modified Booth multiplier," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 5, pp. 522–531, 2004.
- [19] H. Meng, J. Wang, and J.-P. Wang, "A spintronics full adder for magnetic CPU," *IEEE Electron Device Letters*, vol. 26, no. 6, pp. 360–362, 2005.
- [20] S. Matsunaga, J. Hayakawa, S. Ikeda, K. Miura, H. Hasegawa, T. Endoh, H. Ohno, and T. Hanyu, "Fabrication of a nonvolatile full adder based on logic-in-memory architecture using magnetic tunnel junctions," *Applied Physics Express*, vol. 1, no. 9, p. 091301, 2008.
- [21] E. Deng, Y. Zhang, J.-O. Klein, D. Ravelsona, C. Chappert, and W. Zhao, "Low power magnetic full-adder based on spin transfer torque MRAM," *IEEE transactions on magnetics*, vol. 49, no. 9, pp. 4982–4987, 2013.
- [22] Y. Gang, W. Zhao, J.-O. Klein, C. Chappert, and P. Mazoyer, "A high-reliability, low-power magnetic full adder," *IEEE Transactions on Magnetism*, vol. 47, no. 11, pp. 4611–4616, 2011.
- [23] 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*, vol. 57, no. 5, pp. 1023–1028, 2010.
- [24] E. Deng, Y. Zhang, W. Kang, B. Dieny, J.-O. Klein, G. Prenat, and W. Zhao, "Synchronous 8-bit non-volatile full-adder based on spin transfer torque magnetic tunnel junction," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 7, pp. 1757–1765, 2015.

- [25] H.-P. Trinh, W. Zhao, J.-O. Klein, Y. Zhang, D. Ravelsona, and C. Chappert, "Magnetic adder based on racetrack memory," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 6, pp. 1469–1477, 2013.
- [26] M. Zheng and A. Albicki, "Low power and high speed multiplication design through mixed number representations," in *Computer Design: VLSI in Computers and Processors, 1995. ICCD'95. Proceedings., 1995 IEEE International Conference on*, pp. 566–570, IEEE, 1995.
- [27] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.
- [28] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.
- [29] P. Giorgi, L. Imbert, and T. Izard, "Parallel modular multiplication on multi-core processors," in *Computer Arithmetic (ARITH), 2013 21st IEEE Symposium on*, pp. 135–142, IEEE, 2013.
- [30] M.-D. Shieh and W.-C. Lin, "Word-based montgomery modular multiplication algorithm for low-latency scalable architectures," *IEEE transactions on computers*, vol. 59, no. 8, pp. 1145–1151, 2010.
- [31] W.-C. Lin, J.-H. Ye, and M.-D. Shieh, "Scalable montgomery modular multiplication architecture with low-latency and low-memory bandwidth requirement," *IEEE Transactions on Computers*, vol. 63, no. 2, pp. 475–483, 2014.
- [32] A. F. Tenca and Ç. K. Koç, "A scalable architecture for modular multiplication based on Montgomery's algorithm," *IEEE Transactions on computers*, vol. 52, no. 9, pp. 1215–1221, 2003.

- [33] H.-S. 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*, vol. 98, no. 12, pp. 2201–2227, 2010.
- [34] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *ACM SIGARCH Computer Architecture News*, vol. 37, pp. 2–13, ACM, 2009.
- [35] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *ACM SIGARCH computer architecture news*, vol. 37, pp. 14–23, ACM, 2009.
- [36] G. Servalli, "A 45nm generation phase change memory technology," in *Electron Devices Meeting (IEDM), 2009 IEEE International*, pp. 1–4, IEEE, 2009.
- [37] G. W. Burr, M. J. Brightsky, A. Sebastian, H.-Y. Cheng, J.-Y. Wu, S. Kim, N. E. Sosa, N. Papandreou, H.-L. Lung, H. Pozidis, *et al.*, "Recent progress in phase-change memory technology," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 2, pp. 146–162, 2016.
- [38] F. Pellizzer, A. Pirovano, F. Ottogalli, M. Magistretti, M. Scaravaggi, P. Zuliani, M. Tosi, A. Benvenuti, P. Besana, S. Cadeo, *et al.*, "Novel/spl mu/trench phase-change memory cell for embedded and stand-alone non-volatile memory applications," in *VLSI Technology, 2004. Digest of Technical Papers. 2004 Symposium on*, pp. 18–19, IEEE, 2004.
- [39] M. Qiu, Z. Ming, J. Li, K. Gai, and Z. Zong, "Phase-change memory optimization for green cloud with genetic algorithm," *IEEE Transactions on Computers*, vol. 64, no. 12, pp. 3528–3540, 2015.
- [40] K. Huang, Y. Ha, R. Zhao, A. Kumar, and Y. Lian, "A low active leakage and high reliability phase change memory (PCM) based non-volatile FPGA storage element," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 9, pp. 2605–2613, 2014.

- [41] S. Lee, G. Kim, S. Hong, S. J. Baik, H. Hori, and D.-h. Ahn, “Enhanced cycling endurance in phase change memory via electrical control of switching induced atomic migration,” in *Non-Volatile Memory Technology Symposium (NVMTS), 2014 14th Annual*, pp. 1–3, IEEE, 2014.
- [42] M. Boniardi, A. Redaelli, C. Cupeta, F. Pellizzer, L. Crespi, G. D’Arrigo, A. Lacaita, and G. Servalli, “Optimization metrics for phase change memory (PCM) cell architectures,” in *Electron Devices Meeting (IEDM), 2014 IEEE International*, pp. 29–1, IEEE, 2014.
- [43] S. Raoux, F. Xiong, M. Wuttig, and E. Pop, “Phase change materials and phase change memory,” *MRS bulletin*, vol. 39, no. 8, pp. 703–710, 2014.
- [44] I. Baek, M. Lee, S. Seo, M. Lee, D. Seo, D.-S. Suh, J. Park, S. Park, H. Kim, I. Yoo, *et al.*, “Highly scalable nonvolatile resistive memory using simple binary oxide driven by asymmetric unipolar voltage pulses,” in *Electron Devices Meeting, 2004. IEDM Technical Digest. IEEE International*, pp. 587–590, IEEE, 2004.
- [45] H. Lee, P. Chen, T. Wu, Y. Chen, C. Wang, P. Tzeng, C. Lin, F. Chen, C. Lien, and M.-J. Tsai, “Low power and high speed bipolar switching with a thin reactive Ti buffer layer in robust HfO₂ based RRAM,” in *Electron Devices Meeting, 2008. IEDM 2008. IEEE International*, pp. 1–4, IEEE, 2008.
- [46] Z. Wei, Y. Kanzawa, K. Arita, Y. Katoh, K. Kawai, S. Muraoka, S. Mitani, S. Fujii, K. Katayama, M. Iijima, *et al.*, “Highly reliable TaO_x ReRAM and direct evidence of redox reaction mechanism,” in *Electron Devices Meeting, 2008. IEDM 2008. IEEE International*, pp. 1–4, IEEE, 2008.
- [47] M. Di Ventra and Y. V. Pershin, “On the physical properties of memristive, memcapacitive and meminductive systems,” *Nanotechnology*, vol. 24, no. 25, p. 255201, 2013.

- [48] S. Vongehr and X. Meng, “The missing memristor has not been found,” *Scientific reports*, vol. 5, 2015.
- [49] A. Khvalkovskiy, D. Apalkov, S. Watts, R. Chepulsii, R. Beach, A. Ong, X. Tang, A. Driskill-Smith, W. Butler, P. Visscher, *et al.*, “Basic principles of STT-MRAM cell operation in memory arrays,” *Journal of Physics D: Applied Physics*, vol. 46, no. 7, p. 074001, 2013.
- [50] H.-S. P. Wong and S. Salahuddin, “Memory leads the way to better computing,” *Nature nanotechnology*, vol. 10, no. 3, pp. 191–194, 2015.
- [51] E. Kültürsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, “Evaluating STT-RAM as an energy-efficient main memory alternative,” in *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*, pp. 256–267, IEEE, 2013.
- [52] Z. Sun, X. Bi, H. H. Li, W.-F. Wong, Z.-L. Ong, X. Zhu, and W. Wu, “Multi retention level STT-RAM cache designs with a dynamic refresh scheme,” in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 329–338, ACM, 2011.
- [53] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, “Relaxing non-volatility for fast and energy-efficient STT-RAM caches,” in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, pp. 50–61, IEEE, 2011.
- [54] W. Xu, H. Sun, X. Wang, Y. Chen, and T. Zhang, “Design of last-level on-chip cache using spin-torque transfer RAM (STT RAM),” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 3, pp. 483–493, 2011.
- [55] Q. Li, Y. He, J. Li, L. Shi, Y. Chen, and C. J. Xue, “Compiler-assisted refresh minimization for volatile STT-RAM cache,” *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2169–2181, 2015.

- [56] S. Parkin and S.-H. Yang, “Memory on the racetrack,” *Nature nanotechnology*, vol. 10, no. 3, pp. 195–198, 2015.
- [57] M. Hayashi, L. Thomas, R. Moriya, C. Rettner, and S. S. Parkin, “Current-controlled magnetic domain-wall nanowire shift register,” *Science*, vol. 320, no. 5873, pp. 209–211, 2008.
- [58] R. Tomasello, E. Martinez, R. Zivieri, L. Torres, M. Carpentieri, and G. Finocchio, “A strategy for the design of skyrmion racetrack memories,” *Scientific reports*, vol. 4, 2014.
- [59] Z. Sun, W. Wu, and H. Li, “Cross-layer racetrack memory design for ultra high density and low power consumption,” in *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, pp. 1–6, 2013.
- [60] H. Xu, Y. Li, R. Melhem, and A. K. Jones, “Multilane racetrack caches: Improving efficiency through compression and independent shifting,” in *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*, pp. 417–422, IEEE, 2015.
- [61] G. W. Burr, B. N. Kurdi, J. C. Scott, C. H. Lam, K. Gopalakrishnan, and R. S. Shenoy, “Overview of candidate device technologies for storage-class memory,” *IBM Journal of Research and Development*, vol. 52, no. 4.5, pp. 449–464, 2008.
- [62] C. Zhang, G. Sun, X. Zhang, W. Zhang, W. Zhao, T. Wang, Y. Liang, Y. Liu, Y. Wang, and J. Shu, “Hi-fi playback: Tolerating position errors in shift operations of racetrack memory,” in *ACM SIGARCH Computer Architecture News*, vol. 43, pp. 694–706, ACM, 2015.
- [63] M. Mao, W. Wen, Y. Zhang, Y. Chen, and H. H. Li, “Exploration of GPGPU register file architecture using domain-wall-shift-write based race-track memory,” in *Proceedings of the 51st Annual Design Automation Conference*, pp. 1–6, ACM, 2014.

- [64] R. Venkatesan, V. K. Chippa, C. Augustine, K. Roy, and A. Raghunathan, "Domain-specific many-core computing using spin-based memory," *IEEE Transactions on Nanotechnology*, vol. 13, no. 5, pp. 881–894, 2014.
- [65] R. Venkatesan, V. J. Kozhikkottu, M. Sharad, C. Augustine, A. Raychowdhury, K. Roy, and A. Raghunathan, "Cache design with domain wall memory," *IEEE Transactions on Computers*, vol. 65, no. 4, pp. 1010–1024, 2016.
- [66] R. Venkatesan, S. G. Ramasubramanian, S. Venkataramani, K. Roy, and A. Raghunathan, "Stag: Spintronic-tape architecture for GPGPU cache hierarchies," in *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, pp. 253–264, IEEE, 2014.
- [67] R. Venkatesan, M. Sharad, K. Roy, and A. Raghunathan, "DWM-TAPESTRI-an energy efficient all-spin cache using domain wall shift based writes," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 1825–1830, EDA Consortium, 2013.
- [68] R. Venkatesan, M. Sharad, K. Roy, and A. Raghunathan, "Energy-efficient all-spin cache hierarchy using shift-based writes and multilevel storage," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 12, no. 1, p. 4, 2015.
- [69] M. Moeng, H. Xu, R. Melhem, and A. K. Jones, "ContextPreRF: Enhancing the performance and energy of GPUs with nonuniform register access," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 1, pp. 343–347, 2016.
- [70] Y. Zhang, W. Zhao, J.-O. Klein, D. Ravelsona, and C. Chappert, "Ultra-high density content addressable memory based on current induced domain wall motion in magnetic track," *IEEE Transactions on Magnetics*, vol. 48, no. 11, pp. 3219–3222, 2012.

- [71] E. Atoofian, “Reducing shift penalty in domain wall memory through register locality,” in *Proceedings of the 2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pp. 177–186, IEEE Press, 2015.
- [72] E. Atoofian and A. Saghir, “Shift-aware racetrack memory,” in *Computer Design (ICCD), 2015 33rd IEEE International Conference on*, pp. 427–430, IEEE, 2015.
- [73] X. Zhang, L. Zhao, Y. Zhang, and J. Yang, “Exploit common source-line to construct energy efficient domain wall memory based caches,” in *Computer Design (ICCD), 2015 33rd IEEE International Conference on*, pp. 157–163, IEEE, 2015.
- [74] H. Mao, C. Zhang, G. Sun, and J. Shu, “Exploring data placement in racetrack memory based scratchpad memory,” in *Non-Volatile Memory System and Applications Symposium (NVMSA), 2015 IEEE*, pp. 1–5, IEEE, 2015.
- [75] E. Park, S. Yoo, S. Lee, and H. Li, “Accelerating graph computation with racetrack memory and pointer-assisted graph representation,” in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pp. 1–4, IEEE, 2014.
- [76] S. Mittal, “A survey of techniques for architecting and managing GPU register file,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 16–28, 2017.
- [77] Y. Liang and S. Wang, “Performance-centric optimization for racetrack memory based register file on GPUs,” *Journal of Computer Science and Technology*, vol. 31, no. 1, pp. 36–49, 2016.
- [78] J. Chung, K. Ramclan, J. Park, and S. Ghosh, “Exploiting serial access and asymmetric read/write of domain wall memory for area and energy-efficient digital signal processor design,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 1, pp. 91–102, 2016.

- [79] D. G. Elliott, M. Stumm, W. M. Snelgrove, C. Cojocar, and R. McKenzie, "Computational RAM: Implementing processors in memory," *IEEE Design & Test of Computers*, vol. 16, no. 1, pp. 32–41, 1999.
- [80] M. Hall, P. Kogge, J. Koller, P. Diniz, J. Chame, J. Draper, J. LaCoss, J. Granacki, J. Brockman, A. Srivastava, *et al.*, "Mapping irregular applications to DIVA, a PIM-based data-intensive architecture," in *Proceedings of the 1999 ACM/IEEE conference on Supercomputing*, p. 57, ACM, 1999.
- [81] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz, "Smart memories: A modular reconfigurable architecture," in *Computer Architecture, 2000. Proceedings of the 27th International Symposium on*, pp. 161–171, IEEE, 2000.
- [82] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, "A case for intelligent RAM," *IEEE micro*, vol. 17, no. 2, pp. 34–44, 1997.
- [83] S. Das, A. Fan, K.-N. Chen, C. S. Tan, N. Checka, and R. Reif, "Technology, performance, and computer-aided design of three-dimensional integrated circuits," in *Proceedings of the 2004 international symposium on Physical design*, pp. 108–115, ACM, 2004.
- [84] M. Motoyoshi, "Through-silicon via (TSV)," *Proceedings of the IEEE*, vol. 97, no. 1, pp. 43–48, 2009.
- [85] A. M. Shams, T. K. Darwish, and M. A. Bayoumi, "Performance analysis of low-power 1-bit CMOS full adder cells," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 10, no. 1, pp. 20–29, 2002.
- [86] N. Zhuang and H. Wu, "A new design of the CMOS full adder," *IEEE journal of solid-state circuits*, vol. 27, no. 5, pp. 840–844, 1992.
- [87] M. J. Flynn, "Very high-speed computing systems," *Proceedings of the IEEE*, vol. 54, no. 12, pp. 1901–1909, 1966.

- [88] J.-Y. Kang and J.-L. Gaudiot, "A simple high-speed multiplier design," *IEEE Transactions on computers*, vol. 55, no. 10, pp. 1253–1258, 2006.
- [89] S. Nangate, "California (2008). 45nm open cell library," URL; <http://www.nangate.com>, 2008.
- [90] J. Sulistyo, J. Perry, and D. Ha, "Developing standard cells for TSMC 0.25 um technology under MOSIS DEEP rules," *Department of Electrical and Computer Engineering, Virginia Tech, Technical Report VISC-2003-01*, 2003.
- [91] C.-Y. Lee, C.-S. Yang, B. K. Meher, P. K. Meher, and J.-S. Pan, "Low-complexity digit-serial and scalable SPB/GPB multipliers over large binary extension fields using (b, 2)-way Karatsuba decomposition," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 11, pp. 3115–3124, 2014.
- [92] A. Mandal and R. Syal, "Tripartite Modular Multiplication using Toom-Cook Multiplication," *International Journal of Advanced Research in Computer Science and Electronics Engineering (IJARCSEE)*, vol. 1, no. 2, pp. pp–100, 2012.
- [93] S.-K. Chen, C.-W. Liu, T.-Y. Wu, and A.-C. Tsai, "Design and implementation of high-speed and energy-efficient variable-latency speculating Booth multiplier (VLSBM)," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 10, pp. 2631–2643, 2013.
- [94] N. S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power," *computer*, vol. 36, no. 12, pp. 68–75, 2003.
- [95] D. Chabi, W. Zhao, E. Deng, Y. Zhang, N. B. Romdhane, J.-O. Klein, and C. Chappert, "Ultra low power magnetic flip-flop based on check-pointing/power gating and self-enable mechanisms," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 6, pp. 1755–1765, 2014.

- [96] S. Onkaraiah, M. Reyboz, F. Clermidy, J.-M. Portal, M. Bocquet, C. Muller, C. Anghel, A. Amara, *et al.*, “Bipolar ReRAM based non-volatile flip-flops for low-power architectures,” in *New Circuits and Systems Conference (NEWCAS), 2012 IEEE 10th International*, pp. 417–420, IEEE, 2012.
- [97] W. Zhao, E. Belhaire, C. Chappert, F. Jacquet, and P. Mazoyer, “New non-volatile logic based on spin-MTJ,” *physica status solidi (a)*, vol. 205, no. 6, pp. 1373–1377, 2008.
- [98] Y. Wang, H. Yu, D. Sylvester, and P. Kong, “Energy efficient in-memory AES encryption based on nonvolatile domain-wall nanowire,” in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pp. 1–4, IEEE, 2014.
- [99] V. S. Miller, “Use of elliptic curves in cryptography,” in *Conference on the Theory and Application of Cryptographic Techniques*, pp. 417–426, Springer, 1985.
- [100] N. Koblitz, “Elliptic curve cryptosystems,” *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.
- [101] U. Somani, K. Lakhani, and M. Mundra, “Implementing digital signature with RSA encryption algorithm to enhance the Data Security of cloud in Cloud Computing,” in *Parallel Distributed and Grid Computing (PDGC), 2010 1st International Conference on*, pp. 211–216, IEEE, 2010.
- [102] J. Blömer, M. Otto, and J.-P. Seifert, “A new CRT-RSA algorithm secure against bellcore attacks,” in *Proceedings of the 10th ACM conference on Computer and communications security*, pp. 311–320, ACM, 2003.
- [103] B. Kaliski, “What key size should be used?,” 2016.
- [104] C. K. Koc, T. Acar, and B. S. Kaliski, “Analyzing and comparing montgomery multiplication algorithms,” *IEEE micro*, vol. 16, no. 3, pp. 26–33, 1996.

- [105] B. S. Kaliski, "The Montgomery inverse and its applications," *IEEE transactions on computers*, vol. 44, no. 8, pp. 1064–1065, 1995.
- [106] S. E. Eldridge and C. D. Walter, "Hardware implementation of Montgomery's modular multiplication algorithm," *IEEE transactions on Computers*, vol. 42, no. 6, pp. 693–699, 1993.
- [107] H. Orup, "Simplifying quotient determination in high-radix modular multiplication," in *Computer Arithmetic, 1995., Proceedings of the 12th Symposium on*, pp. 193–199, IEEE, 1995.
- [108] A. Bernal and A. Guyot, "Design of a modular multiplier based on montgomery's algorithm," in *13th Conference on Design of Circuits and Integrated Systems*, pp. 680–685, 1998.
- [109] C.-C. Yang, T.-S. Chang, and C.-W. Jen, "A new RSA cryptosystem hardware design based on Montgomery's algorithm," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 45, no. 7, pp. 908–913, 1998.
- [110] C.-Y. Su, S.-A. Hwang, P.-S. Chen, and C.-W. Wu, "An improved Montgomery's algorithm for high-speed RSA public-key cryptosystem," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 2, pp. 280–284, 1999.
- [111] C. McIvor, M. McLoone, and J. V. McCanny, "Modified Montgomery modular multiplication and RSA exponentiation techniques," *IEE Proceedings-Computers and Digital Techniques*, vol. 151, no. 6, pp. 402–408, 2004.
- [112] M.-D. Shieh, J.-H. Chen, H.-H. Wu, and W.-C. Lin, "A new modular exponentiation architecture for efficient design of RSA cryptosystem," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 9, pp. 1151–1161, 2008.

- [113] P. Kornerup, “High-radix modular multiplication for cryptosystems,” in *Computer Arithmetic, 1993. Proceedings., 11th Symposium on*, pp. 277–283, IEEE, 1993.
- [114] C. D. Walter, “Space/Time trade-offs for higher radix modular multiplication using repeated addition,” *IEEE Transactions on Computers*, vol. 46, no. 2, pp. 139–141, 1997.
- [115] A. Royo, J. Moran, and J. C. Lopez, “Design and implementation of a coprocessor for cryptography applications,” in *Proceedings of the 1997 European conference on Design and Test*, p. 213, IEEE Computer Society, 1997.
- [116] T. Blum and C. Paar, “Montgomery modular exponentiation on reconfigurable hardware,” in *Computer Arithmetic, 1999. Proceedings. 14th IEEE Symposium on*, pp. 70–77, IEEE, 1999.
- [117] C. D. Walter, “Systolic modular multiplication,” *IEEE transactions on computers*, vol. 42, no. 3, pp. 376–378, 1993.
- [118] A. F. Tenca and C. K. Koç, “A scalable architecture for Montgomery multiplication,” in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 94–108, Springer, 1999.
- [119] A. F. Tenca, G. Todorov, and C. K. Koç, “High-radix design of a scalable modular multiplier,” in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 185–201, Springer, 2001.
- [120] D. Harris, R. Krishnamurthy, M. Anders, S. Mathew, and S. Hsu, “An improved unified scalable radix-2 Montgomery multiplier,” in *Computer Arithmetic, 2005. ARITH-17 2005. 17th IEEE Symposium on*, pp. 172–178, IEEE, 2005.
- [121] M. Huang, K. Gaj, and T. El-Ghazawi, “New hardware architectures for Montgomery modular multiplication algorithm,” *IEEE Transactions on computers*, vol. 60, no. 7, pp. 923–936, 2011.

- [122] A. Miyamoto, N. Homma, T. Aoki, and A. Satoh, "Systematic design of RSA processors based on high-radix Montgomery multipliers," *IEEE Transactions on very large scale integration (VLSI) Systems*, vol. 19, no. 7, pp. 1136–1146, 2011.
- [123] "A scalable and unified multiplier architecture for finite fields GF (p) and GF (2m), author=Savaš, Erkey and Tenca, Alexandre F and Koç, Cetin K, booktitle=International Workshop on Cryptographic Hardware and Embedded Systems, pages=277–292, year=2000, organization=Springer,"
- [124] A. Satoh and K. Takano, "A scalable dual-field elliptic curve cryptographic processor," *IEEE Transactions on Computers*, vol. 52, no. 4, pp. 449–460, 2003.
- [125] J.-H. Chen, M.-D. Shieh, and W.-C. Lin, "A high-performance unified-field reconfigurable cryptographic processor," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 18, no. 8, pp. 1145–1158, 2010.
- [126] Q. Stainer, L. Lombard, K. Mackay, R. Sousa, I. Prejbeanu, and B. Dineny, "MRAM with soft reference layer: In-stack combination of memory and logic functions," in *Memory Workshop (IMW), 2013 5th IEEE International*, pp. 84–87, IEEE, 2013.
- [127] Y. Wang, H. Yu, L. Ni, G.-B. Huang, M. Yan, C. Weng, W. Yang, and J. Zhao, "An energy-efficient nonvolatile in-memory computing architecture for extreme learning machine by domain-wall nanowire devices," *IEEE Transactions on Nanotechnology*, vol. 14, no. 6, pp. 998–1012, 2015.
- [128] C. Zhang, G. Sun, W. Zhang, F. Mi, H. Li, and W. Zhao, "Quantitative modeling of racetrack memory, a tradeoff among area, performance, and power," in *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*, pp. 100–105, IEEE, 2015.

- [129] Y. Zhang, W. Zhao, D. Ravelosona, J.-O. Klein, J.-V. Kim, and C. Chappert, “Perpendicular-magnetic-anisotropy CoFeB racetrack memory,” *Journal of Applied Physics*, vol. 111, no. 9, p. 093925, 2012.
- [130] X. Dong, C. Xu, Y. Xie, and N. P. 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*, vol. 31, no. 7, pp. 994–1007, 2012.
- [131] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006.
- [132] C. Gentry *et al.*, “Fully homomorphic encryption using ideal lattices,” in *STOC*, vol. 9, pp. 169–178, 2009.
- [133] K. Compton and S. Hauck, “Reconfigurable computing: a survey of systems and software,” *ACM Computing Surveys (csuR)*, vol. 34, no. 2, pp. 171–210, 2002.
- [134] I. M. Khalil, A. Khreishah, and M. Azeem, “Cloud computing security: a survey,” *Computers*, vol. 3, no. 1, pp. 1–35, 2014.
- [135] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, “CryptDB: protecting confidentiality with encrypted query processing,” in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pp. 85–100, ACM, 2011.
- [136] P.-Å. Larson and J. Levandoski, “Modern main-memory database systems,” *Proceedings of the VLDB Endowment*, vol. 9, no. 13, pp. 1609–1610, 2016.