IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 19, 2019, accepted December 31, 2019, date of publication January 20, 2020, date of current version January 31, 2020.

*Digital Object Identifier 10.1109/ACCESS.2020.2968081*

# Domain Wall Memory-Based Design of Deep Neural Network Convolutional Layers

**JINIL CHUNG[1], (Student Member, IEEE), WOONG CHOI[2], (Member, IEEE), JONGSUN PARK[1], (Senior Member, IEEE), AND SWAROOP GHOSH[3], (Senior Member, IEEE)**

[1]School of Electrical Engineering Department, Korea University, Seoul 02841, South Korea
[2]School of Electronics Engineering Department, Sookmyung Women's University, Seoul 04310, South Korea
[3]School of Electrical Engineering and Computer Science, Pennsylvania State University, State College, PA 16802, USA

Corresponding author: Jongsun Park (jongsun@korea.ac.kr)

This work was supported in part by the National Research Foundation of Korea under Grant NRF-2015M3D1A1070465, in part by the Ministry of Science (MSIT) and ICT, South Korea, under the Information Technology Research Center (ITRC) under Grant IITP-2020-2018-0-01433) supervised by the Institute for Information & Communications Technology Promotion (IITP), and in part by the Industrial Strategic Technology Development Program under Grant 10077445, and in part by the Development of SoC Technology based on Spiking Neural Cell for smart mobile and IoT Devices funded by the Ministry of Trade, Industry and Energy (MOTIE, Korea).

**ABSTRACT** In the hardware implementation of deep learning algorithms such as, convolutional neural networks (CNNs) and binarized neural networks (BNNs), multiple dot products and memories for storing parameters take a significant portion of area and power consumption. In this paper, we propose a domain wall memory (DWM) based design of CNN and BNN convolutional layers. In the proposed design, the resistive cell sensing mechanism is efficiently exploited to design low-cost DWM-based cell arrays for storing parameters. The unique serial access mechanism and small footprint of DWM are also used to reduce the area and energy cost of DWM-based design for filter sliding. Simulation results with 65 nm CMOS process show 45% and 43% of energy savings compared to the conventional CNN and BNN design approach, respectively.

**INDEX TERMS** Binarized neural network, convolutional neural network, deep neural network, domain wall memory.

## I. INTRODUCTION

Convolutional Neural Networks (CNNs) are one of the well-known deep learning algorithms that have been widely employed for image classifications [1]. CNN can achieve low error rate with an acceptable complexity by combining three architectural ideas – local connections, shared weights, and spatial/temporal subsampling [2]. Deeper architecture with more layers in CNNs is the key to achieve high accuracy at the cost of complexity. In the hardware implementation of CNNs, multiple dot products for extracting features and memories for storing parameters (input activations and weights) take a significant portion of area and power consumption. Several in-depth studies on the low-cost CNN accelerator design have been carried out over the recent years. Chen *et al.* [3] proposed a CNN accelerator named Eyeriss. Systolic array-based architectures [4], [5] have also been widely adopted among various energy-efficient CNN accelerators using application specific integrated circuits (ASIC). Recently, in order to reduce the complexity of CNN, binarized neural networks (BNN) [6] have been introduced, where all the parameters and activations are represented as binary-based numbers. Using XNOR and bit-count operations (XNOR-popcount), BNN achieved comparable accuracies for MNIST, SVHN, and CIFAR-10 datasets [6]. In addition, XNOR-Net [7] and DoReFa-Net [8] have been proposed to improve the ImageNet classification accuracy of BNNs, and researches [9], [10] are currently in progress to improve those accuracies.

DWM is a flavor of spintronic memory technology that possesses the ability to store multiple bits per cell [11], [12]. The bits are stored in the magnetic nanowire in the form of magnetic orientation and can be accessed serially by performing shift operations. DWM has been proposed for energy efficient deep neural networks [13], [14]. In this work, the unique shift-based access pattern of DWM is efficiently

The associate editor coordinating the review of this manuscript and approving it for publication was Mitra Mirhassani.

exploited [15] to implement the input activations and/or weights storages of CNN and BNN convolutional layer that prefer sequential access. The resistor cell sensing circuit through read MTJ [16] is also proposed to implement a partial dot product (a XNOR-popcount) in CNN and BNN. The major contribution of this work can be summarized as follows:

- We propose a novel DWM-based cell array architecture for an area and energy-efficient CNN convolutional layer with the implementation of a partial dot product using the resistor cell sensing circuit.
- We propose a DWM-based cell array to provide low cost XNOR-popcount operations for a BNN convolutional layer.
- We propose novel DWM design techniques for implementing filter sliding in CNN and BNN convolutional layers by exploiting the sequential access pattern of DWM.

The rest of the paper is organized as follows. Section II describes the basics of DWM. The details of a convolutional layer in CNNs and the differences between CNN and BNN are presented in section III. Sections IV and V provide the DWM-based CNN and BNN convolutional layer design approaches, respectively, including implementations, and numerical results. Finally, conclusions are drawn in section VI.

## II. DWM FUNDAMENTALS

DWM consists of three components: (a) write head (b) read head and (c) magnetic Nanowire (NW). As shown in Fig. 1, the read and write heads are similar to the conventional Magnetic Tunnel Junction (MTJ) whereas NW holds the bits in the form of magnetic polarity. Since a single bit-cell can hold multiple bits in the NW, this memory technology provides high-density. Magnetic NW is the crucial component that holds the bits. In essence, the NW is analogous to a shift register. The NW typically contains physical notches to move the DW in lockstep fashion [12], [17], [18]. It also ensures that the DW does not land in between two notches. The shift pulse is enough to dislodge the DW and shift along the NW.

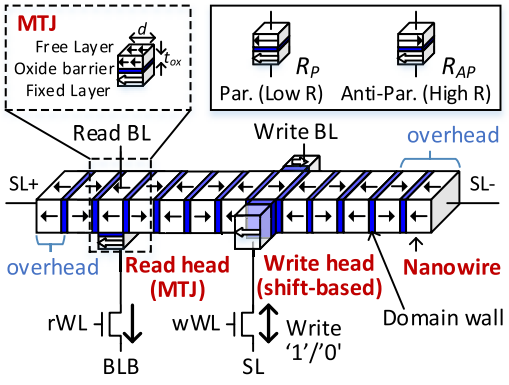



**FIGURE 1.** Schematic of the DWM using the shift-based write scheme [16]. The MTJ at read head, extra two fixed layers at write head, and the overhead bits are also shown.

Note that the MTJ forms naturally between the NW and the fixed magnetic layer that are separated by the tunnel oxide barrier. The left (right) magnetic orientation in the NW can be regarded as '0' ('1').

The most interesting feature of the NW is the formation of domain walls (DWs) between domains of opposite polarities where the local magnetization changes its polarity. The DWs can be shifted forward and backward by injecting charge current from left-shift (SL+) and right-shift (SL-) contacts. The new bits are written by first pushing current through shift contacts to move the bits in lockstep fashion to bring the desired bit under write head. Then, spin polarized current is injected through two extra fixed layers in write head (using Write BL and SL) in positive or negative direction to write a '1' or '0' in the NW. Note that conventional write scheme uses write MTJ instead of extra fixed layers. The writing involves current induced spin-torque transfer to flip the magnetization of the free layer (NW in this case). The bits are shifted back to initial state after the write operation. Read is performed by bringing the desired bit under the read head using shift and sensing the resistance of MTJ formed by DW under the read head (using Read BL and BLB in Fig. 1).

It should be noted that the resistance of MTJ is high (presented by $R_{AP}$) when the fixed layer and the free layer are in antiparallel configuration whereas the resistance is low (presented by $R_P$) when they are parallel to each other (Fig. 1). The bits are shifted back to initial state after the read operation. From the above discussion, it can also be concluded that the read/write involves shifting of bits. For random access, the worst case latency is the summation of number of shifts and read/write latency. However, for serial access, the latency is a summation of single shift and read/write latency.

The conventional DWM has high power consumption for write operation. In this work, a shift-based write scheme is introduced in [17] which uses extra fixed layers to shift the bits in the NW instead of nucleating it through MTJ. The resulting write operation is significantly low-power and fast at the cost of area overhead ($24.7F^2$ per bit vs $2.56F^2$ [17]). The latency, power, and area of DWM are shown in Table 1. The 1D DWM model [19], [20] is used for the latency and power estimation.

**TABLE 1. Characteristics of DWM.**

| Operation | Speed | Power | Footprint |
|---|---|---|---|
| Read | 2.81 ns | 23.08 μW | $2.56F^2$ per domain (4 heads) |
| Write | 3.9 ns | 55 μW | |
| Shift | 1 ns-2 ns | 10 μW | |
| Shift-based write | 1 ns-2 ns | 28.4 μW | $24.7F^2$ per bit |

## III. CNN AND BNN OVERVIEW

In this section, we first introduce the simple CNN and BNN architectures and their comparison. We also discuss the implementation of CNN and BNN convolutional layers based on 2D processing element (PE) array.

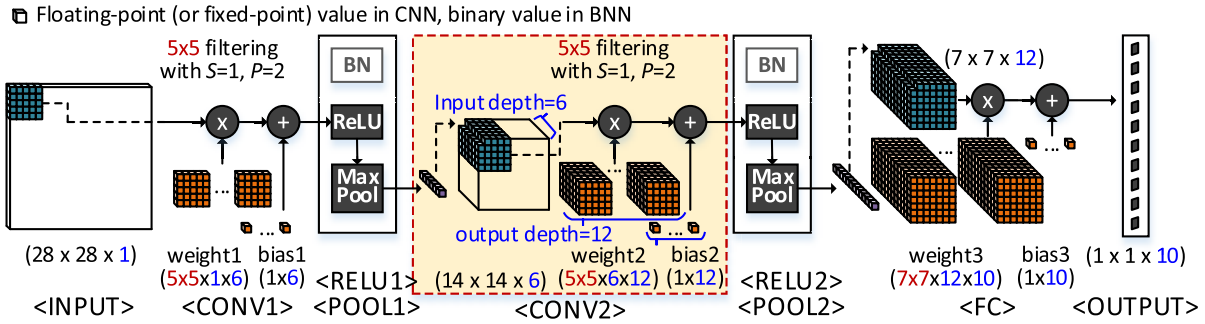



**FIGURE 2.** A simple convolutional neural network (CNN) with three layers for MNIST (28 × 28 × 1) classification.

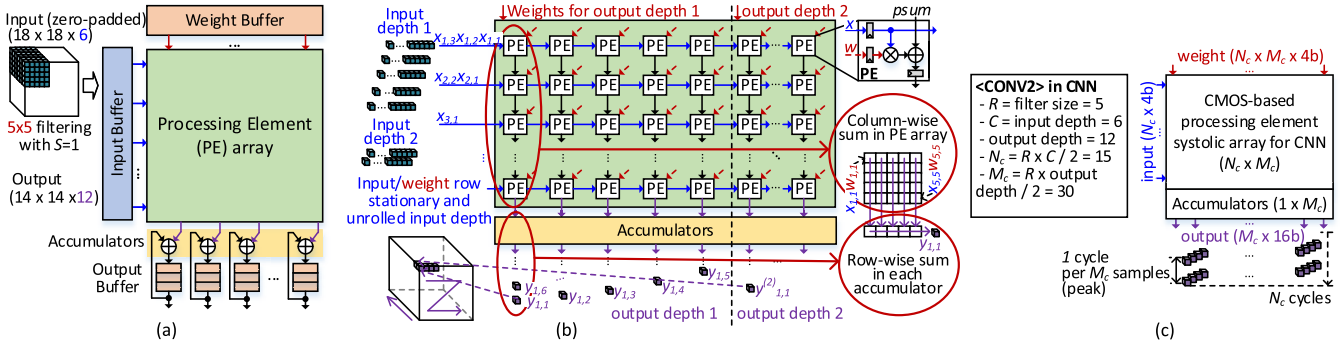



**FIGURE 3.** (a) The hardware architecture of the conventional CNN convolutional layer with (b) PE array. (c) The CONV2 layer example.

### *A. CNN ARCHITECTURE*

CNNs are similar to regular neural networks (NNs) since several hidden layers are composed of a set of neurons where each neuron in the output layer is connected to the neurons in the input layer. While neurons are fully connected in regular NNs, CNNs have local connections by explicitly assuming that the input activations will have spatial locality such as images. The layers of CNNs, unlike regular NNs, have neurons arranged in three dimensions: width, height, and depth. Typical CNNs have six types of layers, shown in Fig. 2: INPUT, CONV (convolutional), BN (batch normalization), RELU (rectified-linear units), POOL (pooling), and FC (fully-connected) layer.

#### 1) CNN CONVOLUTIONAL LAYER DESIGN

Convolutional layer is the core building block of CNNs, which is locally connected to the input volume. The output volume of the convolutional layer can be characterized by three hyper-parameters [21]: (a) The depth of the output volume is the number of filters with the same region of the input volume; (b) The stride is the spatial distance between current and next filtering region; and, (c) With sizing zero-padding, the spatial size of the output volume can be controlled, and the spatial size of the input volume can also be preserved. Fig.2 shows an example of a CNN with three layers for MNIST classification. In the CNN, CONV, POOL, and ReLU layers are performed twice with input image (28 × 28 × 1), followed by the FC layer for MNIST classification. As specified in the CONV2 layer of Fig. 2, a 5 × 5 dot product (or filtering) is computed with output depth 12, stride (=$S$) 1, and zero-padding (=$P$) 2.

The overall hardware architecture of the conventional CNN convolutional layer specified in Fig. 2, is presented in Fig. 3. The PE systolic array that contains PE units in 2D form, and each PE unit compute a partial sum with an input activation and a weight is illustrated in Fig. 3(b). For the computation, weights are preloaded into the PE array from weight buffer and remain stationary until all the related input activations are computed. For the systolic operation, scheduled input activations are streamed into the PE array from the input buffer, and PE outputs (partial sums) are accumulated sequentially through the bottom direction in 2D array. As shown in Fig. 3(c), the number of PE units in the 2D PE array is $N_c \times M_c$, and the number of accumulators is $1 \times M_c$. In Fig. 3(c) (leftmost), $N_c$ (the number of PE units in a column of the PE array) shows the filter size $R$ multiplied by input depth and divided by 2, and $M_c$ (the number of PE units in a row of the PE array) is the filter size $R$ multiplied by output depth and divided by 2. Here, to make the proposed PE array have similar core size with the recent CNN accelerator (32 × 16 [22]) the 'division by 2' is used for $M_c$ and $N_c$. For example, for CONV2 layer, the convolutional layer design includes 15 × 30 (=$N_c \times M_c$) PE units and 30 (=$1 \times M_c$)

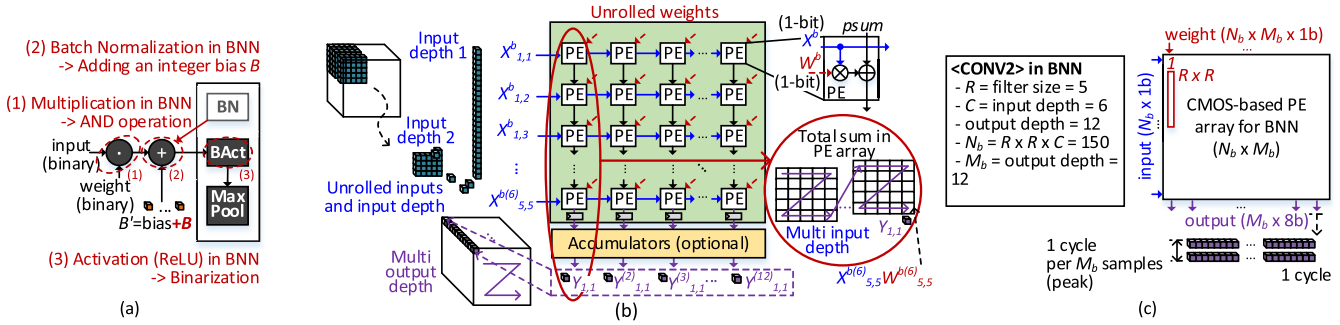



**FIGURE 4.** (a) Differences of the CNN layer operations between CNN and BNN. (b) The architecture that implements the BNN convolutional layer using 2D PE array. (c) The CONV2 layer example.

accumulators, where the filter size $R$/ input depth/ output depth of CONV2 layer are 5/ 6/ 12, respectively. The peak throughput per sample of the convolutional layer can be calculated as $M_c/T_{clk}$ ($T_{clk}$ is a clock cycle time), while the output latency is $N_c$.

### B. BNN ARCHITECTURE

BNNs have a layer structure like CNNs, and the only differences are the binary parameters (input activations and weights). As shown in Fig. 4(a), the layer operation in BNN is different from that of CNN as follows: (1) a multiplication in BNN is a bitwise AND operation, (2) a batch normalization in BNN is just adding integer bias $B$ [23], and (3) an activation operation (e.g. ReLU) in BNN corresponds to binarization.

#### 1) BNN CONVOLUTIONAL LAYER DESIGN

For the hardware implementation of BNN convolutional layer design, a 2D PE array can also be used as presented in Fig. 4(b). However, the area of the PE unit can be significantly reduced, since the multiplications of a binary input activation and a binary weight can be replaced with bitwise AND operations. Due to the smaller area of PE units and the reduced buffer sizes for storing binary parameters (input activation and weight), the fully parallel architecture [24] is usually adopted. In this architecture, both input activations and weights are preloaded to compute the output activation within one clock.

Fig. 4(c) shows the hardware architecture that implements the BNN CONV2 layer. The PE array for BNN consists of $N_b \times M_b (=150 \times 12)$ PE units, where $N_b$ means the input depth multiplied by $R^2$, and $M_b$ denotes the output depth. Compared to CNN convolutional layer, since DFFs are not needed for the BNN PE unit, the accumulation size $N_b$ in column direction decides critical path delay. The accumulation can be performed with optimal adder-tree such as carry save adders [25]. In the BNN convolutional layer, the peak throughput per sample is $M_b/T_{clk}$ with a maximum output latency of one clock cycle.

#### 2) XNOR-POPCOUNT IN BNN CONVOLUTIONAL LAYER

The XNOR-popcount in BNN convolutional layer is expressed as follows:

$$f(Y) = f\left(\sum_{i=1}^{N_b} X_i^b W_i^b + B_{\{0,1\}}\right) = \begin{cases} 1, & if\ Y \geq N_b/2 \\ 0, & else \end{cases} \quad (1)$$

where $Y$ is non-negative output activation, $B_{\{0,1\}}$ is the half of $B$ (bias), and $X_i^b$, $W_i^b \in \{0, 1\}$ and $N_b$ means binarized input activation, binarized filter weight, and accumulation size, respectively (i.e. $N_b = R \times R \times C = 5 \times 5 \times 6 = 150$ as shown in CONV2 layer of Fig. 4(c)). (1) means that the binarization function outputs 1 when the accumulated result is larger than $N_b/2 + B_{\{0,1\}}$, and otherwise the output is 0.

### C. THE CONV2 DESIGN COMPARISON IN CNN AND BNN

Fig. 5 show the comparison results of four different modules in CONV2 layer of CNN and BNN accelerators. Those four modules are CORE (PE array + accumulator), IBUF (SRAM based input buffer), WBUF (SRAM based weight buffer), and WREG (DFF based weight registers). The four modules are implemented using 65nm CMOS process, and the results

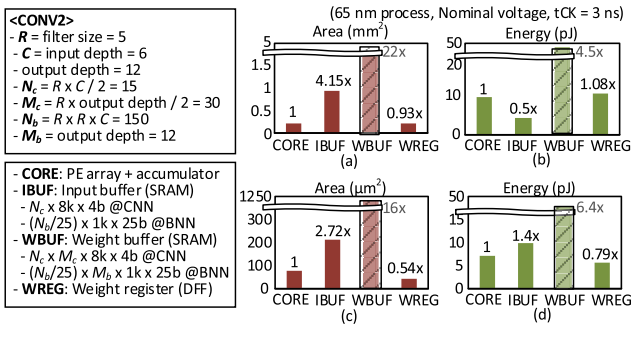



**FIGURE 5.** Comparison results (implemented using 65nm CMOS process) of CORE, IBUF, WBUF, and WREG modules in CNN design in terms of (a) area and (b) energy consumption, and in BNN design in terms of (c) area and (d) energy consumption.

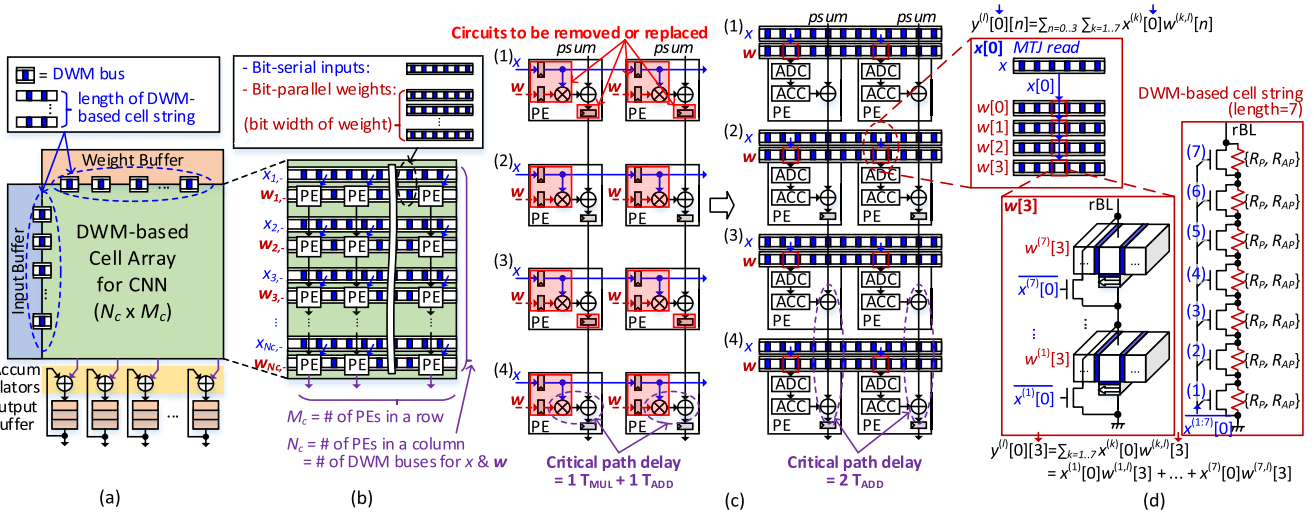



**FIGURE 6.** Architecture of (a) the proposed DWM-based CNN convolutional layer with (b) the DWM-based cell array for CNN, which consists of (c) the DWM-based PEs including (d) the DWM-based cell string (length = 7).

are compared in terms of the area and energy consumption. As shown in Fig. 5, the WBUF takes the largest area and energy consumption since WBUF storing the weight parameters is shared by all the PE units in 2D PE array. WREG is used to store partial weight parameters (in this example, 2*R* weights in each weight register of CNN and 2 weights in each weight register of BNN), and it is locally connected with PE unit in 2D PE array. We can notice from the results that not only the PE array (CORE in Fig. 5) but also the input buffers (IBUF) and weight registers (WREG) should be carefully designed to make an area and energy efficient CNN architecture. In the following sections, we show those expensive WBUF, WREG and IBUF can be efficiently designed using low cost DWM-based in-memory computing.

## IV. DWM BASED CNN CONVOLUTIONAL LAYER DESIGN

In this section, we present the architectural/circuit-level design techniques for the proposed DWM-based CNN convolutional layer design. By exploiting the unique serial access mechanism of DWM, the sequence of input activations and weights in PE array can be efficiently implemented using DWM. In addition, the partial implementation of dot product using the resistive cell sensing of DWM leads to low cost design of CNN convolutional layer.

### A. DWM-BASED CNN CONVOLUTIONAL LAYER

The architecture of the proposed DWM-based CNN convolutional layer is illustrated in Fig. 6(a). It consists of input buffers, weight buffers, DWM-based cell array for CNN, accumulators, and output buffers. In the DWM-based design, input and weight buffers are connected to the cell array through the DWM buses. In Fig. 6(b), the DWM-based cell array is illustrated. As shown in the figures, the DWM-based CNN convolutional layer has similar architecture and operations to the conventional CMOS-based architecture (Fig. 3(b)). The only exception is the bit-serial operation which is implemented using the DWM-based PE units composed of the DWM-based cell string, 3-bit ADC, and the accumulator as shown in Fig. 6(c)-(d) and Fig. 7. By using the DWM shift operation, the convolutional window sliding and serial multiplications can be effectively implemented in the proposed DWM based cell array. For the convolutional window sliding, DWM shift based convolution operations in the first row between multiple inputs and multiple weights are presented in section IV-C. In addition, DWM shift based serial multiplication between one input and one weight is shown in Fig. 7.

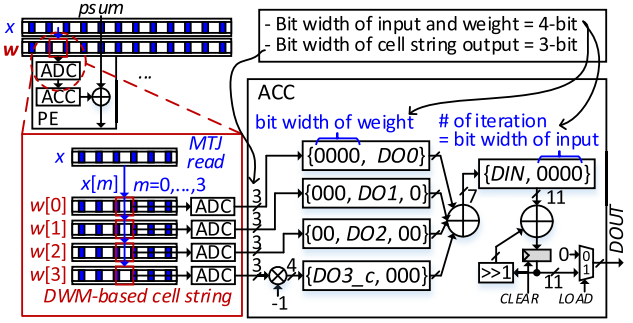



**FIGURE 7.** The accumulator in PE unit in the DWM-based design.

Fig. 6(c) shows the differences of the DWM based PE units compared to the conventional CMOS-based design. First, the DFFs (for storing input activations and weights) and the multiplier in the conventional PE unit are replaced by the DWM-based cell string, 3-bit ADC, and the accumulator in the DWM-based PE unit. As shown in the DWM based design of Fig. 6(c), the registers for storing a partial sum (*psum*) that placed in odd rows are removed. This configuration facilitates to accumulate two rows *psum* while reducing the critical path delay and PE area. The row-wise input timing control

depends on the *psum* register. For example, on the left side of Fig. 6(c), the input of the 1st (3rd) row is faster than the input of the 2nd (4th) row by one latency, but both have the same timing on the right side of Fig. 6(c). The detailed architecture of the PE unit (bit width of input activation and weight = 4-bit) is shown in Fig. 6(d) and Fig. 7. For a bit count operation, bit-serial input (e.g. $x[0]$) comes from the DWM input buses, and it performs the operations using the DWM-based cell string with bit-parallel weights (e.g. $w[3:0]$) that are stored in DWM weight buses. The partial product can be obtained by summing the bit count results of each weight index. Thus, the serial multiplication operations are replaced by accumulating the partial products of each input index. For accumulation outside the cell array, like conventional CMOS architecture, the initial value of accumulators is the bias value, and the outputs of cell arrays are accumulated if the input depth is larger than the height of the cell array. The DWM-based cell string and 3-bit ADC will be discussed in section IV-B, and the DWM input and weight buses for input activations and weights, respectively, are shown in section IV-C.

### B. DWM-BASED CELL ARRAY FOR CNN

In the CMOS-based PE systolic array architecture shown in Fig. 3(b), the cell array and dot product operators are separately implemented. In the proposed DWM-based architecture (Fig. 6), the partial dot products can be merged together and performed in the cell arrays. The DWM-based cell array consists of PE array and DWM input and weight buses. The current reference circuit for sensing the outputs of the DWM-based cell string are located inside the cell array.

#### 1) DWM-BASED CELL STRING

The DWM-based cell array architecture shown in Fig. 6 presents the implementation of the convolutional layer using $5 \times 5$ filter with the input depth of 6 and the output depth of 12. The operation can be considered as parallel multiplications and accumulations on PE array (bit width of each weight is 4 and input bit width is 4). The convolutional layer operation is expressed as

$$y^{(l)} = \sum_{r=1}^{5} \sum_{c=1}^{5} y_{r,c}^{(l)} \tag{2}$$

$$y_{r,c}^{(l)} = \sum_{m=0}^{3} y_{r,c}^{(l)}[m] \cdot 2^m \tag{3}$$

$$y_{r,c}^{(l)}[m] = -y_{r,c}^{(l)}[m][3] + \sum_{n=0}^{2} y_{r,c}^{(l)}[m][n] \cdot 2^{n-3} \tag{4}$$

$$y_{r,c}^{(l)}[m][n] = \sum_{k=1}^{7} x^{(k)}[m] \cdot w^{(k,l)}[n] \tag{5}$$

where $l$ = output depth index, $k$ = input depth index, $m$ = the bit index of input, $n$ = the bit index of weight, and $w$ = weight (in this example, input activations and weights when $k = 7$, $x^{(7)}[m]$ and $w^{(7,l)}[n]$, are zero since the input depth is 6). In Fig. 6(d), the boxes show an example ($m = 0$) of $y_{r,c}^{(l)}[m][n] = \sum_{m=0}^{3} \sum_{n=0}^{3} \sum_{k=1}^{7} x^{(k)}[m] \cdot w^{(k,l)}[n]$ with bit-serial operations, where $m$ varies from 0 to 3 depending on bit-serial index, while $n$ stretches from 0 to 3 in bit-parallel fashion. The operation in the boxes at the bottom ($m = 0$ and $n = 3$) is $x^{(1)}[0] \cdot w^{(1,l)}[3] + x^{(2)}[0] \cdot w^{(2,l)}[3] + x^{(3)}[0] \cdot w^{(3,l)}[3] + x^{(4)}[0] \cdot w^{(4,l)}[3] + x^{(5)}[0] \cdot w^{(5,l)}[3] + x^{(6)}[0] \cdot w^{(6,l)}[3] + x^{(7)}[0] \cdot w^{(7,l)}[3]$. As shown in Fig. 6(d), each of $w^{(k,l)}[3]$ is stored in MTJ, and $x^{(k)}[0]$ is applied to the gate input of the selective transistor. The output of $\sum_{k=1}^{7} x^{(k)}[0] \cdot w^{(k,l)}[3]$ can be represented as the series of resistors (referred as the DWM based cell string in Fig. 6(d)), where the resistance values are dependent on the input activation $x^{(k)}[0]$ and weight $w^{(k,l)}[3]$.

As shown in Fig. 6(d), the DWM-based cell string is composed of serial connection of selective transistor and MTJ pairs. Since an MTJ cell can be modeled as $R_P$ ('0' in logic) or $R_{AP}$ ('1' in logic) resistor, and the selective transistor has $R_{on}$ or open state depending on the gate input $x^{(k)}[m]$, each pair can be modeled as a resistor with four values, which are $R_P$, $R_{AP}$, $(R_{on}||\ R_P)$, and $(R_{on}||R_{AP})$. By assuming that $R_{on} < R_P < R_{AP}$ and $(R_{on}||R_P) \approx (R_{on}||R_{AP})$ in this work, each pair can be expressed as a resistor with three variable resistances, $R_{AP} > R_P > R_S$ where $R_S = R_{on}\ ||R_P$ (or $R_{AP}$). Here, the selective transistor performs a masking operation for the MTJ value based on the gate input. The logic value of the pair is same as AND gate output between the inverted input and the bit stored in MTJ (=the bit information of weights). The DWM-based cell string accumulates the resistance values of these pairs, and the accumulated resistance value is same as the number of '1's in terms of logical value. As a result, DWM-based cell string for 7 input depth performs the operations in (5).

#### 2) VOLTAGE REFERENCE CIRCUIT

As presented in the previous subsection, the resistance of the DWM-based cell string varies depending on the input activation $x^{(k)}[m]$ and weight $w^{(k,l)}[n]$. The varying resistance value will appear as the changes of the voltages on 'read BL (referred as rBL)' in Fig. 6(d). Therefore, we need to generate the reference signals for sensing the read BL voltages. The references can be expressed as the following resistance values:

$$\begin{aligned} R_{BL(0)} &< R_{REF(0)} \\ R_{REF(i)} &< R_{BL(i+1)} < R_{REF(i+1)} \\ R_{REF(N-1)} &< R_{BL(N)} \end{aligned} \tag{6}$$

where $i = 0 \ldots N-2$. The voltage reference circuit in the left side of Fig. 8(a) is composed of input-independent resistor ladder and input-dependent DWM-based cell string consisting only of MTJ cells in the parallel state ($R_P$ represented in the resistance value), which can compensate the $Rs$ terms applied to the voltage on read BL.

#### 3) 3-BIT FLASH TYPE ADC

The read BL (rBL) and the generated reference signals go into the ADC to compute a partial dot product operation. Fig. 8(a) shows the 3-bit flash type ADC with the voltage reference circuit. In the proposed scheme, typical flash ADC can be

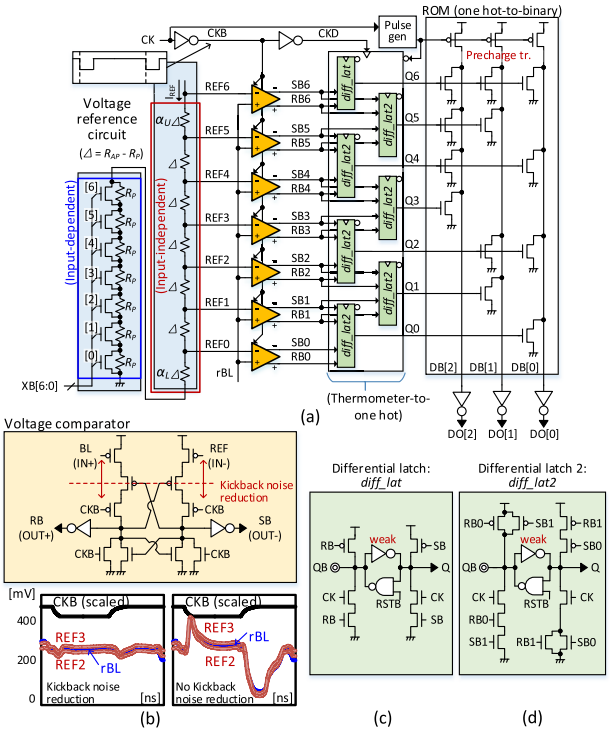



**FIGURE 8.** Schematics of (a) voltage reference and 3-bit ADC with (b) voltage comparator (including the kickback noise simulation), (c) diff_lat, and (d) diff_lat2.

divided into three stages: (a) in the first step, thermometer code is generated using comparators; (b) thermometer code is converted to one-hot code using the gates logic (e.g. AND-NOT gate); and, in the third step, (c) the outputs of the gates are decoded to binary outputs. For the thermometer code generation, we used the voltage comparator with low kickback noise in Fig. 8(b) [26] to ensure the voltage margin as shown on the bottom side of the figure. For the thermometer-to-one hot conversion, the modified CMOS differential latch (*diff_lat*2) in Fig. 8(d) is used that is made with only four additional transistors based on differential latch (*diff_lat* or N-C$^2$MOS latch [27]) shown in Fig. 8(c). Finally, the conventional ROM-based design is employed to obtain binary outputs from one-hot code.

#### 4) DATA DEPENDENCY AND OPTIMIZATION OF DWM-BASED CELL STRING

Fig. 9 shows two input cases for the DWM-based cell string. Although both cell strings are intended to have the same resistance of $3R_{AP}$ for those two input cases, the actual voltage difference ($\Delta_{BL}$) between two read BLs is +9 mV according to our simulations. In addition, the voltage difference ($\Delta_{BL}$) is larger than the reference voltage difference between two DWM-based cell strings, which is $\Delta_{REF}$ of +3 mV. The simulation results show that the reference voltage cannot completely track the read BL of MTJ cells. This is because the $R_{on}$ varies depending on the gate input pattern, and the DWM-based cell string in the voltage reference circuit has

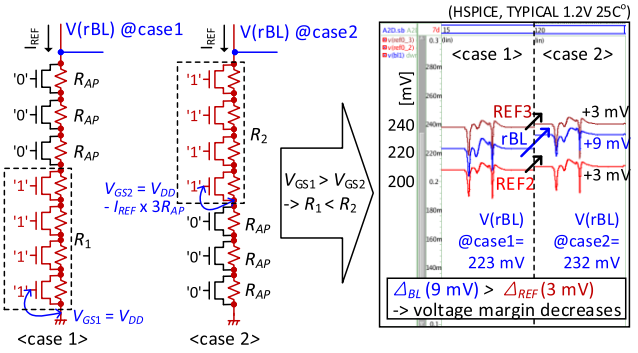



**FIGURE 9.** The data dependency of DWM-based cell string due to stacking effect.

only $R_P$ of MTJ cells while the DWM-based cell sub-array has both $R_P$ and $R_{AP}$. This problem is even more exacerbated with long length of DWM-based cell string. According to our simulation results on the voltage margin and the bit width of binary output (ADC overhead), the length of DWM-based cell string is decided to 7.

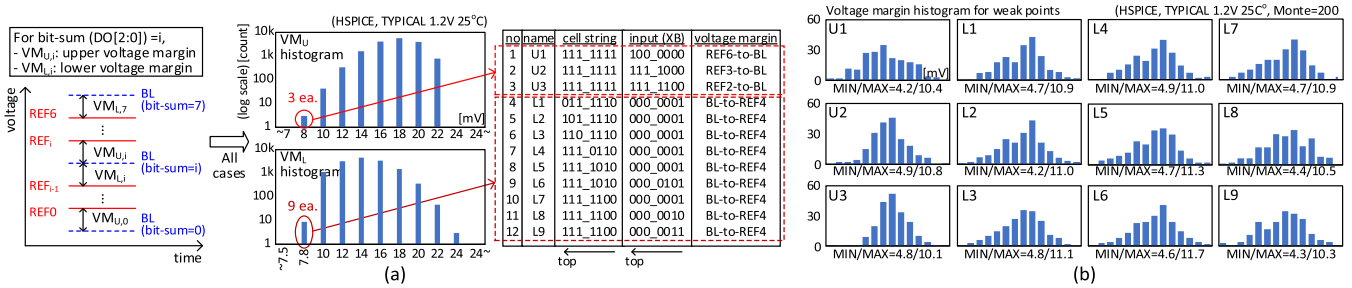
In the proposed DWM-based cell string, to maximize the voltage margin, the scale factor $\alpha_U$ ($\alpha_L$) for the resistor located at the top (bottom) of the input-independent resistor ladder in the voltage reference circuit is tuned to 0.75 (0.75) according to our simulations. To verify the operation reliability, HSPICE Monte-Carlo simulations are performed for the worst-case conditions. As shown in Fig. 10(a), voltage margin of the DWM-based cell string relies on the gate input pattern and the cell states. For the upper voltage margin and the lower voltage margin, 3 worst combinations and 9 worst combinations of the gate input pattern and the cell states are selected, respectively. The simulation results for those worst-case conditions are presented in Fig. 10(b). The negative voltage margin which indicates the functional failure is not observed.

### C. DWM INPUT AND WEIGHT BUS

In the CMOS-based PE systolic array architecture shown in Fig. 3(b), the input activations for each row pass from a DFF's in a PE unit to the next stage PE unit, while the weights are sent from the weight buffer to PE following data flow. In the proposed DWM-based architecture as shown in Fig. 6, the serial access of input activations and weights in PE array can be implemented using DWM bus, which are the collection of DWMs of the same length. The number of the DWMs in the DWM bus is designed as the length of the DWM-based cell string (=7).

#### 1) DESIGN ISSUE OF DWM BUS AS INPUT AND WEIGHT BUFFERS

Fig. 11(a) illustrates the design issues of the DWM input and weight bus when those are used as input activation buffers and weights buffers, respectively. As presented in Fig. 11(a), on the first row in PE array, we have three PE units.




| | noname | cell string | input (XB) | voltage margin |
|---|---|---|---|---|
| 1 | U1 | 111_1111 | 100_0000 | REF6-to-BL |
| 2 | U2 | 111_1111 | 111_1000 | REF3-to-BL |
| 3 | U3 | 111_1111 | 111_1100 | REF2-to-BL |
| 4 | L1 | 011_1110 | 000_0001 | BL-to-REF4 |
| 5 | L2 | 101_1110 | 000_0001 | BL-to-REF4 |
| 6 | L3 | 110_1110 | 000_0001 | BL-to-REF4 |
| 7 | L4 | 111_0110 | 000_0001 | BL-to-REF4 |
| 8 | L5 | 111_1010 | 000_0001 | BL-to-REF4 |
| 9 | L6 | 111_1010 | 000_0101 | BL-to-REF4 |
| 10 | L7 | 111_1100 | 000_0001 | BL-to-REF4 |
| 11 | L8 | 111_1100 | 000_0010 | BL-to-REF4 |
| 12 | L9 | 111_1100 | 000_0011 | BL-to-REF4 |

**FIGURE 10.** (a) The process of finding weak voltage margin points and (b) the result of Monte-Carlo analysis for those points of DWM-based cell string.

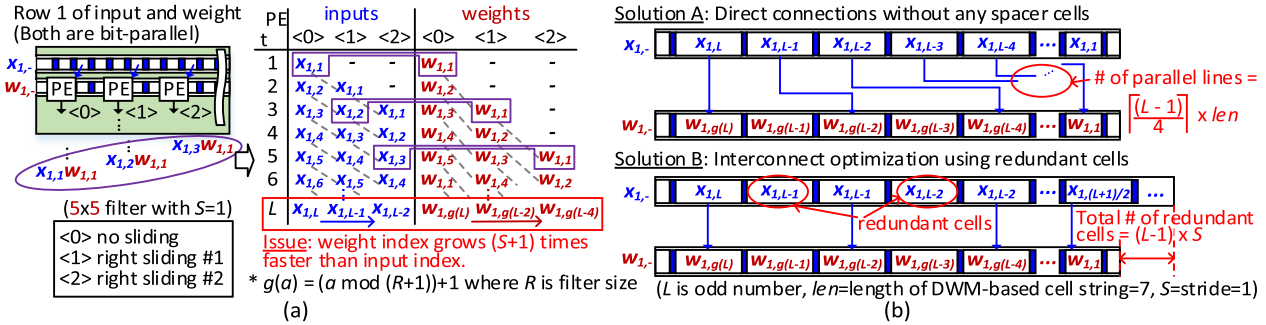



**FIGURE 11.** The proposed DWM-based cell array for CNN operation: (a) Design issue of the DWM input and weight bus, and (b) two solutions to those issues.

PE<0> creates a partial sum for the first row (e.g. $x_{1,1}w_{1,1}$) when the filtered window is not sliding yet. PE<1> generates a partial sum for the first row (e.g. $x_{1,2}w_{1,1}$) after the filtered window slides right once. PE<2> generates the outputs of a partial sum for the first row (e.g. $x_{1,3}w_{1,1}$) when the filtered window slides right twice. The time table of PEs in Fig. 11(a) shows the index changes of input activations and weights for each PE. According to the table, PE<0>, PE<1>, and PE<2> continuously generate outputs (partial sums) after 1 cycle, 3 cycles, and 5 cycles, respectively. For the input activations and weights from PE<0> to PE<2> at cycle $L$, the input sequence appears from $x_{1,L}$ to $x_{1,L-2}$. On the other hand, the weight sequence appears from $w_{1,g(L)}$ to $w_{1,g(L-4)}$ where $g(a) = (a \bmod (R+1)) + 1$ and $R$ is filter size, which differs by four samples. This means that the weight index increases two (=$stride+1 = 2$) times faster than input index, which makes it difficult to physically place the two (input activation and weight) DWM buses side by side.

As presented in Fig. 11(b), one of the possible solutions is the direct connections without any spacer cells. However, this approach has significant interconnection congestions as shown in Fig. 11(b). For example, if the PE cell array size is 15 × 30 and the length of the DWM-based cell string is 7, the number of interconnect lines is more than 750 (=ceil((30-1)/4) × 7 × 15). Therefore, we used the interconnect optimization scheme by employing redundant DWM cells. The additional DWM cells do not incur large area overhead as the DWM cell has small footprint, which will be presented in the next subsection.

### 2) OPERATION OF DWM INPUT AND WEIGHT BUS

Fig. 12 presents the serial multiplications between input activations and weights based on index-time chart and DWM bus update diagram. The chart shows the input activation and weight index at each PE over time, and the DWM bus update diagram shows the overall values of the DWM buses.

Fig. 12(a) shows the schedule of the serial multiplication between $x_{1,1}$ and $w_{1,1}$ at PE<0>, where the input activations come in bit-serial with the bit width of 4, while the weights come in bit-parallel with the bit width of 4. As shown in the index-time chart, the DWM weight bus should be doubled to match the update rates between input activations and weights. This means that $w_{1,1}$ is stored in two consecutive DWM addresses. The serial multiplication process is as follows: i) cycle 1: for the 0-th bit position of $x_{1,1}$, PE<0> computes $x_{1,1}[0]\,w_{1,1}$, ii) cycle 2: with the first bit position of $x_{1,1}$, PE<0> computes $x_{1,1}[1]\,w_{1,1}$, iii) cycle 3: for the second bit position of $x_{1,1}$, PE<0> computes $x_{1,1}[2]\,w_{1,1}$, and iv) cycle 4: with the third bit position of $w_{1,1}$, PE<0> computes $x_{1,1}[3]\,w_{1,1}$ and obtains $x_{1,1}w_{1,1}$. As a result, after 4 cycles, DWM input bus and weight bus are shifted by 4 and 2 cells, respectively, to match the update rates. The output latency for a partial sum generation is equal to the bit width of input activation, assuming that stride is 1 and bit width is an even number.

Fig. 12(b) shows the time schedule of the serial multiplications between input activations and weights in PE<0>, PE<1>, and PE<2>. In the index-time chart, where the top left corner is same as Fig. 12(a), the index of PE<0> is

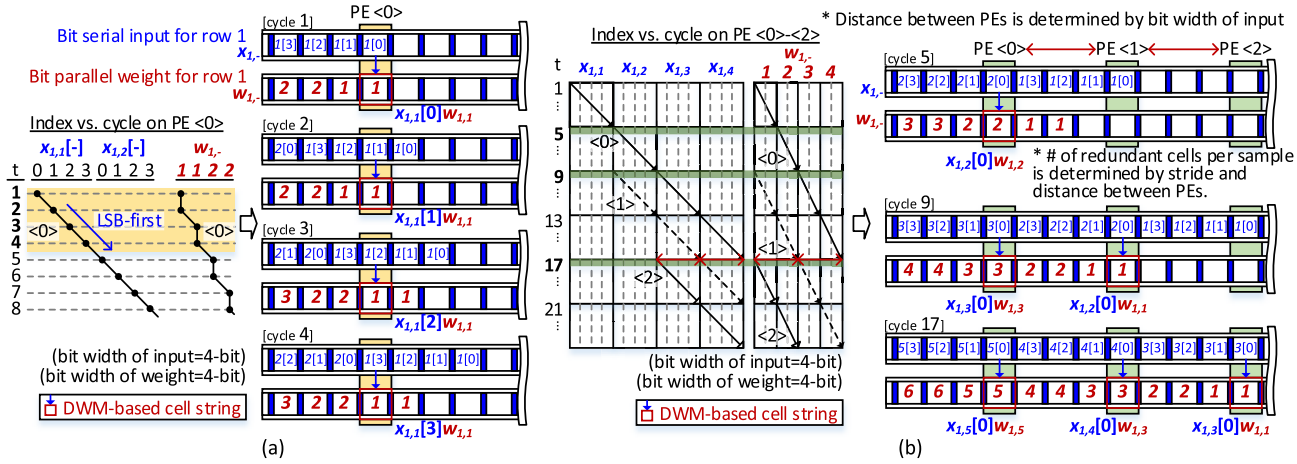



**FIGURE 12.** The proposed DWM-based cell array for CNN operation: index vs. cycle chart and DWM bus update diagram for (a) single PE and (b) multiple PEs.

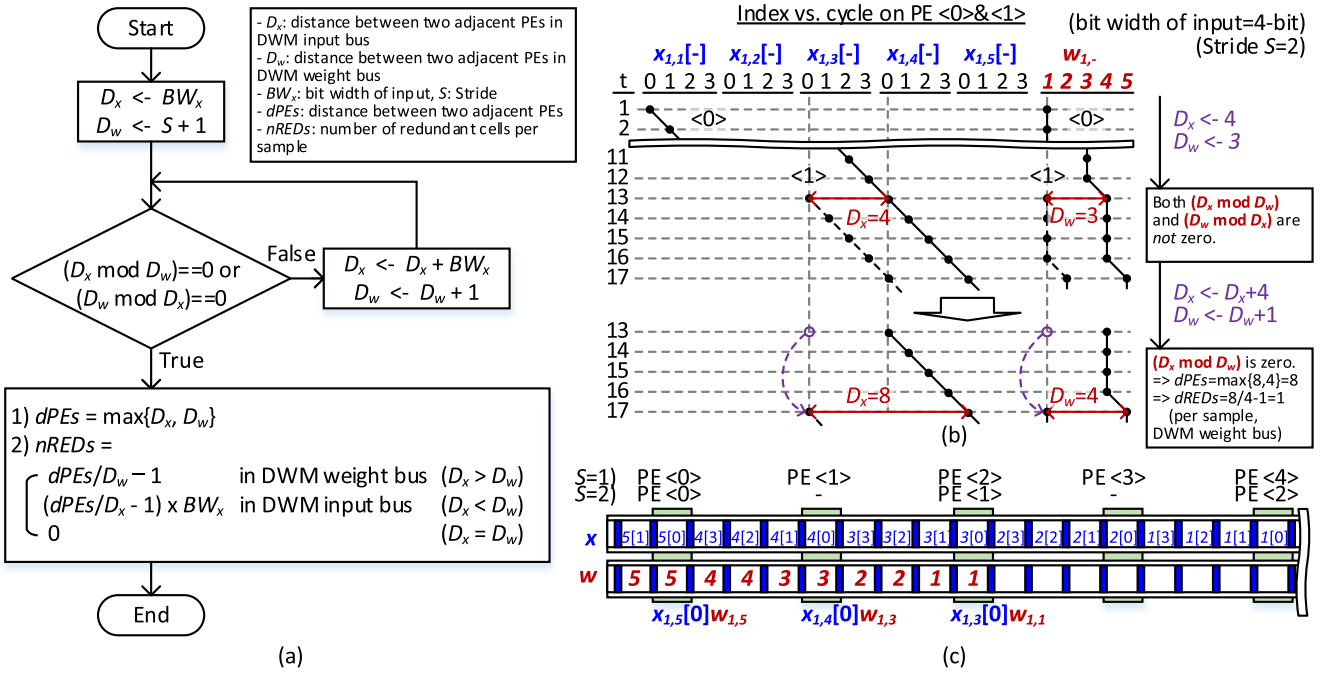



**FIGURE 13.** The proposed DWM-based cell array for CNN operation: (a) flowchart for finding *dPEs* (the distance between two adjacent PEs) and *nREDs* (the number of redundant cells per sample), (b) index vs. cycle chart for $BW_X$ (the bit width of input) = 4 and $S$ (stride) = 2, and (c) the DWM input and weight bus example of $S = 1$ and 2 ($BW_x = 4$).

drawn with a solid line with black arrows, that of PE<1> is a dotted line with black arrows, and that of PE<2> is a solid line with white arrows. During the simultaneous operation of the three PEs, the spacing between charts is kept constant since the spacing between the PEs on the DWM bus is fixed. The proposed DWM-based convolutional layer design shown in Fig. 12 shows the throughput of *the bit width of the input activations* times slower compared to the CMOS-based design. However, the throughput can be made same when the DWM-based cell string in the DWM-based cell array operates in parallel.

### 3) GENERALIZATION OF DWM INPUT AND WEIGHT BUS

In the proposed DWM-based cell array for CNN operation, the distance between two adjacent PEs and the number of redundant cells per sample are determined by the bit width of input sample and stride. For the serial multiplication, the distance between two adjacent PEs in DWM input bus ($D_x$) is initialized to the bit width of input sample ($BW_x$), while the distance between two adjacent PEs in DWM weight bus ($D_w$) is initialized to 'stride ($S$) + 1'. As shown in the flowchart of Fig. 13(a), when $D_x$ can be divisible by $D_w$ (or $D_w$ can be divisible by $D_x$), the two (input activation and weight)

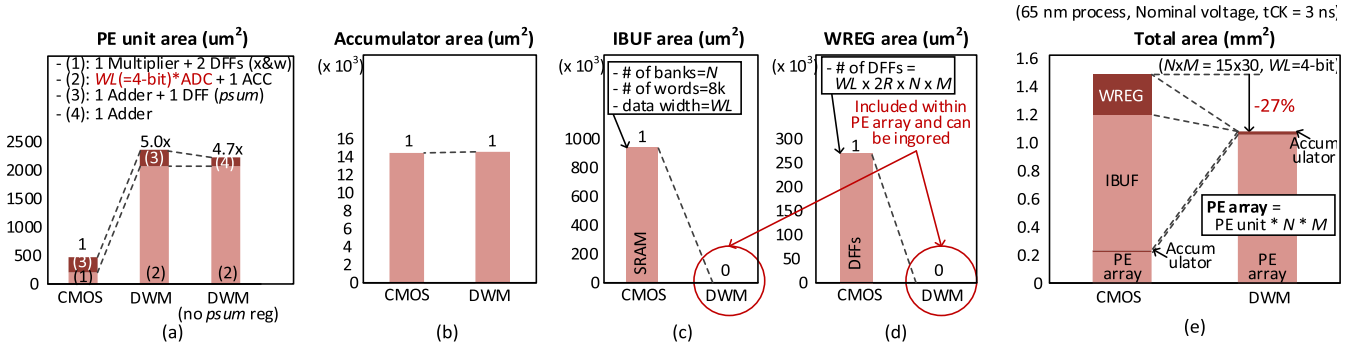



**FIGURE 14.** Area breakdown of the comparison between the conventional CMOS-based and the proposed DWM-based CNN convolutional layer implementations – (a) PE unit, (b) accumulator, (c) IBUF, (d) WREG, and (e) total (the PE array size $N \times M$ is 15 × 30 and the bit width of input activations and weights is 4-bit).

DWM buses, which include redundant cell per sample, can be physically placed side by side. On the other hand, when $D_x$ cannot be divisible by $D_w$ (or $D_w$ cannot be divisible by $D_x$), since the input activations come in bit-serial, if the weights come in bit-parallel, $BW_x$ and 1 are added to $D_x$ and $D_w$, respectively, until $D_x$ is divisible by $D_w$ or vice versa. At this time, to align the input and weight in DWM bus, the distance between two adjacent PEs (*dPEs*) is selected as maximum number between $D_x$ and $D_w$. For the *dPEs*, the number of redundant cells per sample (*nREDs*) can be computed as shown in Fig. 13(a). Fig. 13(b) shows an example of when $BW_x = 4$ and $S = 2$, and Fig. 13(c) shows an example of the DWM input and weight bus when $S = 1$ and 2. As shown in the figure, when the stride changes from 1 to 2, the number of redundant cells is fixed to 1, while the distance between two adjacent PEs (*dPEs*) is doubled. Although large stride or odd number of input feature bit-width can degrade the effectiveness of the proposed DWM-based cell array, since stride is generally less than 3 [21] and the 4-bit input and weight configurations can present the full-precision accuracy [28], [29] for recent CNNs, the proposed systolic DWM-based array is effectively reconfigurable for the different layer configuration.

### D. CNN IMPLEMENTATION RESULTS

DWM-based cell arrays for CNN have been implemented and simulations are performed using 65 nm CMOS standard cell library. Using the liberty file (.lib) format, we also generate a DWM cell library through Synopsys for delay, power and area estimation. The delay of DWM has been obtained based on bitcell delay, interconnect resistances and capacitance estimated for 65nm process. The large DWM (shift-based write) footprint of 24.75 $F^2$/bit is used in our simulations. For the comparisons of power dissipation in architectural/circuit-level, PrimeTime-PX and HSPICE are used in the simulations with TYPICAL 1.2V 25°C corner @333 MHz (clock period = 3ns).

Fig. 14 shows the comparison of the area breakdown between the CMOS-based implementation and DWM-based implementation. For the comparison, we configure the CMOS-based implementation to the baseline architecture of Fig. 3 [4] which is converted to the DWM-based CNN convolutional layer design (Fig. 6). As shown in Fig. 6 and Fig. 14(c)-(d), the DWM-based in-memory-computing feature, which maps the dot product operation to the resistor sensing operation, efficiently eliminate the large footprint IBUF and WREG. Here, the magnetic nanowires of DWM are laid on the back-end-of-line (BEOL) layers while the other PE components (ADC, ACC, Adder, and DFF in Fig. 6) are occupied in the font-end-of-line (FEOL) layers. As a result, the DWM-based approach is 27% compact than the CMOS-based design even though the DWM-based PE unit takes $\times 4.7 \sim \times 5.0$ larger area than the CMOS-based PE unit as shown in Fig. 14. The power and energy breakdowns are presented in Fig. 15. Although the ADC of DWM-based PE unit consumes considerable power, i) inherent low power shift operation of DWM and ii) the 7 parallel DWM-based cell string leads to energy efficient CNN dot product operation. Compared to the CMOS-based design, 13% and 98.4% energy savings (Fig. 15(h)-(i)) are observed in the DWM-based IBUF and WREG, respectively. Consequently, the DWM-based approach shows overall 45% less energy consumption than the CMOS-based implementation. In addition, as presented in section IV-A and Fig. 6(c), about half of $N_c$ cycles are reduced in the DWM-based design since the partial sum registers in odd rows of PE array are removed. Here, the latency is calculated using $N_c$ as shown in Fig. 3(c).

In order to verify the effectiveness of the proposed DWM-based convolutional layer design, hardware costs in various PE array configurations are also simulated. In our design, as the bit-width of inputs and weights are changed from 4, 6, to 8-bit, the required number of iterations for a serial multiplication (for an '*input* × *weight*' term) increases to 4, 6, and 8, respectively. Accordingly, the energy savings over the CMOS-based design becomes smaller as the bit-width increases while the power consumption reduces with increasing bit-width as shown in Fig. 16(c)-(e). However, the 4-bit input and weight configurations can present the full-precision accuracy for recent CNNs as mentioned in recent bit-width quantization approaches [28], [29]. For this reason, the impact of PE array size on the area and energy efficiency has been investigated with 4-bit

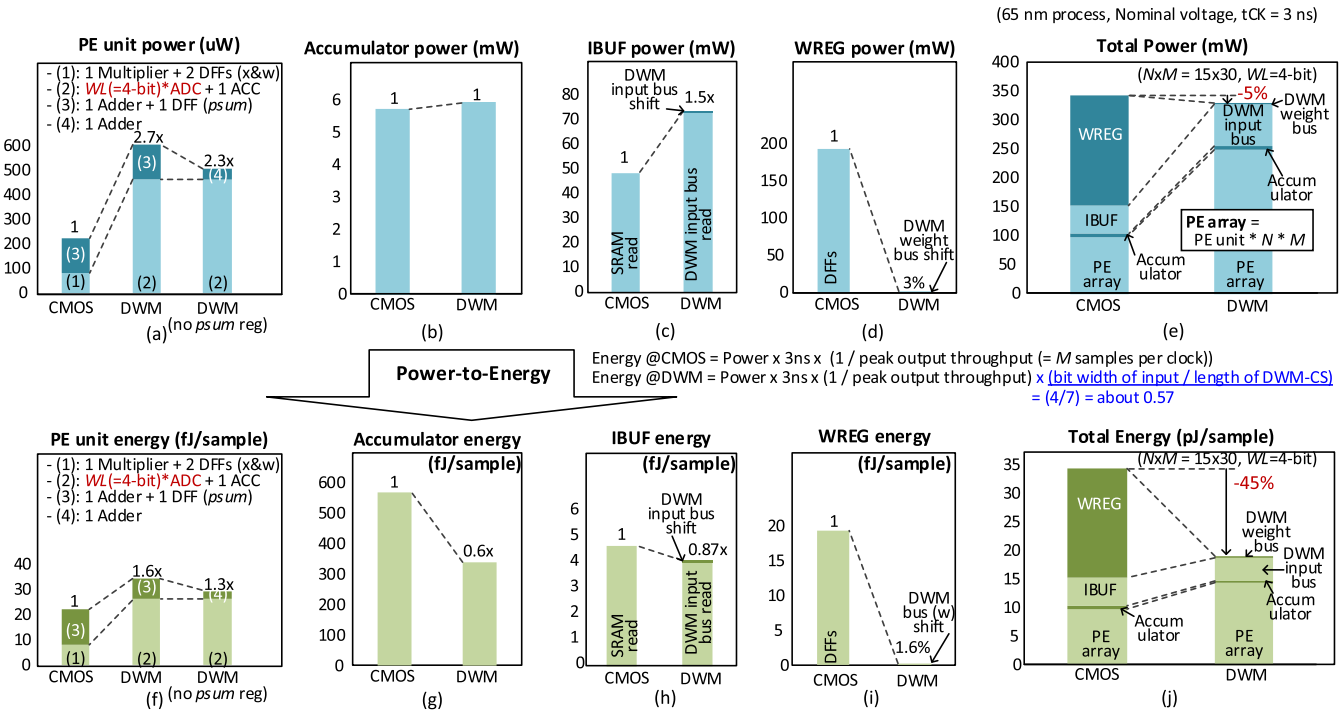



**FIGURE 15.** Power/energy consumption breakdown of the comparison between the conventional CMOS-based and the proposed DWM-based CNN convolutional layer implementations – (a)/(f) PE unit, (b)/(g) accumulator, (c)/(h) IBUF, (d)/(i) WREG, and (e)/(j) total (the PE array size $N \times M$ is 15 × 30 and the bit width of input activations and weights is 4-bit).

* Test accuracies on ImageNet (based on ResNet18)

| Model (bit width) | Top-1 | Top-5 |
|---|---|---|
| Baseline CNN (32-bit) | 69.76 | 89.08 |
| Recent CNN [28] (4-bit) | 69.74 | 89.08 |
| Original BNN [6] (1-bit) | 42.2 | 67.1 |
| Recent BNN [10] (1-bit) | 59.3 | 81.6 |

(65 nm process, Nominal voltage, tCK = 3 ns)

- V1 (2nd CONV layer in VGG16): 3x3 filter, stride=1, input size=224, input depth=64, output depth=64
- V2 (Last CONV layer in VGG16): 3x3 filter, stride=1, input size=14, input depth=512, output depth=512
- R1 (2nd CONV layer in ResNet18): 3x3 filter, stride=1, input size=64, input depth=64, output depth=64
- R2 (Last CONV layer in ResNet18): 3x3 filter, stride=1, input size=7, input depth=512, output depth=512

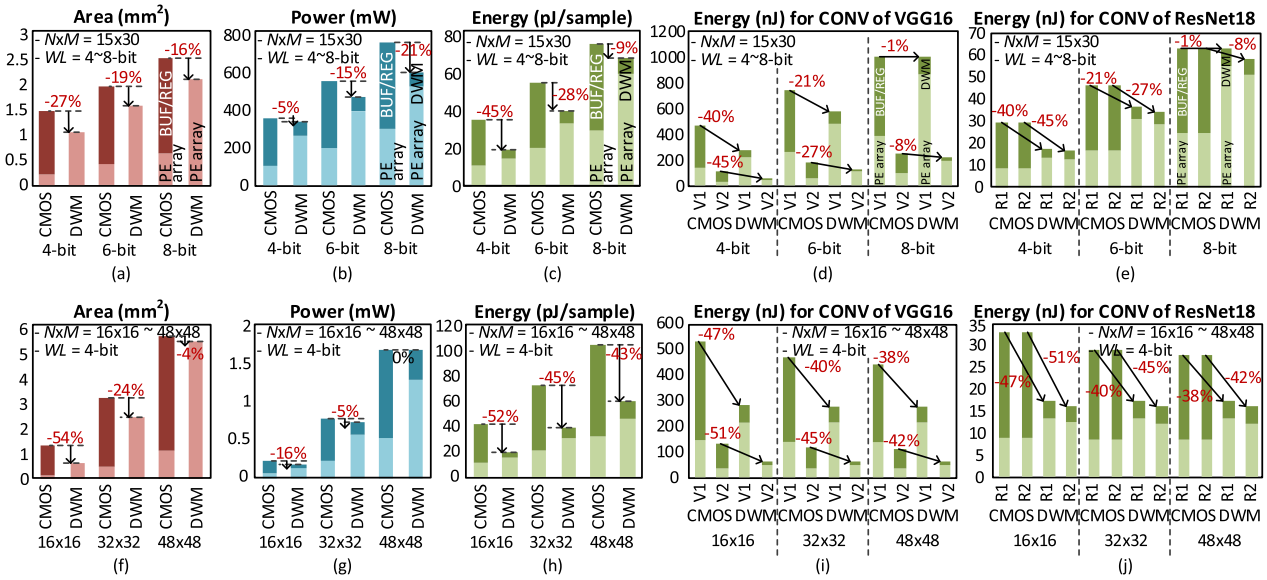



**FIGURE 16.** Comparison between the conventional CMOS-based and the proposed DWM-based CNN convolutional layer implementations in terms of area, power, and energy consumption – (a)-(e) for various bit width of input activation and weight and (f)-(j) for various PE array size at 4-bit.

inputs and weights. From Fig. 16(f), the area savings over the CMOS-based design become smaller due to the ADC overhead as the PE array size increases. The proposed DWM-based design shows significant energy reduction even with increasing PE array size (Fig. 16(h)-(j)). It is because the increased number of ADC can support more parallel dot product operations. Table 2 also shows the comparison results with the recent state-of-the-art work [22]. Compared to

**TABLE 2. Comparison with the state-of-the-art CNN accelerator.**

| | QUEST [22] | This work |
|---|---|---|
| Technology [nm] | 40 | 65 |
| Voltage [V] | 1.1 | 1.2 |
| Clock Frequency [MHz] | 300 | 333 |
| Bit precision [bit] | 4 | 4 |
| Performance [TOPS] | 1.825 | 1.829 |
| Power [W] | 2.083 | *1.697 |
| Energy Efficiency [TOPS/W] | 0.877 | *1.078 |

* For the comparison, the operative voltage is adjusted from 1.2V to 1.1V

the QUEST [22] which includes sufficient on-chip memory buffer using stacked-SRAM, the proposed DWM-based implementation shows 22.9% improved energy efficiency.

## V. DWM BASED BNN CONVOLUTIONAL LAYER DESIGN

This section presents the proposed DWM-based BNN convolutional layer design [30]. For the implementation of the parallel XNOR-popcount operation with DWM, the DWM input buses and the DWM-based weight reorder blocks for filter sliding operations will be discussed with specific examples.

### A. DWM-BASED BNN CONVOLUTIONAL LAYER

The architecture of the proposed DWM-based BNN convolutional layer is illustrated in Fig. 17(a). It consists of the DWM-based cell array for BNN, the accumulators, and the output buffer. In the BNN design, the reduced parameters (input activation and weight) facilitate the spatially unrolled architecture [24], for which the input activations and weights are fully preloaded in the DWM input buses and the DWM-based weight reorder blocks, respectively. Fig. 17(b) shows conceptual diagram of the proposed DWM-based cell array for BNN. Compared to the CNN convolutional layer, the output $Y_{1,1}$ of the convolutional layer is simply generated using bit-wise AND and bit count between the unrolled input activations $X^b$ and weights $W^b$. The detailed architecture of the DWM based BNN will be described in the following subsection.

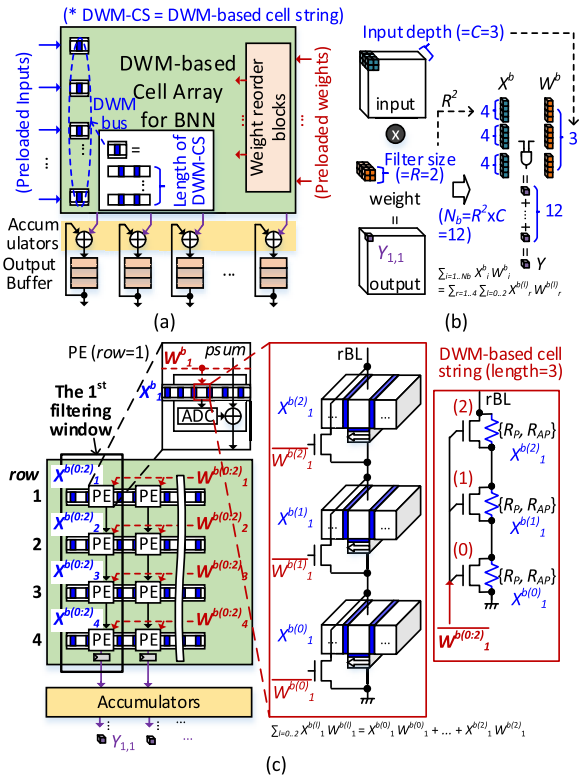



**FIGURE 17. (a) The block diagram of the proposed DWM-based BNN convolutional layer. (b) The conceptual diagram and (c) The detailed architecture of the proposed DWM-based cell array for BNN (for example, input depth $C = 3$, filter size $R = 2$, the length of the DWM-based cell string = 3, the first filtering window only).**

### B. DWM BASED CELL ARRAY FOR BNN

The DWM-based cell array for BNN is presented in Fig. 17(c). In the proposed design, to reduce the memory access in filter sliding operations, the DWM-based cell array performs the parallel XNOR-popcount operation [30].

#### 1) XNOR-POPCOUNT AND DWM-BASED CELL STRING

The accumulated result $\sum_{i=1}^{N_b} X_i^b W_i^b$, as shown in Fig. 17(b), can be decomposed as following:

$$\begin{aligned}&\sum_{i=1}^{N_b} X_i^b W_i^b \\ &= \sum_{l=0}^{2} \sum_{r=1}^{4} X_r^{b(l)} W_r^{b(l)} \\ &= \sum_{r=1}^{4} \left( X_r^{b(0)} W_r^{b(0)} + X_r^{b(1)} W_r^{b(1)} + X_r^{b(2)} W_r^{b(2)} \right)\end{aligned} \tag{7}$$

where $r$ is row index of the cell array, $l$ is the index of the DWM-based cell string, the input depth $C$ is 3, the filter size $R$ is 2, and the accumulation size $N_b$ is 12 ($=R \times R \times C$). The summation of the multiplied terms $(X_r^{b(0)} W_r^{b(0)} + X_r^{b(1)} W_r^{b(1)} + X_r^{b(2)} W_r^{b(2)})$ in (7) can be implemented using the bit count (XNOR-popcount in [6]) and the DWM-based bit-wise AND operations. As presented in Fig. 17(c), the input activations $X_1^{b(l)}$ are stored in MTJ, and weights $W_1^{b(l)}$ are applied to the gate input of the selective transistor. The output of $\sum_{l=0}^{2} X_1^{b(l)} \cdot W_1^{b(l)}$ can be implemented with the series of resistances (referred as 'DWM based cell string' in Fig. 17(c)). Similar to the DMW based cell string in the CNN convolutional layer, the serial resistance values depend on the input activations and weights.

#### 2) PARALLEL XNOR-POPCOUNT OPERATION BASED ON DWM

Fig. 18(a) shows the proposed DWM-based cell array for BNN and physical address mapping of input activations. For the $H \times H \times C$ input feature map and $R \times R \times C$ filtering windows (FWs), each PE that consists of the proposed DWM-based cell string, processes a $D$ vector ($1 \times 1 \times C$ input activations). Each bank in DWM-based cell array stores $R \times N \times N$ ($N = \text{floor}(H/R) + 1$) of $D$ vectors, and a bunch of $D$ vectors ($R \times R$ of $D$ vectors in the example of Fig. 18(a)) in a FW is handled in a column of DWM-based PE array.

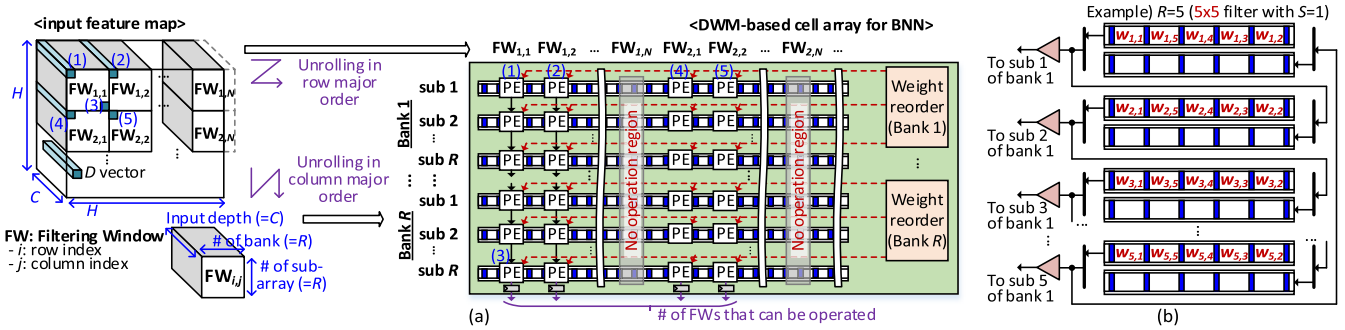



**FIGURE 18.** (a) The proposed DWM-based cell array for BNN and physical address mapping of input activations. (b) The DWM-based weight reorder block (filter size $R = 5$, stride $S = 1$, bank 1 only).

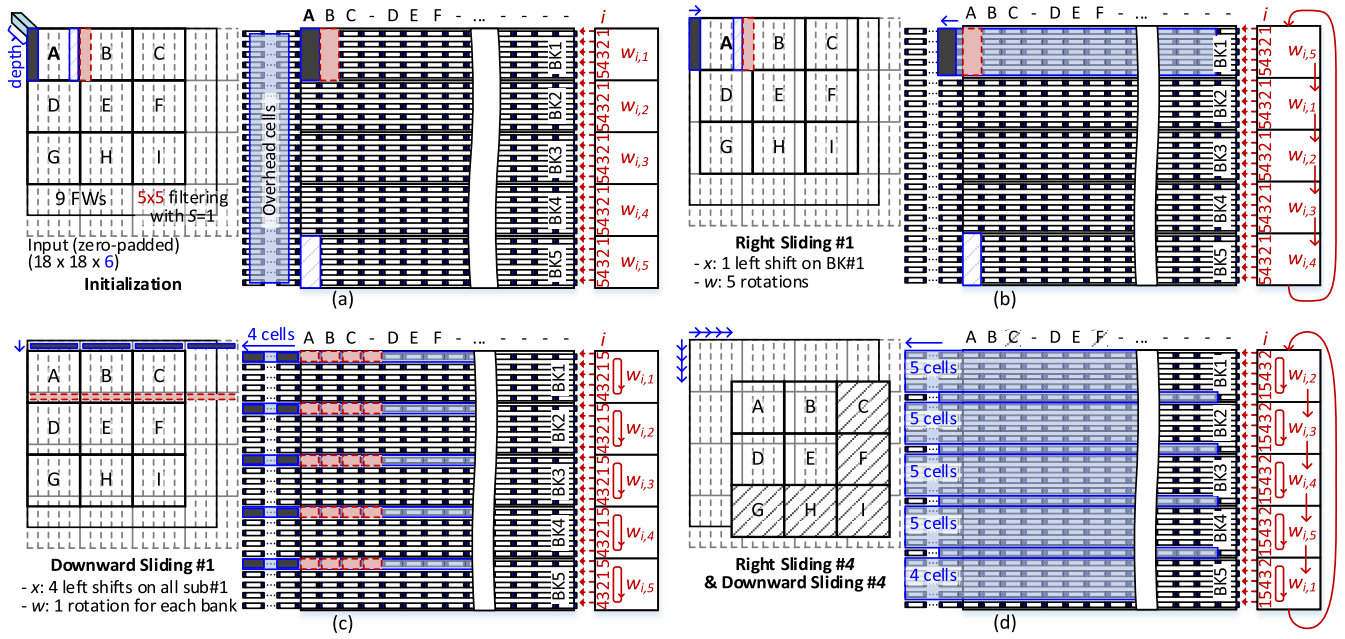



**FIGURE 19.** The proposed DWM-based cell array for BNN: (a) initialization, (b) After 1st right sliding, (c) After 1st downward sliding, and (d) After 4th right and 4th downward sliding with 18 × 18 × 6 input activation and 5 × 5 × 6 filter (stride = 1).

As shown in Fig. 18(a), $N \times N$ FWs can be placed on the input feature map. For those $N \times N$ FWs, the most top left $D$ vectors in the FWs is stored in sub-array 1 of bank 1, while most bottom right $D$ vectors in the FWs are placed in the sub-array $R$ of bank $R$. For the weight, PEs on the same row share the same weights which are generated from the weight reorder block.

Based on the data flow and the unrolled architecture, parallel XNOR-popcount operations on all the non-overlapped FWs can be performed in parallel. Fig. 19 shows the simplified example with the nine non-overlapped FWs (A, B, C, D, E, F, G, H, and I). First, the input activations and the weights of the non-overlapped FWs are loaded to the DWM-based PE array. On one hand, the input activations for the first column of FW-A (Fig. 19(a)) are stored in the first column of the Bank 1 (BK1 in Fig. 19(a)) in the BNN configuration. On the other hand, the input activations for the last column ($5^{th}$ column) of FW-A (Fig. 19(a)) are stored in the first column of Bank 5 (BK5 in Fig. 19(a)). The bunches of the input activations corresponding to each column of the FW are placed to each bank in numerical order. When the FW is sliding as shown in Fig. 19(b), since the input activations for the right-most column in FW-A are located in the second column of Bank 1, the input activations of BK1 for FW-A are laid in the different column of PE array. In order to align the input activations, left-shift operation of DWM buses are performed in Bank 1 such that the input activations and weight can be paired by rotating the weights. Since the weights for the first column of FW-A (Fig. 19(b)) are previously (before sliding) located in Bank 2, weights that are previously paired to Bank 1 location should move to Bank2 location. For this reason, bank-to-bank counter-clockwise rotation is performed as shown in the right-most side of Fig. 19(b). When FW is sliding as shown in Fig. 19(c),

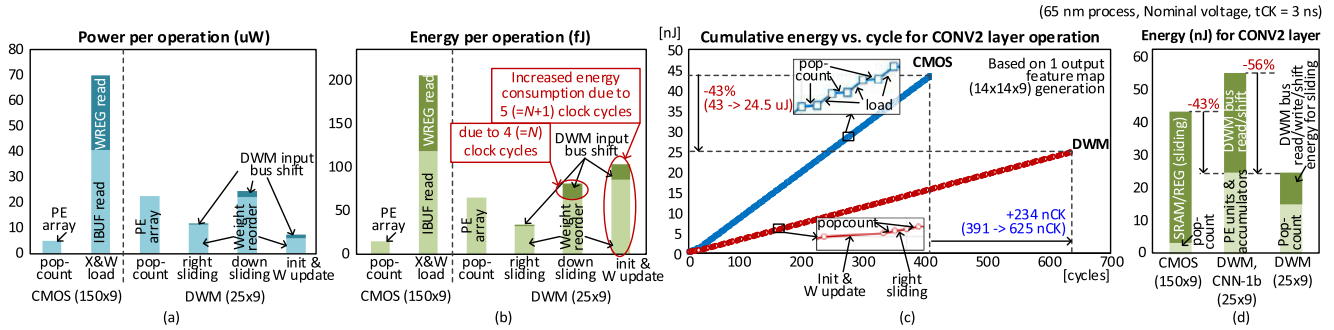



**FIGURE 20.** Comparison of (a) power and (b) energy consumption per operation breakdowns between the conventional CMOS-based and the proposed DWM-based BNN convolutional layer implementations. Comparison of (c) cumulative energy vs. clock cycles and (d) energy consumption for CONV2 layer operation.

the input activations for bottom-most row in FW-A are previously (Fig. 19(a)) located in first row of FW-D. At this time, since those input activations are distributed in the first rows (sub-array 1s) of all Banks, left-shift operation of DWM buses are performed in first rows (sub-array 1s) of all the Banks. In order to align input-weight pair, intra-bank clockwise rotation is conducted as shown in the right-most side of Fig. 19(c). As shown in Fig. 19(d), the number of the overhead cells depends on the size of the input feature map and filter.

Using the shift operations of the DWM buses, the bank-to-bank rotation (inter-bank rotation) and intra-bank rotation can be performed for weight re-ordering. The detailed architecture of the weight reorder block is presented in Fig. 18(b). For the right sliding of FW, the weights are first preloaded in the weight reorder block. The reorder block outputs the leftmost weights in DWM buses during right sliding of FW. For the downward sliding of FW, those weights are shifted to the extra DWM buses, which are located in the next subarray. With the weight reordering and input activation aligning, the parallel XNOR-popcount operation has been enabled in the proposed DWM-based PE array.

The PE utilization ratio (PEUR) of the DWM-based cell array for BNN is determined by input feature map size ($H$) and filter size ($R$). Under the same number of PE units, different layer configuration (different input feature map size ($H_d$) and filter size ($R_d$)) induces different PEUR. The PEUR can be calculated as follows:

$$\text{PEUR} = \frac{\left(\min\left\{\acute{N}_d, \acute{N}\right\}\right)^2 \cdot \left(\min\left\{R_d, R\right\}\right)^2}{\acute{N}^2 \cdot R^2} \times 100\% \quad (8)$$

where $N' = \text{floor}(H/R)$ and $N'_d = \text{floor}(H_d/R_d)$. After FW sliding, when some of FWs overlap the boundaries of the input feature map only to the right or down, the PEUR can be calculated by replacing $(\min\{N'_d, N'\})^2$ in the numerator of (8) to $\min\{N'_d, N'\} \times (\min\{N'_d, N'\}\text{-}1)$. On the other hand, when some FWs overlap the boundaries of the input feature map to the right and down (Fig. 19(d)), the PEUR can be calculated by replacing $(\min\{N'_d, N'\})^2$ in the numerator of (8) to $(\min\{N'_d, N'\}\text{-}1)^2$.

### C. BNN IMPLEMENTATION RESULTS

Based on the simulation environment of the CNN design, the BNN hardware cost comparisons between the CMOS-based design and DWM-based design are performed, and the numerical results are presented in Fig. 20. For the comparison, since the DWM-based BNN design enables the dot product operations of the multiple FWs simultaneously (Fig. 19), the CMOS-based BNN architecture is modified to have the same output throughput. The power and energy cost per operation (pop-count and data movement) are presented in Fig. 20(a)-(b). In the overall CONV2 layer operations, as the pop-count and right sliding are dominant, the cumulative energy during each time interval is computed and presented in Fig. 20(b). As shown in Fig. 20(c)-(d), since the DWM-based in-memory-computing feature alleviates the expensive data movement cost in the IBUF and WREG, 43% reduced cumulative energy at the cost of 234 increasing number of processing cycles. Since those spatially unrolled architecture shows relatively higher throughput, and the off-chip DRAM access latency is larger than the accelerator latency, the timing overhead can be overshadowed in most of the neural network accelerator. Fig. 21 shows the area comparison for the overall CONV2 layer. As shown in the figure, since the DWM nests on the BEOL layer, 60% of area saving is achieved compared to the CMOS-based design. Compared to the DWM-based 1-bit CNN cell array, the proposed BNN cell array shows 26% area saving and 56% energy

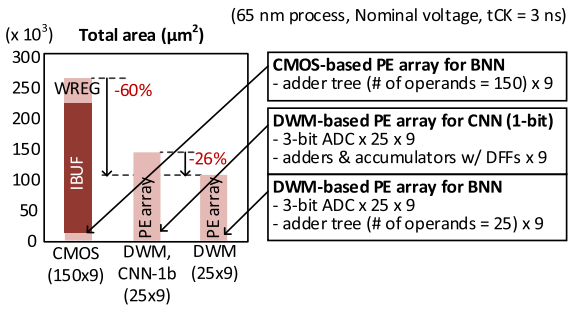



**FIGURE 21.** Area comparisons among the conventional CMOS-based, the proposed DWM-based CNN (1-bit version), and the proposed DWM-based BNN convolutional layer implementations.

reduction, as shown in Fig. 21 and Fig. 20(d), respectively. This is because the CNN cell array is systolic based, the PE units need DFFs for the intermediate results. On the other hand, since the unrolled BNN structure facilitates the XNOR pop-count operation at once, the intermediate stage and DFF overhead can be eliminated in the proposed DWM-based BNN cell array. The comparison with the state-of-the-art is also presented in Table 3. The proposed DWM-based BNN design shows 17.8% higher energy efficiency than [20] without erroneous output activations.

**TABLE 3. Comparison with the state-of-the-art BNN accelerator.**

| | IPDPSW2017 [23] (redesigned [30]) | This work |
|---|---|---|
| Technology [nm] | 65 | 65 |
| Voltage [V] | 1.1 | 1.2 |
| Clock frequency [MHz] | - | 333 |
| Energy efficiency [TOPS/W] | 25.2 | *29.7 |
| Computation method (accuracy degradation) | Bit-exact (0%) | Bit-exact (0%) |

* For the comparison, the operative voltage is adjusted from 1.2V to 1.1V

## VI. CONCLUSION

In this paper, we propose a novel DWM-based cell array architecture for an area and energy-efficient CNN/BNN convolutional layer design. By exploiting the resistive cell sensing mechanism, the dot product and XNOR-popcount operations that mainly compose the CNN/BNN convolutional layer, are effectively implemented with DWM. For designing filter sliding in CNN and BNN convolutional layers, optimal data flow is also proposed by exploiting the sequential access pattern of DWM. In the CNN convolutional layer design, the proposed DWM-based implementation shows 27% of area reduction compared to the CMOS-based design due to the BEOL layer nested DWM feature. In addition, the inherent low power shift operation of DWM and the 7 parallel DWM-based cell string lead to 45% of energy savings. Similarly, in the BNN convolutional layer design, the efficient data movement of DWM-based architecture enables 43% of reduced energy cost while occupying only 40% area compared to the CMOS-based design.

## REFERENCES


[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[3] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[4] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE Int. Symp. Comput. Arch.*, Jun. 2017, pp. 1–12.

[5] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2014, pp. 609–622.

[6] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1," Feb. 2016, *arXiv:1602.02830*. [Online]. Available: https://arxiv.org/abs/1602.02830

[7] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: Imagenet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 525–542.

[8] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2016, *arXiv:1606.06160*. [Online]. Available: https://arxiv.org/abs/1606.06160

[9] W. Tang, G. Hua, and L. Wang, "How to train a compact binary neural network with high accuracy?" in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 2625–2631.

[10] J. Gu, J. Zhao, X. Jiang, B. Zhang, J. Liu, G. Guo, and R. Ji, "Bayesian optimized 1-bit CNNs," in *Proc. IEEE Int. Conf. Comput. Vis.*, Oct. 2019, pp. 4909–4917.

[11] S. Ghosh, "Path to a terabyte of on-chip memory for petabit per second bandwidth with < 5 Watts of power," in *Proc. ACM/IEEE Design Autom. Conf.*, May/Jun. 2013, pp. 1–2.

[12] 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 *Proc. ACM/IEEE Int. Symp. Low Power Electron. Design*, 2012, pp. 185–190.

[13] B. Liu, S. Gu, M. Chen, W. Kang, J. Hu, Q. Zhuge, and E. H.-M. Sha, "An efficient racetrack memory-based processing-in-memory architecture for convolutional neural networks," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Appl. IEEE Int. Conf. Ubiquitous Comput. Commun. (ISPA/IUCC)*, Dec. 2017, pp. 383–390.

[14] M. H. Samavatian, A. Bacha, L. Zhou, and R. Teodorescu, "RNN-Fast: An accelerator for recurrent neural networks using domain wall memory," Nov. 2018, *arXiv:1812.07609*. [Online]. Available: https://arxiv.org/abs/1812.07609

[15] J. Chung, K. Ramclam, 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 Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 1, pp. 91–102, Jan. 2016.

[16] J. Chung, J. Park, and S. Ghosh, "Domain wall memory based convolutional neural networks for bit-width extendability and energy-efficiency," in *Proc. Int. Symp. Low Power Electron. Design-ISLPED*, 2016, pp. 332–337.

[17] R. Venkatesan, M. Sharad, K. Roy, and A. Raghunathan, "DWM-TAPESTRI-an energy efficient all-spin cache using domain wall shift based writes," in *Proc. Design, Autom. Test Europe Conf. Exhibit. (DATE)*, 2013, pp. 1825–1830.

[18] C. Augustine, "Spintronic memory and logic: From atoms to systems," Ph.D. dissertation, Dept. Elect. Eng., Purdue Univ., West Lafayette, IN, USA, 2011.

[19] M. Hayashi, "Current driven dynamics of magnetic domain walls in permalloy nanowires," Ph.D. dissertation, Dept. Mater. Sci. Eng., Stanford Univ., Stanford, CA, USA, 2006.

[20] A. Iyengar and S. Ghosh, "Modeling and analysis of domain wall dynamics for robust and low-power embedded memory," in *Proc. 51st Annu. Design Autom. Conf. Design Autom. Conf.-DAC*, 2014, pp. 1–6.

[21] *CS231n Convolutional Neural Networks for Visual Recognition*. Accessed: Nov. 8, 2019. [Online]. Available: http://cs231n.github.io/convolutional-networks/

[22] K. Ueyoshi, K. Ando, K. Hirose, S. Takamaeda-Yamazaki, M. Hamada, T. Kuroda, and M. Motomura, "QUEST: Multi-purpose log-quantized DNN inference engine stacked on 96-MB 3-D SRAM using inductive coupling technology in 40-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 186–196, Jan. 2019.

[23] H. Yonekawa and H. Nakahara, "On-chip memory based binarized convolutional deep neural network applying batch normalization free technique on an FPGA," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2017, pp. 98–105.

[24] D. Miyashita, S. Kousai, T. Suzuki, and J. Deguchi, "A neuromorphic chip optimized for deep learning and CMOS technology with time-domain analog and digital mixed-signal processing," *IEEE J. Solid-State Circuits*, vol. 52, no. 10, pp. 2679–2689, Oct. 2017.

[25] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. Boston, MA, USA: Addison-Wesley, 2010.

[26] R. J. Baker, *CMOS Digital Circuit Design-No Text*, 3rd ed. Piscataway, NJ, USA: Wiley-IEEE Press, 2010.

[27] J.-C. Kim, Y.-C. Jang, and H.-J. Park, "CMOS sense amplifier-based flip-flop with two N-C2MOS output latches," *Electron. Lett.*, vol. 36, no. 6, p. 498, 2000.
[28] J. L. McKinstry, S. K. Esser, R. Appuswamy, D. Bablani, J. V. Arthur, I. B. Yildiz, and D. S. Modha, "Discovering low-precision networks close to full-precision networks for efficient embedded inference," Sep. 2018, *arXiv:1809.04191*. [Online]. Available: https://arxiv.org/abs/1809.04191
[29] S. Jung, C. Son, S. Lee, J. Son, J.-J. Han, Y. Kwak, S. J. Hwang, and C. Choi, "Learning to quantize deep networks by optimizing quantization intervals with task loss," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4350–4359.
[30] W. Choi, K. Jeong, K. Choi, K. Lee, and J. Park, "Content addressable memory based binarized neural network accelerator using time-domain signal processing," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.

**JINIL CHUNG** (Student Member, IEEE) received the B.S. and M.S. degrees from Konkuk University, Seoul, South Korea, in 2002 and 2004, respectively. He is currently pursuing the Ph.D. degree with the School of Electrical Engineering, Korea University, Seoul. Since 2004, he has been with DRAM Design Team, SK hynix Inc., South Korea. He has developed DLL and data path for DRAM device. His research interest includes VLSI architectures and circuit design techniques for low-complexity digital signal processor.

**WOONG CHOI** (Member, IEEE) received the B.S. and Ph.D. degrees in electronics engineering from Korea University, Seoul, South Korea, in 2011 and 2018, respectively. In 2018, he joined Samsung Electronics Ltd., Hwa-sung, South Korea. Since 2019, he has been an Assistant Professor with the Department of Electronics Engineering, Sookmyung Women's University, Seoul. His current research interests include the neural network accelerator and emerging memory (STT-MRAM) designs in advanced technologies.

**JONGSUN PARK** (Senior Member, IEEE) received the B.S. degree in electronics engineering from Korea University, Seoul, South Korea, in 1998, and the M.S. and Ph.D. degrees in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 2000 and 2005, respectively.

He joined the Electrical Engineering Faculty of Korea University, Seoul, in 2008. He was with the Digital Radio Processor System Design Group, Texas Instruments, Dallas, TX, USA, in 2002. From 2005 to 2008, he was with the Signal Processing Technology Group, Marvell Semiconductor Inc., Santa Clara, CA, USA. His research interests focus on variation-tolerant, low-power, high-performance VLSI architectures and circuit designs for digital signal processing, and digital communications.

Dr. Park is a member of the Circuits and Systems for Communications Technical Committee of the IEEE Circuits and Systems Society. He served as the Guest Editor of the IEEE TRANSACTIONS ON MULTI-SCALE COMPUTING SYSTEMS. He has also served in the Technical Program Committees of various IEEE/ACM conferences, including ICCAD, ISLPED, ISCAS, ASP-DAC, HOST, VLSI-SoC, and APCCAS. He is an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS.

**SWAROOP GHOSH** (Senior Member, IEEE) received the B.E. (Hons.) from IIT Roorkee, Roorkee, India, in 2000, the M.S. degree from the University of Cincinnati, Cincinnati, OH, USA, in 2004, and the Ph.D. degree from Purdue University, West Lafayette, IN, USA, in 2008.

Since 2016, he has been an Assistant Professor with Penn State University. Earlier, he was with the Faculty of University of South Florida, from 2012 to 2016. Prior to that, he was a Senior Research and Development Engineer in Advanced Design, Intel Corp, from 2008 to 2012. At Intel, his research was focused on low power and robust embedded memory design in scaled technologies. His research interests include low-power circuits, hardware security, quantum computing and digital testing for nanometer technologies.

Dr. Ghosh is a Senior Member of the National Academy of Inventors (NAI) and the Associate Member of Sigma Xi. He was a recipient of Intel Technology and Manufacturing Group Excellence Award, in 2009, the Intel Divisional Award, in 2011, the Intel Departmental Awards, in 2011 and 2012, the USF Outstanding Research Achievement Award, in 2015, the College of Engineering Outstanding Research Achievement Award, in 2015, the DARPA Young Faculty Award (YFA), in 2015, the ACM SIGDA Outstanding New Faculty Award, in 2016, the YFA Director's Fellowship, in 2017, the Monkowsky Career Development Award, in 2018, Lutron Spira Teaching Excellence Award, in 2018, and the Dean's Certificate of Excellence, in 2019. He served as the Program Chair of ISQED 2019 and DAC Ph.D. Forum 2016 and the Track Co-Chair of CICC 2017–2019, ISLPED 2017–2018 and ISQED 2016–2017. He served as the Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I and the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN, and as the Senior Editorial Board Member of the IEEE JOURNAL OF EMERGING TOPICS ON CIRCUITS AND SYSTEMS (JETCAS). He served as the Guest Editor of the IEEE JETCAS and IEEE TRANSACTIONS ON VLSI SYSTEMS. He has also served in the Technical Program Committees of ACM/IEEE conferences, such as, DAC, ICCAD, CICC, DATE, ISLPED, GLSVLSI, Nanoarch, and ISQED.