A Processing-In-Memory Implementation of SHA-3 Using a Voltage-Gated Spin Hall-Effect Driven MTJ-based Crossbar

Chengmo Yang
University of Delaware
chengmo@udel.edu

Zeyu Chen
University of Delaware
zeyuchen@udel.edu

ABSTRACT

Processing-In-Memory (PIM), which implements logic operations within memory cells, opens up a new direction on organizing data and computation. Leveraging resistive or magnetic characteristics of nonvolatile memory (NVM) devices, platforms such as PLiM and ReVAMP have been proposed. This paper presents a PIM implementation of SHA-3, a state-of-the-art secure hash algorithm using a Voltage-Gated Spin Hall-Effect (SHE) Driven magnetic tunnel junction (MTJ) based crossbar, which is able to achieve a complete set of Boolean operations. The work includes the design of the crossbar circuit, the instruction set, and both unpipelined and pipelined implementations of SHA-3. Experimental results show that the proposed SHE MTJ-based implementation is able to achieve 2.16X higher throughput than a state-of-the-art Resistive RAM based SHA-3 implementation. Further throughput improvement can be achieved with multiple message hash (MMH) pipelining.

CCS CONCEPTS

• **Hardware → Non-volatile memory**; **Hardware accelerators**; **Emerging architectures**.

KEYWORDS

Processing-in-Memory; Non-volatile memories

ACM Reference Format:
Chengmo Yang and Zeyu Chen. 2019. A Processing-In-Memory Implementation of SHA-3 Using a Voltage-Gated Spin Hall-Effect Driven MTJ-based Crossbar. In *Great Lakes Symposium on VLSI 2019 (GLSVLSI '19), May 9–11, 2019, Tysons Corner, VA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3299874.3317972

1 INTRODUCTION

Today's applications entail more data storage and computing power than ever before. However, when executed on a traditional von Neumann architecture that separates computation and data storage, the time and energy taken to commute the inputs from the memory to the processor and then write the results back to the memory is hundreds or even thousands of times longer than the computation operation itself. The existing solution to this well known "memory wall" problem relies on multiple levels of caches. Unfortunately, the limited memory bandwidth still imposes a crucial limitation for data-intensive applications.

One promising solution to overcome the bottleneck of von Neumann architecture is *processing-in-memory (PIM)*, which integrates logic and memory on the same device. PIM can be achieved with emerging non-volatile memory (NVM) devices, which leverage physical characteristics such as resistance or magnetic field to store data and are able to perform logic operations as well. A number of NVM-based PIM designs have been proposed. Among them, IMPLY [1], majority function [2], and Boolean [3] are three representative designs based on Resistive RAM (RRAM) memory.

Compared with RRAM, spintronic magnetic memory (MRAM) shows its advantages of longer endurance and lower power consumption. Different MRAM-based logic implementations have been shown in [4–6]. Recently, Kang *et al.* reported a realization of stateful reconfigurable logic functions via a single three-terminal magnetic tunnel junction (MTJ) device, which exploits a novel voltage-gated Spin Hall Effect (VG-SHE) driven magnetization-switching mechanism [7]. This device is a promising candidate for future PIM architecture implementations. To demonstrate its potential, this work presents an implementation of SHA-3 [8], the latest secure hash algorithm standard released by NIST, on a VG-SHE driven MTJ crossbar. A secure hash function is an important component of data confidentiality used for data authentication. The basic operations of SHA-3 are XOR, rotation, and AND, which can be efficiently implemented on a VG-SHE driven MTJ crossbar. This work develops a comprehensive PIM implementation of SHA-3, including the design of the crossbar circuit, the instruction set, an unpipelined single message hash (SMH) implementation, and a pipelined multiple message hash (MMH) implementation. Compared with the most related previous work, i.e., a SHA-3 implementation on a RRAM crossbar [9], the proposed SMH and MMH designs improve throughput by 216% and 851%, respectively.

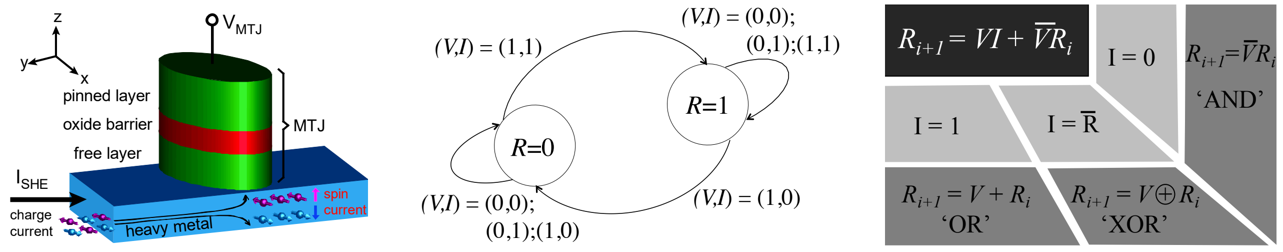
The rest of this paper is organized as follows. Section 2 introduces the fundamental structure of VG-SHE driven MTJ and its reconfigurable logic implementation, followed by a brief description of SHA-3. Section 3 presents the implementation details of SHA-3 on a VG-SHE driven MTJ crossbar. Section 4 experimentally compares the proposed MTJ crossbar with state-of-the-art SHA-3 implementations. At the end, conclusions are given in Section 5.

2 PRELIMINARIES

2.1 VG-SHE Driven MTJ Device

The VG-SHE driven MTJ structure was proposed in [7]. Figure 1(a) shows the schematic of this three-terminal device. The voltage

This work is partially supported by NSF Award #1527464.



(a) Device schematic **(b) State transition diagram** **(c) Boolean logic**

Figure 1: Three-terminal VG-SHE driven MTJ device and its stateful reconfigurable logic.

V_{MTJ} and the current I_{SHE} are the two inputs, represented as V and I, respectively. The resistance of the MTJ is the state and output, denoted as R, which can be read out by measuring the tunnel magnetoresistance of the MTJ. The key property of this device is that the current for switching the SHE-driven MTJ can be modulated by a voltage applied across the MTJ via the voltage-controlled magnetic anisotropy (VCMA) mechanism. Further details can be found in [7].

This work makes the following assumptions regarding inputs and output: (1) a high/low resistance state represents logic state and output R=1/0; (2) a positive bias voltage (600 mV) corresponds to input V=1, while a zero bias voltage represents V=0; (3) a positive/negative SHE current $\pm I_{SHE}$ denotes input I=1/0.

Functionality of this three-terminal MTJ device can be summarized by a state machine, shown in Figure 1(b). Once the MTJ is applied with a positive voltage (i.e., V=1), the next state R_{i+1} depends on the polarity of the SHE current (i.e., R_{i+1}=I); If a zero bias voltage is applied (i.e., V=0), the next state will not change (i.e., R_{i+1}=R_i). Combining the two cases, the stateful Boolean logic function can be expressed as:

$$R_{i+1} = VI + \overline{V}R_i \tag{1}$$

Equation (1) indicates that the MTJ can be viewed as a *memory* cell when input V is used as a control signal. On the other hand, when input I is used as a control signal, the MTJ can be viewed as a *logic* cell. As illustrated in Figure 1(c), I can be used to select among AND, OR, and XOR operations between V (or $\overline{V}$) and R. This is the key to the proposed crossbar design.

2.2 VG-SHE Driven MTJ Crossbar

A high density cross-point array built with the VG-SHE driven MTJ device was proposed in [7]. The crossbar structure is composed of multiple strings of MTJ devices located on a heavy metal strip. The wordline (WL) and sourceline (SL) switches are row-wise, while the bitline (BL) switches are column-wise. They together control the memory and logic operations. Status of these switches as well as values of the two inputs V and I are summarized in Table 1 for each memory and logic operation.

Figure 2 illustrates the read and write operations in VG-SHE crossbar, which require one and two cycles, respectively. To perform read (Figure 2(a)), the SL switch of the selected row is activated, a small negative voltage is applied on all the columns, and all the BL switches are activated. Data can be read out via measuring the tunnel resistance state. As shown in Figure 2(a), a 'low/high' on

Table 1: Signal values in memory/logic operations

	WL	**SL**	**BL**	**V**	**I**
Read	OFF	ON	ON	$-V$	
Write	ON	ON	$\overline{D_{DMR}}$	1	-I_{SHE}
			D_{DMR}		+I_{SHE}
AND	ON	ON	D_{DMR}	1	-I_{SHE}
OR	ON	ON	D_{DMR}	1	+I_{SHE}
XOR	OFF	ON	ON	$-V$	
	ON	ON	D_{DMR}	D_{XR}	-I_{SHE}
	ON	ON	$\overline{D_{DMR}}$	D_{XR}	+I_{SHE}

the bitline represents data '1/0'. After passing through the sense amplifier (SA) and comparator, the readout data is stored in the data memory register (DMR).

Write operations take two cycles because currents of different polarities need to be flowed through the heavy metal. This process is shown in Figure 2(b). Both SL and WL switches of the selected row are activated. All the columns are applied with a positive voltage, while the BL switches are controlled by the data in DMR and a control signal S. In step 1, S=1 which connects the columns corresponding to 0's in the DMR. MTJs on those columns are written with 0's, by flowing $-I_{SHE}$ through the heavy metal. In step 2, S=0 and columns corresponding to 1's in the DMR are connected. MTJs on those columns are written with 1's, by flowing $+I_{SHE}$ through the heavy metal.

Logic operations AND and OR are highly similar to the write operation. As shown in Table 1, status of WL and SL switches are the same in these three operations. AND and OR differ from write in that they require only one cycle and the control signal S is always set to 0. Moreover, AND and OR use different values of I. As shown in Figure 1(c), OR is performed when I=1, while AND is performed when I=0. This difference is reflected in the last column of Table 1.

2.3 Introduction to SHA-3

SHA-3 is a subset of the cryptographic primitive family *Keccak* [8]. It is based on sponge construction, which operates on a state of b=r+c bits; r is the *rate* or *blocksize* which comes directly from the message, c is the *capacity* and determines the security level, and b is the state which is $5 \times 5 \times w$ bits long, with w denoting the word size. In the most commonly used case of SHA3-256, w=64, b=1600, r=1088, and c=512.

Sponge construction uses the block permutation function Keccak-f, which consists of 24 rounds. In each round, the message block is operated in 5 steps: θ, ρ, π, χ, and ι, as shown in Algorithm 1. Each

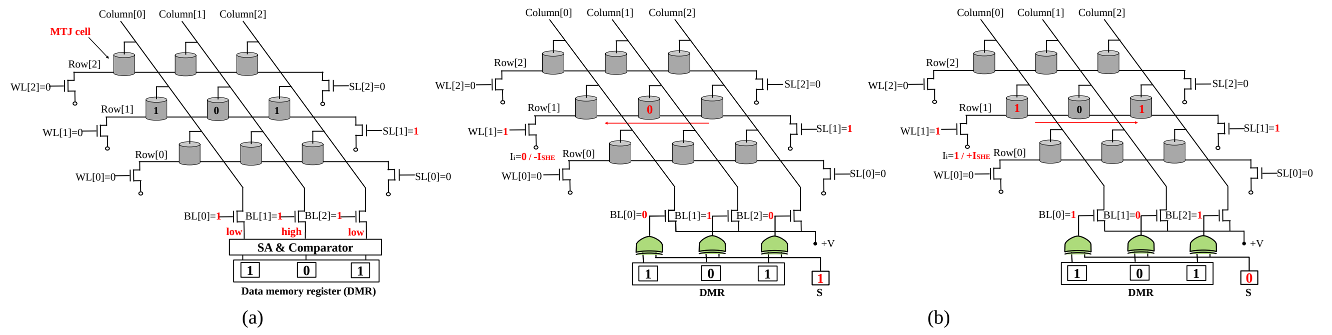


Figure 2: Illustration of (a) read; (b) 2-step write in VG-SHE crossbar. Step 1 writes 0's and step 2 writes 1's.

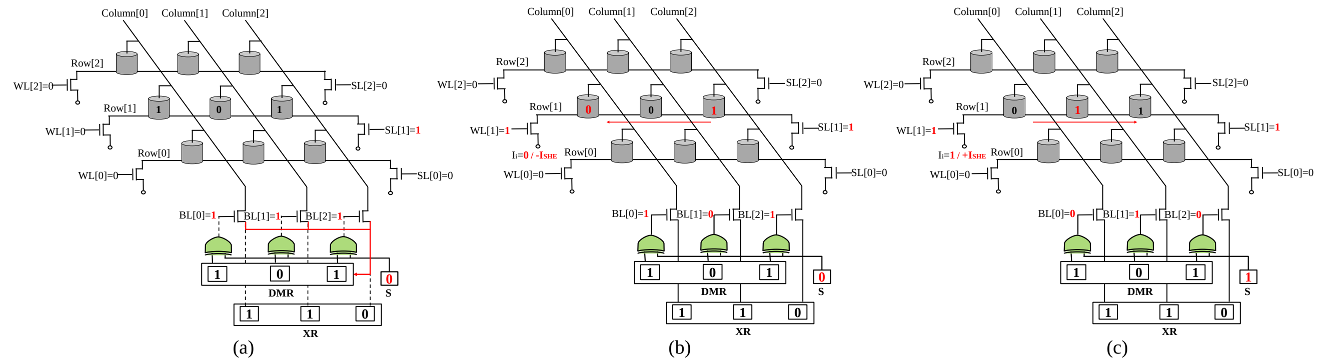


Figure 3: Implement XOR on a VG-SHE crossbar in 3 steps; (a) Data R_i=101 on the second row is read into DMR; (b) MTJs with R_i=1 and V=1 are programed to R_{i+1}=0; (c) MTJs with R_i=0 and V=1 are programed to R_{i+1}=1.

Algorithm 1 Keccak-f round function

```
Require: A: message block matrix; RC: round constant;
Ensure: Hashed message block A
    # $\theta$ step
 1: C[x] = A[x,0]⊕A[x,1]⊕A[x,2]⊕A[x,3]⊕A[x,4] ∀x in 0...4
 2: D[x] = C[x-1]⊕(C[x+1]≪1)                      ∀x in 0...4
 3: A[x,y] = A[x,y]⊕D[x]                     ∀(x,y) in (0...4,0...4)
    # $\rho$ and $\pi$ step
 4: B[y,2x+3y] = A[x,y]≪r[x,y]               ∀(x,y) in (0...4,0...4)
    # $\chi$ step
 5: A[x,y] = ($\overline{B[x+1,y]}$∧B[x+2,y])          ∀(x,y) in (0...4,0...4)
 6: A[x,y] = A[x,y]⊕B[x,y]                   ∀(x,y) in (0...4,0...4)
    # $\iota$ step
 7: A[0,0] = A[0,0]⊕RC
```

round takes two inputs: A is a 5×5 matrix with 64-bit elements, while RC is the round constant that varies across rounds. At line 4, $r[x,y]$ is the rotation matrix. Both RC and $r[x,y]$ are known and given in [8]. As the algorithm shows, the basic operations of SHA-3 are XOR, rotation, and AND, which can be efficiently implemented in a VG-SHE driven MTJ crossbar.

3 PROPOSED SHA-3 IMPLEMENTATION

This section presents the details on implementing SHA-3 on a VG-SHE driven MTJ crossbar, from circuit level all the way up to instruction level.

3.1 XOR and Write Implementations

While Figure 1(c) shows that XOR in a single MTJ can be achieved by setting $I=\overline{R_i}$, this is not the case for XORing a whole word line in the crossbar since $+I_{SHE}$ and $-I_{SHE}$ cannot co-exist on the same row at the same time.

To implement XOR, we introduce another register, i.e., the XOR register (XR). More importantly, since the BL switches need to be controlled by the data read from the crossbar, such data will be placed in DMR while the other input will be placed in XR. Figure 3 shows the steps for XORing R_i=101 and V=110 and getting R_{i+1}=011, which takes three cycles. In cycle 1, R_i is read into DMR, shown in the red path in Figure 3(a). In cycle 2, the MTJs with R_i=1 and V=1 are programed to R_{i+1}=0. This is achieved by setting S=0 which applies R_i=101 on the BL switches and hence conducts Column [0] and [2]. Meanwhile, 110 is applied on the bit lines, and $-I_{SHE}$ is flowed through the heavy metal. As a result, 0' is written to the MTJ on Column[0], while the MTJ on Column[2] remains unchanged. Finally, in cycle 3, the MTJs with R_i=0 and V=1 are programed to R_{i+1}=1. This is achieved by setting S=1 which applies $\overline{R_i}$=010 on the BL switches and hence conducts Column[1]. By applying 110 on the bit lines and flowing $+I_{SHE}$ through the heavy metal, 1' is written to the MTJ on Column[1].

Write operations in the VG-SHE driven MTJ crossbar can be optimized as well. While the original write operations introduced in [7] take two cycles for writing 0's and 1's, write in the proposed design is reduced to one cycle with the help of a precharge' operation. Specifically, by activating all the BL switches, applying positive

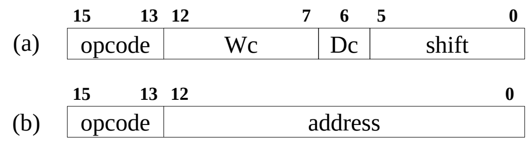


Figure 4: Instruction format. (a) Format of read, write, AND, OR, XOR. (b) Format of jump, branch, DMA, and precharge.

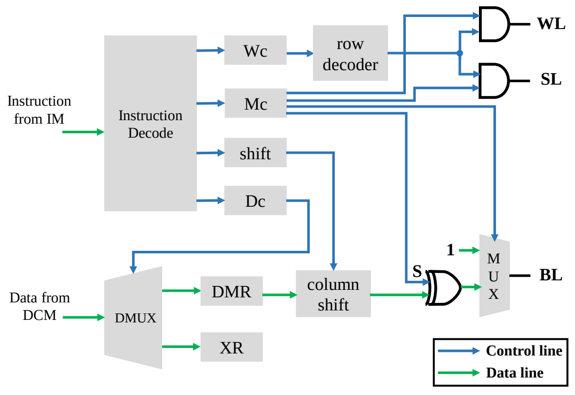


Figure 5: Generation of crossbar control signals

voltage on all the columns, and flowing $+I_{SHE}$ through multiple rows, all these rows are written with 1's simultaneously in one step. Later when writing data on these precharged rows, only 0's need to be programed which can be completed in one cycle.

3.2 Generation of Crossbar Control Signals

To realize both memory (read, write) and logic (XOR, AND, rotation) operations, we develop a PIM architecture similar to the ReVAMP platform [10]. The architecture uses a data and computation memory (DCM) which stores data and performs computation directly on data, as well as an instruction memory (IM) which stores the instructions for controlling the DCM. Both IM and DCM are VG-SHE driven MTJ cross-point arrays.

The proposed PIM architecture uses a fix-length instruction set. Two instruction formats are supported, shown in Figure 4. Most of instructions, including read, write, AND, OR, and XOR, are in the first format. It has four fields: a 3-bit opcode, a 6-bit wordline address W_c, a 1-bit signal D_c determining the read destination (1' for DMR and 0' for XR), and a 6-bit shift offset for implementing bitwise shift and rotation. Figure 5 shows the circuit for generating the signals for controlling the WL, SL and BL switches, selecting read destination, and implementing shift based on instruction decoding results. The second format is used for branch, jump, as well as direct memory access (DMA) instructions that can be used to load A[x,y] to DCM before starting the Keccak-f function. It is also used for implementing the aforementioned precharge instruction which requires a 6-bit starting address and a 6-bit ending address that can be held together in the 'address' field of the instruction.

3.3 Data Layout

As shown in Algorithm 1, each round of the Keccak-f function operates on four matrices A, B, C and D with sizes of 5×5, 5×5, 5×1, and 5×1, respectively. While the four matrices together require a memory of 60 words, our goal is to minimize the required DCM size, by allowing matrices with non-overlapping lifetime to share the same space. It turns out that matrices A and C, C and D, D and A, and A and B are dependent, while matrix B can overwrite the space occupied by C and D. Accordingly, a DCM of 50 words is sufficient to hold the four matrices, as shown in Figure 6.

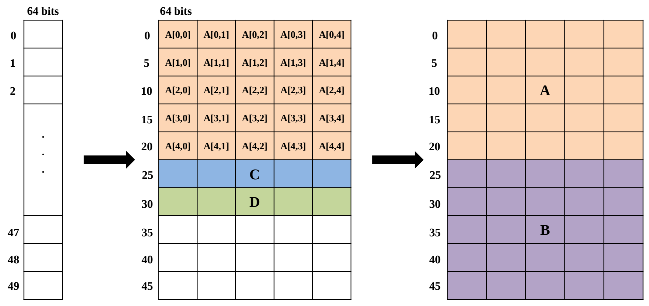

64 bits
0
1
2
47
48
49
64 bits
0
5
10
15
20
25
30
35
40
45
A[0,0] A[0,1] A[0,2] A[0,3] A[0,4]
A[1,0] A[1,1] A[1,2] A[1,3] A[1,4]
A[2,0] A[2,1] A[2,2] A[2,3] A[2,4]
A[3,0] A[3,1] A[3,2] A[3,3] A[3,4]
A[4,0] A[4,1] A[4,2] A[4,3] A[4,4]
C
D
A
B

Figure 6: DCM state transition. (a) DCM layout; (b) DCM state in step θ; (c) DCM state in step ρ, π and χ

Figure 6(a) shows the DCM layout with a word size of 64-bit and a length of 50 words. This is transformed into a 10×5 matrix representation in Figure 6(b) and 6(c), wherein each entry corresponds to 64 bits. Figure 6(b) shows the data layout in step θ. Words 0-24, 25-29 and 30-35 are respectively allocated to matrices A, C and D. Updates to matrix A[x,y] at line 3 of Algorithm 1 is performed in-place. Figure 6(c) shows the data layout in steps ρ, π, and χ. Words 25-49 are allocated to matrix B, which overwrites matrices C and D.

In the rest of this paper, all the logic and memory operations are assumed to operate on a word with all the 64 bits being processed simultaneously. It is also assumed that $A[x, y]$ is loaded into DCM before the Keccak-f function starts.

3.4 Keccak-f Implementation

Table 2 presents the implementation of the Keccak-f function step by step. This part describes each step in detail and computes the number of cycles and instructions needed, which are summarized in Table 3.

3.4.1 step θ. Line 1 in Algorithm 1 is a five-input XOR operation. Before starting it, one cycle will be spent on precharging words 25-34 (i.e., the locations of C[x] and D[x]) in DCM to 1. Then, A[x,0] is read into DMR and written to DCM on the address of C[x]. This is shown in Table 2. After that, A[x,1] is read into XR and XORed with the data at the location of C[x]. The second step is repeated 4 times with inputs A[x,1] to A[x,4]. Overall, it takes $(N_{read}+N_{write})+4\times(N_{read}+N_{xor})=18$ cycles and 2+4×2=10 instructions to update one element of C[x]. N_{read}, N_{write}, and N_{xor} denote the latencies of read, write, and XOR operations, which equal 1, 1, and 3, respectively. Since C[x] has 5 elements, line 1 of Algorithm 1 requires 5×18+1 (for precharging)=91 cycles and 50 instructions in total.

Line 2 in Algorithm 1 takes one read and one write to write C[x+1] and then one read and one XOR for XORing C[x-1] and C[x+1]. Data rotation $(C[x + 1] \ll 1)$ is achieved with a column

Table 2: Step-by-step implementation of Keccak-f. (* indicates bit shift during write)

Step	Op	Input	Output	Repeat
$\theta 1$	read	DCM (A[x,0])	DMR	
	write	DMR	DCM (C[x])	
	read	DCM (A[x,i])	XR	i=1 to 4
	XOR	XR, DCM (C[x])	DCM (C[x])	
$\theta 2$	read	DCM (C[x+1])	DMR	
	write*	DMR	DCM (D[x])	
	read	DCM (C[x-1])	XR	
	XOR	XR, DCM (D[x])	DCM (D[x])	
$\theta 3$	read	DCM (D[x])	XR	
	XOR	XR, DCM (A[x,i])	DCM (A[x,i])	i=0 to 4
ρ & π	read	DCM (A[x,y])	DMR	x=1 to 4
	write*	DMR	DCM (A[x,y])	y=1 to 4
$\chi 1$	read	DCM (B[x+2,y])	DMR	x=1 to 4 y=1 to 4
	write	DMR	DCM (A[x,y])	
	read	DCM (B[x+1,y])	DMR	
	AND	DMR, DCM (A[x,y])	DCM (A[x,y])	
$\chi 2$	read	DCM (B[x,y])	XR	x=1 to 4
	XOR	XR, DCM (A[x,y])	DCM (A[x,y])	y=1 to 4

shifter and does not bring any extra cycle, as shown in Figure 5. Overall, this line requires $5\times(N_{read}+N_{write}+N_{read}+N_{xor}) = 30$ cycles and 5×4=20 instructions in total.

Line 3 of Algorithm 1 updates the 5×5 state matrix A[x,y]. For each row, D[x] is read into the XR register once and used as the input for the entire row. Therefore, each row takes $N_{read}+5\times N_{xor} = 16$ cycles and 6 instructions. The entire state takes 80 cycles and 30 instructions to update.

3.4.2 steps ρ and π. This step reads from matrix A and writes to matrix B. Same as before, this step requires one cycle for precharging words 25-49 (i.e., the locations of B[x,y]) so as to save write latency. Then, A[x,y] is read into DMR and written into B[y,2x+3y]. Overall, this step requires 1+25×(N_{read} + N_{write})=51 cycles and 50 instructions.

3.4.3 step χ. This step computes matrix A based on matrix B. First, words 0-24 (i.e., the locations of A[x,y])) are precharged. Then, B[x+2,y] is read into DMR and then written into the location of A[x,y]. Next, B[x+1,y] is read into DMR and then ANDed with the data at the location of A[x,y]. Note that as Figure 1(c) shows, when I=0, R_{i+1}=$\overline{V}R_i$. In other words, by flowing $-I_{SHE}$ through the heavy medal, we directly get A[x,y]= $\overline{B[x+1,y]} \wedge B[x+2,y]$. This line requires 1+25×(N_{read}+N_{write}+N_{read}+N_{and})=101 cycles and 100 instructions to complete.

In line 6 of Algorithm 1, B[x,y] is readout and then XORed with A[x,y]. It requires 25×(N_{read}+N_{xor})=100 cycles and 50 instructions in total.

3.4.4 step ι. This step (line 7 of Algorithm 1) involves a single XOR of the round constant RC and the element $A[0,0]$. It takes (N_{read}+N_{xor})=4 cycles and 2 instructions to complete.

Table 3: Implementation details of SHA-3 round function

Step	Cycle #	Instruction #
$\theta 1$	91	50
$\theta 2$	30	20
$\theta 3$	80	30
ρ and π	51	50
$\chi 1$	101	100
$\chi 2$	100	50
ι	4	2
Total	457	302

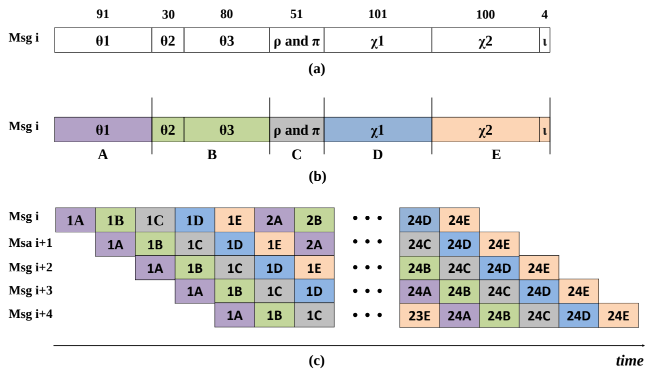


Figure 7: Different implementations of the Keccak function. (a) Unpipelined implementation. Each round takes 457cycles. (b) A 5-stage pipeline with a longest stage (B) of 110 cycles. Each round takes 550 cycles. (c) Steps for processing 5 messages for 24 rounds. '1A' indicates stage A of round 1.

3.5 Unpipelined vs. Pipelined Implementations

The proposed VG-SHE driven MTJ implementation of SHA-3 can be done in both unpipelined and pipelined forms. For single message hash (SMH), the input of each round function relies on the output of the previous round. As a result, an unpipelined implementation such as the one in Figure 7(a) is used. As each round takes N_{round}=457 cycles and loading the initial state A[x,y] takes 25 cycles, the overall latency of the unpipelined implementation is 24×N_{round}+25 = 10993 cycles.

While the different rounds of a single message cannot be pipelined because of data dependency, a pipelined implementation can be used for multiple message hash (MMH). Figure 7(b) shows a 5-stage pipeline which merges $\theta 2$ and $\theta 3$ as well as $\chi 2$ and ι. The pipeline length is constrained by the longest stage ($\theta 2$ and $\theta 3$) which takes 110 cycles. This implementation requires a larger DCM to hold 5 messages simultaneously (i.e., 250 words). as well as 5 individual ports so that different messages can be processed in parallel at different pipeline stages, as shown in Figure 7(c). It takes 110×(24×5+4)=13640 cycles to process 5 messages and 5 cycles to load the first message (since there are 5 ports), while the latency for loading the other messages is completely hidden. Therefore the overall latency of the pipelined implementation is 13645 cycles.

4 EXPERIMENTAL RESULTS

The proposed VG-SHE driven MTJ based crossbar memory is simulated with NVSim [14]. Both SMH and MMH implementations are

Table 4: Key parameters of NVSim

VG-MTJ [7]	RRAM [11]
feasure size = 80 nm	feasure size = 10 nm
read time = 0.5 ns	set time = 1 ns
switching time = 2.3 ns	reset time = 1 ns
read energy = 5 fJ	read energy = 0.6 pJ
write energy = 12 fJ	write energy = 0.6 pJ
low resistance = 50 $k\Omega$	low resistance = 100 $k\Omega$
high resistance = 100 $k\Omega$	high resistance = 1000 $k\Omega$
DCM size = 400 bytes	DCM size = 344 bytes

Table 5: Results of NVSim

Tech.	area mm^2	energy μJ	frequency MHz	Efficiency
RRAM	0.1438	1.53	471.70	83.55
VG-MTJ (SMH)	0.3608	0.39	401.61	282.5
VG-MTJ (MMH)	1.4263	0.40	392.15	274.0

Table 6: Comparison of different SHA-3 implementations

Tech.	f MHz	Area	Latency cycles	Throughput Mbps
Virtex5[12]	248	134 slices	1730	250
Virtex6[13]	285	188 slices	466	77
ReVAMP[9]	472	0.144 mm^2	27920	18.38
Pro.(SMH)	402	0.361 mm^2	10993	39.75
Pro.(MMH)	392	1.426 mm^2	13645	156.34

simulated. Meanwhile, the RRAM-based SHA-3 implementation introduced in [9] is also simulated. The key device parameters are listed in Table 4, while the simulation results are shown in Table 5.

Compared with RRAM, the proposed VG-SHE driven MTJ-based crossbar consumes much lower energy per round, as a result of the fewer cycles and lower read/write energy of the device. On the other hand, it requires more area due to the larger feature size of the device. Note that the area of MMH is more than 5 times of the area of SMH due to the need of extra I/O ports as well as registers between pipeline stages. Finally, cycle time is computed as the longest read/write latency, and frequency is determined accordingly. The cycle times of VG-SHE driven MTJ-based crossbar is about 18%–20% slower than RRAM. However, this does not necessarily implies performance degradation since the proposed PIM impelementation requires fewer cycles to process each round of the message hash.

Table 6 compares the proposed SMH and MMH implementations with existing FPGA-based (first two rows) and RRAM-based (3rd row) SHA-3 implementations. Data of frequency, area, latency, and throughput are reported. Moreover, the efficiency of the PIM implementations is reported in Table 5. Throughput and efficiency are computed with the following equations:

$$Throughput = \frac{Blocksize \times \#MH}{Latency} \times Frequency \quad (2)$$

$$Efficiency = \frac{Throughput}{Area \times Energy} \quad (3)$$

Blocksize is the length of the message block processed by the hash function at a given time, and #MH equals 1 and 5 in SMH and MMH, repestively. Each slice of Vertex 5 and 6 family is comprised of 450 and 256 bits, respectively. The proposed PIM design, in comparison to the FPGA-based implementations, implements a lightweight controller and requires far less hardware. In terms of latency, the proposed SMH and MMH implementations outperform the RRAM implementation by 2.54 and 2.05 times, respectively. Such latency improvement directly leads to 2.16 and 8.51 times throughput improvement, thus confirming the performance advantage of the proposed designs. Finally, in terms of design efficiency, the proposed VG-SHE driven MTJ implementations are more than 3 times of ReVAMP, making it a suitable candidate for resource constrained systems.

5 CONCLUSIONS

NVM-based Processing-In-Memory (PIM) architectures open up a new direction to break through the bottleneck in traditional von Neumann architectures. This paper presented a PIM architecture built with VG-SHE driven MTJ devices and showed a comprehensive implementation of SHA-3, including the design of the crossbar circuit, the instruction set, the step-by-step implementation of Keccak-f function, and both unpipelined single message hash (SMH) and pipelined multiple message hash (MMH) implementations. Our design outperforms FPGA-based and RRAM-based implementations in terms of throughput and efficiency, making it a suitable candidate for resource constrained systems.

REFERENCES

[1] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "'Memristive' switches enable 'stateful' logic operations via material implication," *Nature*, vol. 464, no. 7290, p. 873, 2010.

[2] P.-E. Gaillardon, L. Amarú, A. Siemon, E. Linn, R. Waser, A. Chattopadhyay, and G. De Micheli, "The programmable logic-in-memory (PLiM) computer," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, pp. 427–432.

[3] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Magic-Memristor-Aided Logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.

[4] S. Gao, G. Yang, B. Cui, S. Wang, F. Zeng, C. Song, and F. Pan, "Realisation of all 16 Boolean logic functions in a single magnetoresistance memory cell," *Nanoscale*, vol. 8, no. 25, pp. 12 819–12 825, 2016.

[5] C. Wan, X. Zhang, Z. Yuan, C. Fang, W. Kong, Q. Zhang, H. Wu, U. Khan, and X. Han, "Programmable spin logic based on spin hall effect in a single device," *Advanced Electronic Materials*, vol. 3, no. 3, 2017.

[6] J. Lee, D. I. Suh, and W. Park, "The universal magnetic tunnel junction logic gates representing 16 binary Boolean logic operations," *Journal of Applied Physics*, vol. 117, no. 17, 2015.

[7] H. Zhang, W. Kang, L. Wang, K. L. Wang, and W. Zhao, "Stateful reconfigurable logic via a single-voltage-gated spin hall-effect driven magnetic tunnel junction in a spintronic memory," *IEEE Transactions on Electron Devices*, vol. 64, no. 10, pp. 4295–4301, 2017.

[8] P. Pritzker and P. Gallagher, "SHA-3 standard: Permutation-based hash and extendable-output functions," *Information Tech Laboratory National Institute of Standards and Technology*, pp. 1–35, 2014.

[9] D. Bhattacharjee, V. Pudi, and A. Chattopadhyay, "SHA-3 implementation using ReRAM based in-memory computing architecture," in *18th International Symposium on Quality Electronic Design (ISQED)*, 2017, pp. 325–330.

[10] D. Bhattacharjee, R. Devadoss, and A. Chattopadhyay, "ReVAMP: ReRAM based VLIW architecture for in-memory computing," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017, pp. 782–787.

[11] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, "Metal–oxide RRAM," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 1951–1970, 2012.

[12] J. Winderickx, J. Daemen, and N. Mentens, "Exploring the use of shift register lookup tables for Keccak implementations on Xilinx FPGAs," in *26th International Conference on Field Programmable Logic and Applications (FPL)*, 2016, pp. 1–4.

[13] S. Kerckhof, F. Durvaux, N. Veyrat-Charvillon, F. Regazzoni, G. M. de Dormale, and F.-X. Standaert, "Compact FPGA implementations of the five SHA-3 finalists," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2011, pp. 217–233.

[14] X. Dong, C. Xu, N. Jouppi, and Y. Xie, "NVSim: A circuit-level performance, energy, and area model for emerging non-volatile memory," in *Emerging Memory Technologies*. Springer, 2014, pp. 15–50.