Accelerating Low Bit-Width Deep Convolution Neural Network in MRAM

Zhezhi He*, Shaahin Angizi† and Deliang Fan‡
Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL 32816
Email: {*Elliot.he, †Angizi}@knights.ucf.edu, ‡Dfan@ucf.edu

***Abstract*—Deep Convolution Neural Network (CNN) has achieved outstanding performance in image recognition over large scale dataset. However, pursuit of higher inference accuracy leads to CNN architecture with deeper layers and denser connections, which inevitably makes its hardware implementation demand more and more memory and computational resources. It can be interpreted as 'CNN power and memory wall'. Recent research efforts have significantly reduced both model size and computational complexity by using low bit-width weights, activations and gradients, while keeping reasonably good accuracy. In this work, we present different emerging non-volatile Magnetic Random Access Memory (MRAM) designs that could be leveraged to implement 'bit-wise in-memory convolution engine', which could simultaneously store network parameters and compute low bit-width convolution. Such new computing model leverages the 'in-memory computing' concept to accelerate CNN inference and reduce convolution energy consumption due to intrinsic logic-in-memory design and reduction of data communication.**

***Index Terms*—Neural network acceleration, In-memory computing, Magnetic Random Access Memory**

I. Introduction

In virtue of the fast development of the deep learning algorithm, design of a highly parallel and energy-efficient neural network accelerator has recently drawn tremendous research interest. On the one hand, a great deals of model compression techniques (e.g., pruning, parameter quantization, model encoding [1]) have been explored to lower the network model size, thus reducing the memory and computation cost. On the other hand, to optimize the massive data communication used in neural network inference, the 'wall' between memory and processor in classical Von-Neumann computing architecture has been crashed while the in-memory computing has emerged as a promising candidate for deep neural network acceleration [2]. In this work, we will focus on neural network binarization in algorithm level and its hardware implementation using Magnetic Random Access Memory (MRAM) based in-memory computing technique.

As the first work which successfully provides the deep convolutional neural network with binarized weight (i.e. -1 and +1), BinaryConnect [3] achieves considerably negligible accuracy degradation on small datasets like MNIST [4] and CIFAR-10 [5]. The following work, BNN [6], aggressively converts both weights and interlayer tensors into binary representation (i.e. -1 and +1), which correspondingly transforms the dominant computation of convolution layer from multiplication and additions to bulk bit-wise XNOR operation. Such binarization scheme in BNN reveals the path for designing a neural network inference accelerator with extreme low bit-with weight/interlayer-tensor. However, the binarization method in BNN is criticized by researchers and engineers about its poor inference accuracy on large dataset, like ImageNet [7]. In order to solve the severe accuracy degradation degradation issue, a series of optimized neural network binarization techniques has been proposed and discussed in XNOR-Net [8], Dorefa-Net [9] and ABC-Net [10]. The essential tricks extracted from the aforementioned works can be summarized and further optimized as (1) introducing scaling factor for both binarized interlayer tensor and weight, and (2) using binarization functions with various thresholds (e.g. vanilla binarization function taken 0 as default threshold) to avoid information loss, which will be discussed in the main body.

Even though network quantization techniques have made deep neural network model compact and hardware-friendly, its inference computing is still hampered by the limited memory bandwidth. The newly announced NVIDIA Volta GPU [11] has adopted the brand new DDR-6 (i.e. 3D stacked memory) technologies which significantly improves the bandwidth by $4\times$ ($\sim$ 1 TB/s). However, the growing speed of memory consumption for the state-of-the-art network structure is overwhelming the hardware evolution speed. For example, the extremely dense structure in DenseNet [12] raises the memory usage in quadratic manner when the network goes deeper. Therefore, in-memory computing has emerged as a promising countermeasure to eliminate the long-distance data communication through merging the storage and computing component together. In this paper, we introduce two variant in-memory computing model using Spin-Transfer Torque Magnetic Random Access Memory (STT-MRAM) and domain wall based racetrack memory, respectively. As the emerging nonvolatile memory technologies, both STT-MRAM and racetrack memory own the characteristic of on-volatility, zero standby leakage, high write/read speed, compatibility with CMOS fabrication process, scalability, superior endurance, excellent retention time and high integration density [13], [14]. With the moderate adjustment on peripheral circuitry or device structure itself, we could perform bit-wise bulk logic (i.e. AND/OR/XOR and their complementary) taken two selected bit-cells as inputs. With the assistance of model binarization for both weight and interlayer tensor, the dominant amount of operations is converted into bit-wise XNOR operations which is suitable to take the aforementioned in-memory computing

2159-3477/18/$31.00 ©2018 IEEE
DOI 10.1109/ISVLSI.2018.00103

IEEE computer society

architecture as the computing accelerator.

II. Low bit-width network quantization

In order to obtain the low bit-width quantized neural network with minimum accuracy degradation, one argument have been adopted in almost all the related works is that there is supposed to be two systems of model parameters: one model with full precision (i.e. 32bit floating point) parameter $\boldsymbol{w}$ and one model with corresponding quantized parameters $\boldsymbol{w}_q$. For each parameters optimization iteration during the training process, the full precision $\boldsymbol{w}$ will be updated first, then the $\boldsymbol{w}_q$ will be calculated correspondingly. In this work, we mainly discuss the binary format weight and interlayer tensor. The mathematical formula for weight $\boldsymbol{w}$ and interlayer tensor $\boldsymbol{x}$ binarization firstly discussed in [6] can be described as:

$$\text{Forward}: \quad q = Sign(r) = \begin{cases} +1 & if\ r \geq 0 \\ -1 & otherwise \end{cases} \tag{1}$$

$$\text{Backward}: \quad \frac{\partial g}{\partial r} = \begin{cases} \frac{\partial g}{\partial q} & if\ |r| \leq 1 \\ 0 & otherwise \end{cases} \tag{2}$$

where q is the input ($\boldsymbol{w}$ or $\boldsymbol{x}$) while r is the output ($\boldsymbol{w}_q$ or $\boldsymbol{x}_q$) to the binarization function. In the forward path (i.e. inference phase), $\boldsymbol{w}$ or $\boldsymbol{x}$ is binarized using $Sign()$ function. Note that, the sign function owns zero derivatives almost everywhere, which makes it impossible to calculate the gradient using chain rule in backward path (i.e. training phase). Thus, the Straight-Through Estimator (STE) [6], [15] is applied to calculate gradient in this work. In the backward path, the input gradient of binarization activation function clones the gradient at output, if the input q is in the range from -1 to +1. Otherwise, the gradient is cancelled to preserve training performance. Furthermore, more works [8], [16] find that, in order to gain high accuracy, scaling factor plays vital role in the binarized neural network. There are several solutions to introduce the scaling factors, such as (1) inserting the batch normalization layer [17] right before the interlayer tensor binarization function [8], and (2) replacing the activation function with Parametric Rectifier Linear Unit (PReLU) [16]. Through our investigation, we find that inserting batch normalization layer before the interlayer tensor binarization function works better for scaling the interlayer tensor, since the distribution varies for each input. However, for the weight scaling factor computation, the current best solution is to iteratively compute based on the weight distribution during training, which can be written as:

$$\text{Forward}: \quad q = Sign(r) = \begin{cases} +E(|W_l|) & if\ r \geq 0 \\ -E(|W_l|) & otherwise \end{cases} \tag{3}$$

where $E(|W_l|)$ is the mean of the absolute value of full precision weights in l_{th} layer. Note that, the backward path of Eq. (3) is identical to Eq. (2). Owing to the computation of convolution layer or linear layer is linear transformation (i.e. dot-product), the layer-wise weight scaling factor $E(|W_l|)$ can be extracted and integrated with following activation function or batch normalization function to perform element-wise computation. Therefore, the computation for one convolution layer or linear layer can be described as:

$$\boldsymbol{x}_{\mathrm{q}}^T \cdot \boldsymbol{w}_{\mathrm{q}} = \sum_{i=1}^{N} x_{\mathrm{q},i} \cdot w_{\mathrm{q},i} \quad \forall x_{\mathrm{q},i}, w_{\mathrm{q},i} \in \{-1, +1\} \tag{4}$$

where $\boldsymbol{x}_{\mathrm{q}}$ and $\boldsymbol{w}_{\mathrm{q}}$ are the vectorized form of quantized interlayer tensor and weight. N is the vector size of $\boldsymbol{x}_{\mathrm{q}}$. Since $\boldsymbol{x}_{\mathrm{q}}$ and $\boldsymbol{w}_{\mathrm{q}}$ only consist of -1 and +1, in order to map the computation of $x_{q,i} \cdot w_{q,i}$ into hardware, we use single bit of 0 and 1 to represent -1 and +1 respectively. For the converted form $x'_{q,i}$ and $w'_{q,i}$, the computation of $x_{q,i} \cdot w_{q,i}$ is equivalent to XNOR($x'_{q,i}$,$x'_{q,i}$) as the truth tables shown in Table I and Table II.

TABLE I
Truth table for original binarized dot-product

Input		Output
x_i	w_i	$x_i \cdot w_i$
-1	-1	+1
-1	+1	-1
+1	-1	-1
+1	+1	+1

TABLE II
Truth table for converted binarized dot-product

Input		Output
x'_i	w'_i	XNOR(x'_i,w'_i)
0	0	1
0	1	0
1	0	0
1	1	1

Thus, the computation in Eq. (4) can be transformed into bitwise XNOR, and bit-count operations without multiplications [9]:

$$\boldsymbol{x}_{\mathrm{q}}^T \cdot \boldsymbol{w}_{\mathrm{q}} = 2 \cdot bitcount(\mathrm{XNOR}(\boldsymbol{x}_{\mathrm{q}}, \boldsymbol{w}_{\mathrm{q}})) - N \tag{5}$$

where $bitcount()$ function function count the number of '1'. Based on such reformulated dot-product equation, we can utilize the counter and XNOR logic gate as the primary computing element to accelerate the neural network inference in energy-efficient manner. Moreover, in order to encounter the aforementioned limited memory bandwidth issue in the Section I, we leverage the MRAM based in-memory computing techniques to further boost the performance of such binarized neural network accelerator which will be elaborated in the following sections.

Beyond that, for contradicting the argument that network binarization can lead to significant accuracy loss which makes such extreme low bit-width quantization scheme not practical in the real world applications, recent research efforts in [10], [16] has brought up multiple binarization method to compensate the information loss due to the aggressive quantization. Thus, further integrating such multiple binarization with in-memory computing technique will provide accurate deep neural network inference result with high throughput.

III. MRAM-based neural network acceleration

In this section, we will first provide an over-view introduction for low bit-width CNN accelerator in block level. Then, as the primary computing unit, several different designs utilizing STT-MRAM array and racetrack memory based magnetic crossbar will be introduced to perform bit-wise XNOR operations.

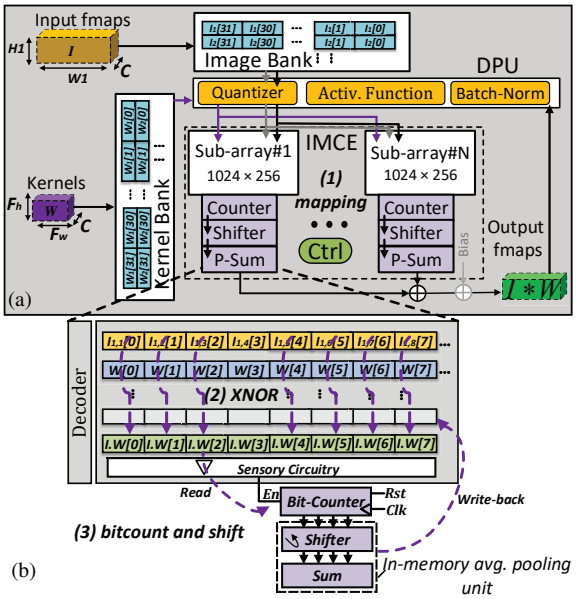


Fig. 1. (a) General overview of the CNN accelerator with image bank, kernel bank, computational sub-arrays, and DPU, (b) Bit-wise IMCE's sub-array.

The general overview of the system architecture for performing low bit-width CNN is shown in Fig. 1.a [18]. This architecture mainly consists of Image Bank, Kernel Bank, bit-wise In-Memory Convolution Engine (IMCE), and Digital Processing Unit (DPU). Since linear layer can be visualized as convolution layer with 1×1 kernel size, the computation of linear layer could be implemented by the same convolution accelerator as well. Assuming the Input feature maps (I) and Kernels (W) are initially stored in the Image Banks and Kernel Banks of memory, respectively. As depicted in Fig. 1.a, inputs need to be constantly quantized before mapping into computational sub-arrays. However, quantized shared kernels can be utilized for different inputs. This step is performed using DPU's Quantizer and then the results are mapped to IMCE's sub-arrays (Fig. 1.b). For the realization of bit-wise IMCE, several different in-memory computing techniques are readily utilized such that ultra-efficient and parallel in-memory *XNOR* operations required for convolutions can be handled. The functionality of peripheral components are elaborated as follow:

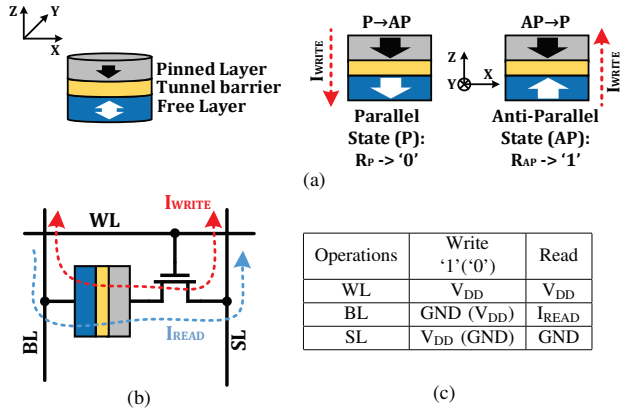
- **Quantizer**: This unit binarize the interlayer tensor or weight w.r.t Eq. (1), Eq. (2) and Eq. (3).
- **Batch-Norm**: Batch Normalization layer [17] alleviates the information loss caused by weight and activation binarization, through normalizing the input mini-batch to have zero mean and unit variance. The batch normalization function can be considered as an affine function $\boldsymbol{y} = k\boldsymbol{x} + b$ [19], where

$$k = \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}} \quad \text{and} \quad b = \beta - \frac{\mu\gamma}{\sqrt{\sigma^2 + \epsilon}} \tag{6}$$

- **Activ. Function**: the activation function module perform the element-wise computation, which normally takes ReLU as the activation function.

Moreover, as the vital component of the low bit-width accelerator, IMCE mainly consists of in-memory computing sub-array, bit-counter, shifter. The in-memory computing sub-array plays the role of both on-chip buffer and the computing bit-wise computing unit. In this work, we enumerate two variant in-memory computing designs leveraging STT-MRAM [2] and racetrack memory based magnetic crossbar [20] that are in-charge for the most computationally-intensive operations i.e. *XNOR*. After this computation, a CMOS counter unit is devised to count the number of +1 elements within generated XNOR vector. According to Eq. (5), the result of bitcount operation should be multiplied by 2, which is implemented by shifter unit in our design. Then, the produced binary result is processed in parallel by partial sum (subtract) units.

A. STT-MRAM array



Operations	Write '1'('0')	Read
WL	V_{DD}	V_{DD}
BL	GND (V_{DD})	I_{READ}
SL	V_{DD} (GND)	GND

Fig. 2. (a) Device structure of conventional magnetic tunnel junction in parallel- and anti-parallel states, with current-induced spin-transfer torque switching scheme. (b) Bit-cell schematic of 1T1R STT-MRAM. (c) Biasing conditions for STT-MRAM operations.

1) memory mode: A typical Magnetic Tunnel Junction (MTJ) structure, as shown in Fig.2a, consists of two ferromagnetic layers with a tunnel barrier sandwiched between them. Due to the Tunnel MagnetoResistance (TMR) effect [21], the resistance of MTJ is high (low) when the magnetization of two ferromagnetic layers are in anti-parallel (parallel) state. The *TMR ratio* is defined as (R_{AP}-R_P)/R_P, which may vary from 10% to 400% depending on materials and temperature [21]. Thus, the data are stored as the magnetization direction in the free layer, which could be programmed through current induced Spin-Transfer Torque (STT). Note that, the MTJ with Perpendicular Magnetic Anisotropy (PMA) is used in this work. The 1T1R bit-cell is widely used in the typical STT-MRAM design, as depicted in Fig. 2b, which is correspondingly controlled by Bit Line (BL), Word Line (WL) and Source Line (SL). The biasing conditions of memory read and write are presented in Fig. 2c. For both memory read and write operation, the WL is enabled, which turns on the access transistor. Then, a voltage drop -V_{DD} or +V_{DD} is applied across the BL and SL, in order to realize write '1' or '0' respectively. For memory read, a sensing current (I_{READ}) is applied on the

BL and consequently generates a sensing voltage, which can be detected by sense amplifier.

B. computing mode

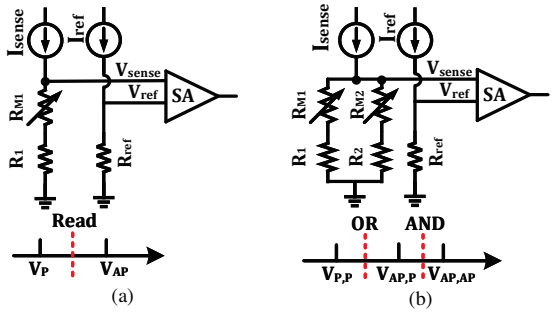


Fig. 3. The idea of voltage comparison between V_{sense} and V_{ref} for (a) memory read and (b) in-memory logic operation.

The key idea to perform memory read and in-memory computing is to choose different thresholds when sensing the selected memory cell(s). As shown in Fig. 3a, for memory read operation, a single memory cell is addressed and routed in the memory read path to generate a sense voltage (V_{sense}), which will be compared with a reference voltage (V_{ref}). Owing to the parallel- or anti-parallel state of selected STT-MRAM bit-cell (R_{M1}), the sense voltage are V_P or V_{AP} (V_P<V_{AP}) respectively. Thus, through setting the reference voltage at (V_{AP}+V_P)/2, the sense amplifier outputs binary '1' when V_{sense}>V_{ref}, otherwise the sense amplifier outputs '0'. the sensing-based method of in-memory Boolean computing is depicted in Fig. 3b, where two memory bit-cells (R_{M1} and R_{M2}) are sensed simultaneously. R_1 and R_2 corresponds to the access transistors within the sensing path. Owing to the different resistance combinations of two selected STT-MRAM bit-cells (i.e. R_{AP}, R_{AP}; R_{AP}, R_P; R_P, R_P), three different sense voltages V_{sense} ($V_{AP,AP}$; $V_{AP,P}$; $V_{P,P}$) could be generated respectively. Consider setting the reference voltage as ($V_{AP,AP}$+$V_{AP,P}$)/2 through tuning reference resistance, the sense amplifier only outputs '1' when both selected STT-MRAMs are in anti-parallel state (V_{sense}>V_{ref}). Thus, this sensing operation with modified reference voltage performs an AND logic operation taken the binary data stored in R_{M1} and R_{M2} as logic inputs. Similarly, when the reference voltage is shifted to ($V_{P,P}$+$V_{AP,P}$)/2, the OR logic operation can be performed as well. Therefore, through tuning the reference voltage for comparison, the sense amplifier can perform reconfigurable in-memory computations.

In order to build the in-memory computing STT-MRAM array to perform bitwise XNOR operation, further modification is made on the peripheral control and sensing circuit. Fig. 4a depicts the architecture of STT-MRAM based IMCE, where memory read/write path of the specific bit-cell is enabled by the row/column decoders. As shown in Fig. 4b, the modified row/column decoders can enable either single line (memory write/read) or double lines (bit-wise Boolean computation), depending on the addresses (Addr1 and Addr2) provided. For memory write, the voltage drop across BL and SL is generated

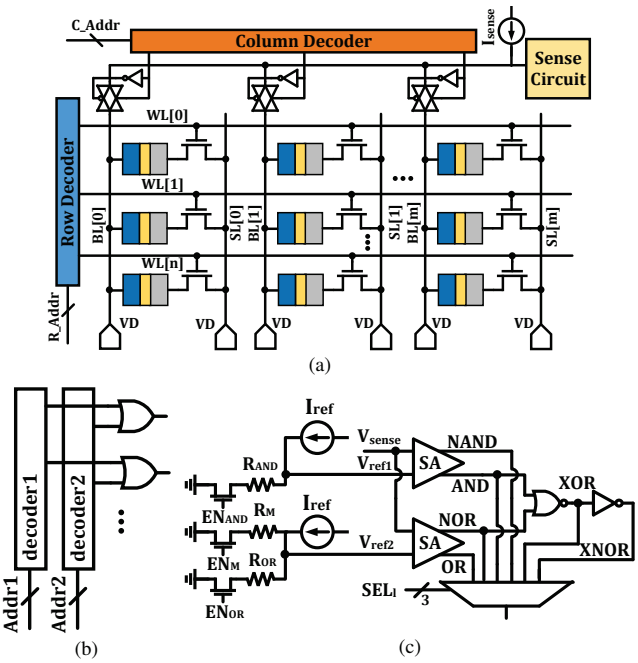


Fig. 4. (a) The modified sub-array structure of STT-MRAM.(b) Modified decoder which provides single/multiple lines enable function. (c) Modified sense circuit for regular memory W/R and in-memory computing operations.

by the Voltage Drivers (VD), which realize the fast memory switching [2]. For memory read and Boolean computation, a small sense current ($I_{sense} \simeq 3\mu A$) is injected into the read path to generate a sense voltage (V_{sense}), which is taken as the input of modified sense circuit. As shown in Fig. 4c, the modified sense circuit can provide memory read, AND/NAND, OR/NOR and XOR/XNOR functions, through combining two sense amplifiers (i.e. StrongARM latch [22]), external CMOS logic gate and control units. Owing to the complementary outputs of SA, the modified sensing circuit can provide NAND and NOR without additional cost. Moreover, according to the Boolean representation of $p \bigoplus q = (p \vee q) \wedge \overline{(p \wedge q)}$, the XOR and XNOR can be realized with two sense amplifiers (i.e. performing AND and NOR logic respectively) and an additional CMOS NOR gate. The operation of such sense circuit is determined by the control signals (EN_{AND}, EN_{OR} and EN_M), while the desired result is acquired by the selection signal (SEL) of the output multiplexer. Note that, only one sense amplifier is used during AND/OR/memory-read operation, in order to reduce the power consumption of sensing.

Moreover, transient simulation result of the sense circuit under a 2ns period clock signal (CLK) is included in Fig. 5 to validate the in-memory computing functionality. It takes the data stored in MRAM1 and MRAM2 as inputs. When CLK is high, the sense amplifier is in pre-charge phase and the output is reset to '0'. When CLK is low, the sense amplifier is in sampling phase, and generates logic computation result depending on the reference voltage configuration. V_{cmp} includes all the input signals of SAs, which are sense voltage (V_{sense}) and two reference voltage (V_{ref1} and V_{ref2}). V_{ref1} is set to ($V_{AP,AP}$+$V_{AP,P}$)/2, and V_{ref2} is set to ($V_{P,P}$+$V_{AP,P}$)/2, for

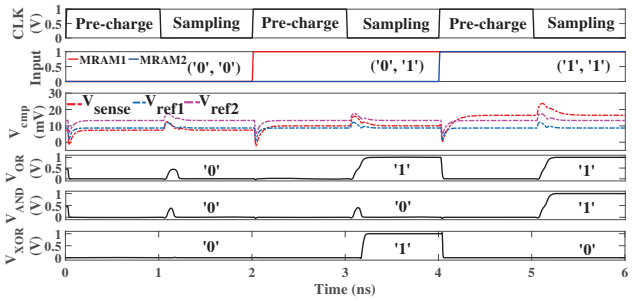


Fig. 5. Transient simulation results of in-memory computing operations (i.e. AND, OR and XOR).

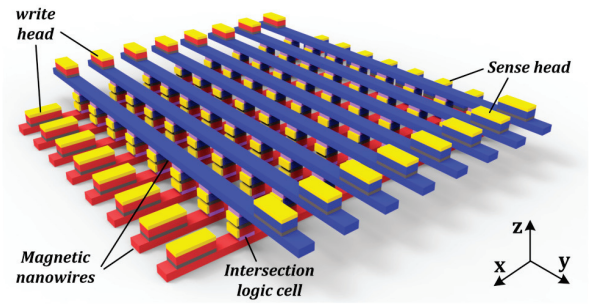
performing AND and OR respectively. Through the combination of two SAs in Fig. 4c, XOR is correctly performed as well. The XNOR bit-wise logic could be easily obtain with an cascaded NOT gate.

C. Magnetic crossbar

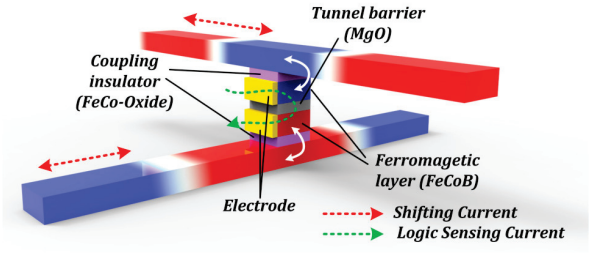
In this subsection, we describe a magnetic crossbar architecture consisting of perpendicularly coupled magnetic domain wall motion racetracks [20], which is able to morph between two modes: memory and computation mode. Different from that the STT-MRAM based in-memory computing structure which can perform AND/OR/XOR and theirs complementary, the introduced magnetic crossbar can only perform XOR/XNOR.

1) memory mode: In the memory mode, all the magnetic domain wall motion racetrack nano-wires are employed as conventional magnetic racetrack memory for non-volatile multi-bit data storage [23]. The device structure of the computational magnetic crossbar design is shown in Fig. 6a, which mainly constitutes of equally spaced ferromagnetic nanowires both longitudinally and latitudinally. Each nanowire could work individually as a normal domain wall racetrack memory to store the interlayer tensor and weight, where binary data are represented by the magnetization directions and stored in the form of domain wall pair train within the nanowires [23]. In this work, we define +z oriented magnetization (in red) as binary '1', and -z oriented magnetization (in blue) as binary '0'. Write and read operations are performed through the write head (spin polarizer) and read head (sensing MTJ) mounted on the two sides of nanowires [20].

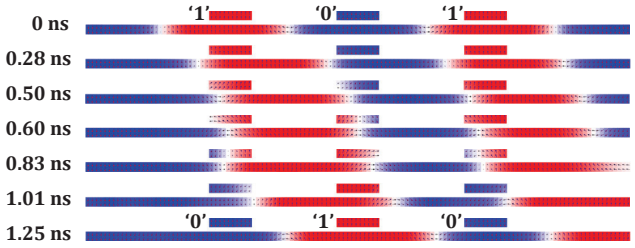
2) compute mode: In order to realize the in-memory XOR/XNOR operation, we modify the normal racetrack memory through introducing XOR/XNOR joint (i.e. a multi-layer structure) at the intersection of each longitudinal and latitudinal nanowire. It consists of coupling insulator (FeCo-Oxide [24]), ferromagnetic layer (FeCoB), tunnel barrier(MgO), ferromagnetic layer (FeCoB) and coupling insulator (FeCo-Oxide), from top to bottom as shown in Fig. 6b. Owing to the magnetic coupling effect [24], two ferromagnetic layers are consistently staying in the identical magnetization state with corresponding top and bottom nanowire as indicated by the white arrow in Fig. 6b. When the domain wall nanowire is



(a)



(b)



(c)

Fig. 6. (a) The device structure of magnetic crossbar in the size of 8×8. (b) The XOR logic cell in the intersection of longitudinal and latitudinal domain wall nanowires. (c) The micromagnetic simulation of current induced domain wall motion and magnetic coupling effect.

written with a new data followed by a domain wall pair shifting operation, the ferromagnetic layers switch their magnetization direction simultaneously due to magnetic coupling. As the transient micromagnetic simulation result shown in Fig. 6c, the initial binary bits (1, 0, 1) are stored in the bottom nanowire at 0ns. The coupled ferromagnetic layers have the same magnetization with the corresponding bottom domain wall nanowire. Through injecting lateral shifting current into the bottom nanowire, the domain wall pairs start to shift from L.H.S to R.H.S, which simultaneously rotate the magnetization of the coupled ferromagnetic layers. As shown in Fig. 6b,the ferromagnetic layers that respectively couple with the top and bottom magnetic nanowires can jointly work with the tunnel barrier layer to form a Magnetic Tunneling Junction (MTJ). In this design, the binary bit A and B located within intersection region are taken as the inputs of XOR computation, while the intersectional MTJ resistance (R_{MTJ}) represents the output result ($A \oplus B$).

As listed in Table. III, when the magnetization of two

TABLE III
THE XOR LOGIC OPERATION IN MAGNETIC CROSSBAR

A	B	$A \oplus B$	M_A	M_B	R_{MTJ}
0	0	0	Down	Down	R_P
0	1	1	Down	Up	R_{AP}
1	0	1	Up	Down	R_{AP}
1	1	0	Up	Up	R_P

ferromagnetic layers (M_A and M_B) are both up ('1') or down ('0'), the intersectional MTJ cell is in parallel state with low resistance (R_P). Otherwise, when M_A and M_B are in the opposite orientation (i.e. one up and one down), the intersectional MTJ cell is in anti-parallel state with high resistance (R_{AP}). Based on this description, we could see that the XOR computation is automatically implemented by the magnetic coupling physics when the data are loaded into the top and bottom domain wall nanowires with no cost of additional operations, leading to extreme energy efficient logic in-memory design. In order to read out the computation result, a similar circuit as Fig. 4a is needed. Furthermore, since the StrongARM sense amplifier generates complementary output, the XNOR(A,B) can be computed with no extra cost.

TABLE IV
ENERGY COST FOR $w \cdot x$ W.R.T MEMORY ACCESS AND COMPUTATION

	Baseline (32bit *mul*)	STT-MRAM (1bit *xnor*)	Magnetic crossbar (1bit *xnor*)
memory access	65.6nJ	1.6nJ/-	4.8nJ/-
compute	3.7pJ	0.9nJ	0.8nJ

In general, two designs with STT-MRAM and magnetic crossbar are introduced in details to provide in-memory bit-wise XNOR operation through either circuit level or device level modification. In comparison with normal network accelerator design [25] using 32bit floating multiplication (i.e., baseline in Table IV), the computation of dot-product involves energy-expensive off-chip DRAM access to load the model parameter w to the processing element, and the on-chip SRAM access to load interlayer tensor x. Such long-distance data communication is avoided with our in-memory bit-wise computing method, and the energy consumption is shown in Table IV. Note that, the memory access for in-memory computing with STT-MRAM and magnetic crossbar is only required when w and x are not stored in the same memory subarray.

IV. SUMMARY

In this work, we explicitly discuss the method to leverage in-memory XNOR computation using two different computational MRAM designs to accelerate the state-of-the-art binarized deep neural network, spanning across algorithm, architecture, circuit and device. Great energy efficiency is achieved due to its intrinsic in-memory logic designs and processing-in-memory architecture to reduce off-chip memory access.

Acknowledgement: This work is supported in part by the National Science Foundation under Grant No. 1740126 and Semiconductor Research Corporation nCORE.

REFERENCES

[1] S. Han *et al.*, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[2] Z. He, S. Angizi, and D. Fan, "Exploring stt-mram based in-memory computing paradigm with application of image edge extraction," in *Computer Design (ICCD), 2017 IEEE International Conference on*. IEEE, 2017, pp. 439–446.

[3] M. Courbariaux *et al.*, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.

[4] Y. LeCun *et al.*, "Mnist handwritten digit database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, vol. 2, 2010.

[5] A. Krizhevsky *et al.*, "The cifar-10 dataset," *online: http://www. cs. toronto. edu/kriz/cifar. html*, 2014.

[6] I. Hubara *et al.*, "Binarized neural networks," in *Advances in neural information processing systems*, 2016, pp. 4107–4115.

[7] J. Deng *et al.*, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[8] M. Rastegari *et al.*, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.

[9] S. Zhou, *et al.*, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[10] X. Lin *et al.*, "Towards accurate binary convolutional neural network," in *Advances in Neural Information Processing Systems*, 2017, pp. 344–352.

[11] Nvidia, "Nvidia volta gpu," March 2018, https://www.nvidia.com/en-us/data-center/volta-gpu-architecture/.

[12] G. Huang *et al.*, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, vol. 1, no. 2, 2017, p. 3.

[13] Y. Kim *et al.*, "Write-optimized reliable design of stt mram," in *Proceedings of the 2012 ACM/IEEE ISLPED*. ACM, 2012, pp. 3–8.

[14] X. Fong *et al.*, "Spin-transfer torque memories: Devices, circuits, and systems," *Proceedings of the IEEE*, vol. 104, pp. 1449–1488, 2016.

[15] Y. Bengio *et al.*, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[16] W. Tang *et al.*, "How to train a compact binary neural network with high accuracy?" 2017.

[17] S. Ioffe *et al.*, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[18] S. Angizi *et al.*, "Imce: energy-efficient bit-wise in-memory convolution engine for deep neural network," in *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*. IEEE Press, 2018, pp. 111–116.

[19] L. Yang, Z. He, and D. Fan, "A fully onchip binarized convolutional neural network fpgaimpelmentation with accurate inference," in *Proceedings of the 2018 International Symposium on Low Power Electronics and Design*. ACM, 2018.

[20] Z. He, S. Angizi, F. Parveen, and D. Fan, "Leveraging dual-mode magnetic crossbar for ultra-low energy in-memory data encryption," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*. ACM, 2017, pp. 83–88.

[21] G. Autès *et al.*, "Strong enhancement of the tunneling magnetoresistance by electron filtering in an fe/mgo/fe/gaas (001) junction," *Physical review letters*, p. 217202, 2010.

[22] Z. He and D. Fan, "A low power current-mode flash adc with spin hall effect based multi-threshold comparator," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*. ACM, 2016, pp. 314–319.

[23] S. S. Parkin *et al.*, "Magnetic domain-wall racetrack memory," *Science*, vol. 320, pp. 190–194, 2008.

[24] V. Sokalski *et al.*, "Naturally oxidized feco as a magnetic coupling layer for electrically isolated read/write paths in mlogic," *IEEE Trans. Magn.*, vol. 49, no. 7, pp. 4351–4354, 2013.

[25] Y.-H. Chen *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.