Design and Optimization of FeFET-based Crossbars for Binary Convolution Neural Networks

Xiaoming Chen[1,2], Xunzhao Yin[1], Michael Niemier[1], Xiaobo Sharon Hu[1]
[1]Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA
[2]State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
Email: chenxiaoming@ict.ac.cn, {xyin1, mniemier, shu}@nd.edu

***Abstract*—Binary convolution neural networks (CNNs) have attracted much attention for embedded applications due to low hardware cost and acceptable accuracy. Nonvolatile, resistive random-access memories (RRAMs) have been adopted to build crossbar accelerators for binary CNNs. However, RRAMs still face fundamental challenges such as sneak paths, high write energy, etc. We exploit another emerging nonvolatile device–ferroelectric field-effect transistor (FeFET), to build crossbars to improve the energy efficiency for binary CNNs. Due to the three-terminal transistor structure, an FeFET can function as both a nonvolatile storage element and a controllable switch, such that both write and read power can be reduced. Simulation results demonstrate that compared with two RRAM-based crossbar structures, our FeFET-based design improves write power by 5600× and 395×, and read power by 4.1× and 3.1×. We also tackle an important challenge in crossbar-based CNN accelerators: when a crossbar array is not large enough to hold the weights of one convolution layer, how do we partition the workload and map computations to the crossbar array? We introduce a hardware-software co-optimization solution for this problem that is universal for any crossbar accelerators.**

I. Introduction

In the past decade, convolution neural networks (CNNs) have achieved great successes in numerous applications, such as image classification, object detection, natural language processing, etc. High-accuracy results produced by state-of-the-art CNNs almost exclusively rely on complex models and large amounts of training data, which in turn require large amounts of memory and energy. As a result, it is extremely challenging to run deep CNNs on resource-limited embedded platforms, such as smart phones. To tackle this challenge, binary neural networks (NNs) [1]–[3] have recently been proposed. In binary NNs, the weights and/or activations are binarized to ± 1. Such an approximation significantly reduces the energy, memory usage, and execution time, with acceptable accuracy.

To further reduce energy and computation time, researchers are studying hardware NN accelerators, especially based on emerging devices. As a nonvolatile device, resistive random-access memories (RRAMs) can realize analog multiplications by programming the devices to multi-level states, and are promising candidates for area-efficient NN accelerators/crossbar structures. In an RRAM crossbar array, RRAMs store NN weights, *and* also perform computations between weights and inputs. RRAM-based crossbar arrays have been used for accelerators for both conventional and binary NNs (e.g., [4]–[14]). However, RRAMs still face challenges such as sneak paths, high write energy, device variations, etc.

Ferroelectric field-effect transistors (FeFETs) [15] have gained significant attention as **(i)** they have the potential to save power and area by leveraging device nonvolatility and **(ii)** devices have been experimentally demonstrated [16]–[18]. An FeFET is made by integrating a ferroelectric (FE) layer in the gate stack of a metal-oxide-semiconductor field-effect transistor (MOSFET). By tuning FE layer thickness, hysteresis can be obtained in the I_{DS}-V_{G} curve. As a three-terminal transistor, an FeFET can function as both a nonvolatile storage element and a controllable switch. This is a unique advantage when compared to RRAMs, as the three-terminal structure allows for easy control of both write and read power.

We introduce crossbar-based binary CNN (BCNN) accelerators by exploiting FeFET nonvolatility to improve power efficiency for BCNN inference. We present qualitative analysis and quantitative simulations to demonstrate superior power efficiency of FeFET-based crossbars enabled by a three-terminal structure. Compared with two RRAM-based crossbar structures, our design improves write power by 5600× and 395×, and read power by 4.1× and 3.1×. To our best knowledge, this is the first FeFET-based crossbar BCNN. We also tackle an important challenge in crossbar-based CNN accelerators: when a crossbar array is not large enough to hold the weights of one convolution layer, how do we partition the workload and map computations to the crossbar array? We introduce a hardware-software co-optimization solution for this problem that is universal for any crossbar-based CNN accelerators.

II. Preliminaries

A. Basics of FeFETs

An FeFET is made by integrating an FE layer in the gate stack of a MOSFET [15], per Fig. 1a. This is equivalent to integrating a negative capacitance in the gate stack. The behavior of FeFETs strongly depends on the thickness of the FE layer (T_{FE}). FeFETs nonvolatility is achieved by properly selecting T_{FE} so as to introduce a hysteresis loop in the I_{DS}-V_{G} curve [19]. FeFETs are compatible with MOSFETs [18].

Fig. 1b shows a simulated hysteresis curve of an FeFET, using a recently developed FeFET simulation model [20]. Simulations use a 10nm predictive technology model (PTM) [21]. T_{FE} is 10.5nm. By selecting a proper T_{FE}, a hysteresis in the I_{DS}-V_{G} curve is obtained. Similar hysteresis curves have been observed from fabricated devices [16], [17]. The existence of the hysteresis means that the state of an FeFET can be read out

This work was supported in part by the Center for Low Energy Systems Technology (LEAST), one of six SRC STARnet centers sponsored by MARCO and DARPA. This work was also supported in part by the Nanoelectronics Research Corporation (NERC), a wholly-owned subsidiary of the Semiconductor Research Corporation (SRC), through Extremely Energy Efficient Collective Electronics (EXCEL), an SRC-NRI Nanoelectronics Research Initiative under Research Task ID 2698.004.

by setting a proper V_G (e.g., $V_G = 0$ in the case of Fig. 1b). By applying a certain V_G, the current I_{DS}, which reflects the state of the FeFET, depends on the polarization of the FE layer.

To change the polarization of the FE layer for programming, we need to apply a positive or negative V_G pulse with a proper magnitude (e.g., 0.6V per Fig. 1b), as shown in Fig. 1c.

B. Basics of BCNNs

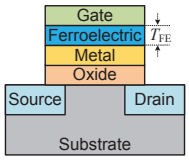
We consider BCNNs with both binarized weights and activations [2], [3]. We focus on convolution layers which dominate execution time and energy. Fully connected layers can also be mapped to crossbars. Other CNN operations such as max pooling can be computed by software or simple hardware, and are not described here.

Fig. 2 illustrates a convolution layer in CNNs. One convolution operation can be described by the inner product between a filter matrix and a convolution window on the input matrix:

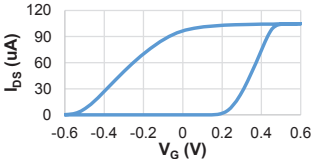
$$Y(i,j,k) = \sum_{p=1}^{W_F} \sum_{q=1}^{H_F} \sum_{r=1}^{C_{\text{in}}} X(i+p, j+q, r) \cdot W_k(p,q,r) \quad (1)$$

where (i, j, k) is the 3D coordinate in the output matrix, X is the input matrix, Y is the output matrix, and W_k is the kth filter matrix (i.e., weight matrix). We slide the convolution window on the input matrix to compute different output points.

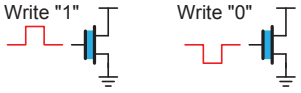
In BCNNs, both weights and inputs are ± 1. In implementation, -1 is mapped to 0. Then a multiplication between ± 1 is equivalent to an XNOR operation. Thus, the convolution operation in Eq. (1) is converted to a number of bit-wise XNOR operations followed by an accumulation (i.e., bit-counting) operation [2], [3]. The result of one binary convolution is an integer. As outputs of one layer will be used as the inputs to the next, we need to convert outputs to binary numbers. This is typically done by simply taking the signs of the resulting integers [2], [3]. Before binarization, we can apply



(a) FeFET structure.



(b) Simulated I_{DS}-V_G hysteresis curve.



(c) Programming an FeFET.

Fig. 1: Background of FeFETs.

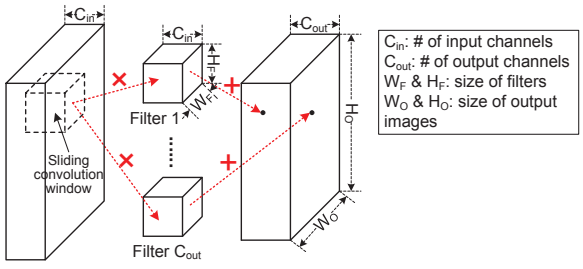


Fig. 2: Illustration of a convolution layer in CNNs.

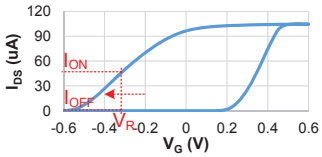


Fig. 3: Tuning the read current of an FeFET by controlling V_G.

batch normalization [22], which can improve the accuracy of CNNs. In the context of BCNN inference, batch normalization is equivalent to adding a fixed offset to the resulting integers.

C. Related Work

1) RRAM-based Accelerators for Conventional NNs: For conventional NNs (with floating-point values), researchers have comprehensively studied how to construct crossbar-based accelerators using RRAMs at various design levels. Architectures for inference and online training have been proposed [4], [5]. Circuits for RRAM programming and state tuning have been designed [6]. Optimization of peripheral circuits and interfaces has been proposed [7], [8]. There are also fabricated chips to demonstrate the superior performance and energy efficiency of RRAM-based NN accelerators [9]–[11].

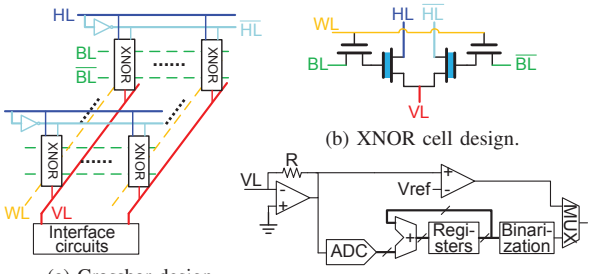
2) RRAM-based Accelerators for Binary NNs: Although crossbars based on multi-state RRAMs have been widely studied, multi-state RRAMs still face challenges such as limited precision, variability, non-uniformity, etc. However, if only on and off states of RRAMs are used, these problems can be avoided. Binary NN accelerators based on binary-state RRAMs have recently been proposed [12]–[14]. Work in [14] also discussed the workload partitioning problem. However, [14] just calculated how many sub-matrices will be generated after partitioning. To the best of our knowledge, optimization of the workload mapping problem has not been studied.

3) FeFET-based Circuit Designs: FeFETs have been adopted to build novel logic-in-memory circuits by integrating nonvolatile storage elements into logic [23]–[25]. FeFET-based memories, latches, and flip-flops have been proposed [26]–[28]. All circuits have lower power and area than complementary metal-oxide-semiconductor (CMOS) based equivalents, which is obtained from FeFET nonvolatility. There is no published work on FeFET-based crossbars.

III. FeFET Crossbar Accelerator for BCNNs

We now describe our FeFET-based crossbar accelerator for BCNNs as well as write and read schemes. We first present a qualitative comparison between FeFETs and RRAMs in the context of crossbar-based BCNN accelerators to explain why FeFETs can reduce both write and read power. Advantages in power efficiency are mainly obtained by the three-terminal structure of FeFETs.

To program an RRAM, a write voltage (typically 2V-3V) is applied to its two terminals to change its state. This voltage needs to be maintained during programming and leads to a static write current and high power. For FeFETs, however, programming is realized by applying a gate voltage (see Fig. 1c). Thus, we can set V_{DS} to zero when programming. This will not affect the programming operation, but I_{DS} is reduced to almost zero during programming. Write power is



(a) Crossbar design. (b) XNOR cell design. (c) Interface circuit for one column.

Fig. 4: FeFET crossbar, XNOR cell, and interface circuit designs.

only consumed by charging the FE layer capacitance, which is much lower than that caused by the write current of RRAMs.

Read power can also be reduced by controlling gate voltage. While the drain is typically connected to a power supply (making it difficult to tune), we can apply a lower read voltage V_R to the gate to reduce I_{DS} (see Fig. 3) and read power. If we set the read voltage applied to the two terminals of an RRAM to be the same as the drain voltage of an FeFET, there is no way to tune the read power of the RRAM.

A. Crossbar Design

Fig. 4a shows our FeFET-based crossbar structure. Each cell is an FeFET-based XNOR gate and is connected to a horizontal line (HL), its inverse ($\overline{\text{HL}}$), a vertical line (VL), a word line (WL), a bit line (BL), and its inverse ($\overline{\text{BL}}$). Peripheral circuits such as input and output buffers are not shown. These circuits and the architectural-level designs have been well studied in prior work [4], [5], [14].

Fig. 4b shows the basic cell in a FeFET-based crossbar array. It consists of two FeFETs and two access transistors. The two FeFETs (storing complementary bits) store one weight bit. One bit of the inputs and its complementary bit are applied to HL and $\overline{\text{HL}}$. The cell performs an XNOR operation between the input bit and the weight bit stored in the two FeFETs. The output bit can be read out from VL by either voltage or current. In the context of crossbars, we sense and amplify the current of VLs to read the outputs. Two access transistors control read and write operations (described in Section III-B).

The interface circuits in Fig. 4a are used to convert the VL currents to binary bits, which is used as the input bits of the next layer. Fig. 4c shows the interface circuit for one column. The first operational amplifier (OA) is a current-to-voltage converter. The second OA is a voltage comparator. One issue to address is that when a crossbar array is not sufficiently large to hold weights for one convolution layer, we need to partition computations and accumulate intermediate results (i.e., partial sums), which are integers. We add a branch composed of an analog-to-digital converter (ADC), an adder, registers and a digital binarization module to perform accumulation. The final output bit is selected by a multiplexer (MUX). Batch normalization [22] can be easily realized by tuning the reference voltage of the voltage comparator or the offset point of the ADC.

The overall structure of our FeFET crossbar array is conventional, while the major difference with RRAM-based crossbars is at the cell level. Existing RRAM-based crossbars typically use two crossbar arrays to store positive and negative weights, respectively (e.g., [6]), where each cell is a single RRAM (1R) or an RRAM with an access transistor (1T1R) [29], [30]. The reason for using two separate crossbar arrays is due to programming. The VL voltages for writing "1" and "0" are different in an RRAM crossbar array. If we build similar XNOR cells using RRAMs, the cells cannot be programmed without access transistors, while for the cells with access transistors, we need two steps to program them. For FeFETs however, we can combine two crossbar arrays into one array by creating FeFET XNOR gates, as programming FeFETs is controlled by the gate voltage (see Fig. 1c). Both the HLs and VLs are not needed during programming so they are set to zero to eliminate I_{DS} when writing.

TABLE I: Read scheme for our FeFET-based crossbar array.

HL	$\overline{\text{HL}}$	WL	BL	$\overline{\text{BL}}$
Input bit	Inverse of input bit	V_{DD}	V_{R}	V_{R}

TABLE II: Write scheme for our FeFET-based crossbar array.

	HL & $\overline{\text{HL}}$	WL (selected)	WL (unselected)	BL	$\overline{\text{BL}}$
Write "1"	0	V_{WL}	$-V_{\text{WL}}$	V_{W}	$-V_{\text{W}}$
Write "0"	0	V_{WL}	$-V_{\text{WL}}$	$-V_{\text{W}}$	V_{W}

When the crossbar array size is large enough to hold the weights of one convolution layer, the mapping of the computations is straightforward and well-known. The input bits in one convolution window are applied to HLs, and each convolution filter matrix is mapped to one column of the crossbar array. Each column outputs one point on the corresponding output image channel. The problem of insufficient crossbar array size is discussed in the next section.

B. Read and Write Schemes

The voltage settings for inference (i.e., read) operations are shown in Table I. Input bits and their inverse are applied to HLs and $\overline{\text{HL}}$s. Both BLs and $\overline{\text{BL}}$s are set to the read voltage level V_{R} (see Fig. 3). V_{DD} is applied to all the WLs to transmit the read voltage to the gate terminals of FeFETs.

Programming (i.e., writing) a crossbar array is done column by column. The voltage settings for selected and unselected columns are different. Table II shows the voltage settings for programming. During programming, both HLs and $\overline{\text{HL}}$s are set to zero to eliminate I_{DS}. For the selected column (i.e., the column to be programmed), its WL is set to V_{WL} to transmit the BL and $\overline{\text{BL}}$ voltages, while for all the unselected columns, the WLs are set to $-V_{\text{WL}}$ to cut off the BL and $\overline{\text{BL}}$ voltages. The voltages of all the BLs and $\overline{\text{BL}}$s are set to $\pm V_{\text{W}}$ according to the bits that will be written into the selected column, where V_{W} is the write voltage. V_{WL} should be $\geq V_W$. In our simulations, both V_{WL} and V_W are set to 0.6V.

During programming, our FeFET-based crossbar array does not suffer from the sneak path problem [29], [30] which exists in RRAM crossbars based on the 1R cell, because the voltages of HLs, $\overline{\text{HL}}$s, and VLs are all zero during programming. Elimination of sneak paths significantly reduces write power.

IV. Workload Partitioning and Mapping

The size of a crossbar array is often limited due to yield concerns. If a crossbar array is not large enough to hold the weights of a single layer, different ways of partitioning and mapping computations can have significant impact on performance and energy. This problem is not limited to our FeFET based crossbar. Here we introduce a general solution to tackle this problem. For discussion purposes, we assume that only one crossbar array is available to describe challenges and solutions. In practice, we may have multiple crossbar arrays but the nature of the problem remains the same. Practically speaking, because hardware resources are always limited, but the sizes of convolution layers can be arbitrarily large, the problem of insufficient hardware resources always exists.

A. Problem Description

We consider one convolution layer shown in Fig. 2 and one crossbar array. To complete the convolution layer in $O(1)$ time, the crossbar array size should be at least $(C_{\text{in}}{\times}W_F{\times}H_F){\times}C_{\text{out}}$. If the array size is smaller than the minimum size stated above, we need to partition both the workloads (i.e., the convolution computations) and the weights, and map the partitioned workloads and weights to the crossbar. Two challenges must be addressed. **(i)** Even for inference, the crossbar array may need to be re-programmed for different computations after partitioning. As programming is done in a column-by-column manner, it is extremely time-consuming if we frequently re-program the crossbar array. **(ii)** Registers (that consume large area overheads) are required to store intermediate results. Consequently, during the mapping process, execution time, power, register area, etc. should be optimized.

We describe the problem from the matrix point of view, per Fig. 5, where the available crossbar size is $M{\times}N$. The *unrolled* input matrix is a virtual matrix built from the input matrix shown in Fig. 2. It is composed of $W_O{\times}H_O$ blocks, where each block is of size $(C_{\text{in}}{\times}W_F{\times}H_F){\times}C_{\text{out}}$. Each block corresponds to one convolution window on the original input matrix (see Fig. 2). The 3D convolution window is reshaped into a column, and the column is duplicated to all columns in a block (i.e., all columns in one block are identical). Each block produces C_{out} points at the same location on the C_{out} output channels (see Fig. 2). If size of the crossbar array is equal to or greater than $(C_{\text{in}} \times W_F \times H_F) \times C_{\text{out}}$, we just compute column-wise inner products between each block and the crossbar array to obtain the outputs. Otherwise, we need to partition the blocks and weights into tiles with the same size as the available crossbar array. The computation of one tile can be finished in $O(1)$ time, yielding a partial sum of length N by computing column-wise inner products between the tile and the available crossbar array. The partial sums generated by the tiles in the same column in one block need to be accumulated.

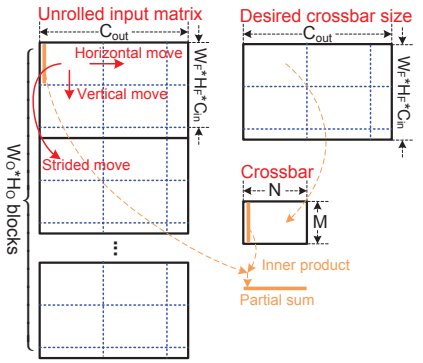


Fig. 5: Illustration of workload partitioning and mapping.

The computation order has a significant impact on the execution time, energy, register area, and the number of re-programming operations. In Fig. 5, after the computation of one tile is finished, we have three choices for the next tile: the tile on the right, the tile below, or the tile at the same location in the next block. We refer to the three schemes as "horizontal move", "vertical move", and "strided move" respectively. Next, we investigate the optimal computation order by deriving the execution time, energy and area requirements.

B. Optimal Computation Order

For both horizontal and vertical moves, the crossbar array needs to be re-programmed for every move. Since horizontal moves need many more registers than vertical moves for storing intermediate results, we only compare vertical moves with strided moves. For strided moves, re-programming is not required for the tiles at the same location in different blocks. This will greatly save programming time and energy. However, once we move to the next tile by a strided move, intermediate results from the previous tile need to be stored in its exclusive registers, which greatly increase the area overhead.

Let S represent the number of register rows available, i.e., the maximum number of strided moves that we can make. Then strided moves reduce the number of re-programming operations by approximately S times. Let T_{clk} be the clock period, P_W be the average write power of one column of the crossbar array, P_R be the average read power of the crossbar array, P_{Reg}, E_{Reg} and A_{Reg} be the static power, write energy and area of a 1-bit register, respectively, and B be the bit width of each column's intermediate result.

Table III estimates the execution time, energy and register area for vertical moves and strided moves. Since typically $N \gg 1$, strided moves reduce the execution time by approximately S times. Strided moves are also beneficial to the crossbar energy (since P_R is typically lower than NP_W). The register energies of the two methods are approximately the same. Consequently, considering performance and energy, strided moves are preferred. The disadvantage is the increased register area. To obtain the best tradeoff, we can adopt the energy-delay-area product (EDAP) to select the optimal S via experiments, which will be discussed in Section V-B.

If we have multiple crossbar arrays, they can be mapped to multiple tiles simultaneously. In this case, we should still use strided moves to reduce time and energy. The physical meaning of a strided move is that once partial sums for one convolution window have been computed, we should move to the next convolution window to compute its partial sums, such that the weights stored in the crossbar array can be reused, instead of continuously accumulating partial sums for

TABLE III: Comparison between vertical moves and strided moves.

	Vertical move	Strided move
Execution time	$\left\lceil \frac{C_{\text{out}}}{N} \right\rceil \left\lceil \frac{W_F H_F C_{\text{in}}}{M} \right\rceil T_{\text{clk}} W_O H_O (1+N) \triangleq T_V$	$\left\lceil \frac{C_{\text{out}}}{N} \right\rceil \left\lceil \frac{W_F H_F C_{\text{in}}}{M} \right\rceil T_{\text{clk}} \left(W_O H_O + \left\lceil \frac{W_O H_O}{S} \right\rceil N \right) \triangleq T_S$
Crossbar energy	$\left\lceil \frac{C_{\text{out}}}{N} \right\rceil \left\lceil \frac{W_F H_F C_{\text{in}}}{M} \right\rceil T_{\text{clk}} W_O H_O \left(P_R + N P_W \right)$	$\left\lceil \frac{C_{\text{out}}}{N} \right\rceil \left\lceil \frac{W_F H_F C_{\text{in}}}{M} \right\rceil T_{\text{clk}} \left(W_O H_O P_R + \left\lceil \frac{W_O H_O}{S} \right\rceil N P_W \right)$
Register energy	$NB \left(T_V P_{\text{Reg}} + \left\lceil \frac{C_{\text{out}}}{N} \right\rceil \left\lceil \frac{W_F H_F C_{\text{in}}}{M} \right\rceil W_O H_O E_{\text{Reg}} \right)$	$SNB \left(T_S P_{\text{Reg}} + \left\lceil \frac{C_{\text{out}}}{N} \right\rceil \left\lceil \frac{W_F H_F C_{\text{in}}}{M} \right\rceil \left\lceil \frac{W_O H_O}{S} \right\rceil E_{\text{Reg}} \right)$
Register area	NBA_{Reg}	$SNBA_{\text{Reg}}$

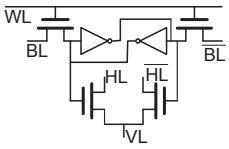


Fig. 6: CMOS-based cell design.

TABLE IV: RC parasitic parameters used in our simulations.

	Cell area (F^2)	R_{wire} (Ω)	C_{wire} (fF)
CMOS	150	0.245	0.059
FeFET	60	0.155	0.037
RRAM (1R*2)	4 ($\times$2) [30]	0.04	0.0096
RRAM (1T1R*2)	20 ($\times$2) [30]	0.09	0.022

the same convolution window. Obviously, the computation order problem is independent of any particular devices so our analysis is universal for any crossbar-based CNN accelerators.

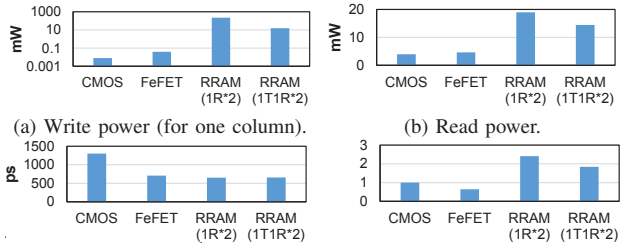
V. Evaluation

We simulated our crossbar design with HSPICE using an FeFET model from [20]. The 10nm fin field-effect transistor (FinFET) PTM (`tfin=8nm`, `hfin=21nm`, `nfin=1`) [21] is adopted for all MOSFET devices. The FE layer thickness T_{FE} is 10.5nm. The crossbar array size is 64×64. We compare our design with RRAM and CMOS equivalents. RRAM-based crossbars have two different structures (1R and 1T1R), and typically two separate crossbar arrays are used. We refer to the two structures as "1R*2" and "1T1R*2", respectively. We use $R_{\text{ON}} = 10K\Omega$ and $R_{\text{OFF}} = 1M\Omega$ for RRAMs [31]. The programming voltage of RRAMs is 2V [30]. For CMOS equivalents, the cell design is shown in Fig. 6; a static random-access memory (SRAM) and two transistors to realize the XNOR function are used. For all designs, the HL and $\overline{\text{HL}}$ voltage is 0.3V when reading. The operating frequency is 100MHz. Unless otherwise noted, area is estimated by assuming that a transistor with the unit width consumes $15F^2$ area. Simulations account for distributed wire parasitics. Table IV shows the wire resistance and capacitance between adjacent cells, which are estimated from cell area.

A. Results of FeFET-based Crossbar Array and Comparisons

As different distributions of inputs and weights can lead to different write and read power, we consider the average case, in which half of the inputs and weights are 1. We randomly select the locations of these 1s and calculate the average power and delay over multiple simulations.

Fig. 7 shows the results of our FeFET-based crossbar array as well as the comparisons with CMOS- and RRAM-based equivalents. The read voltage of FeFETs (V_R, see Fig. 3) is -0.55V. Compared with the two RRAM-based designs, our design reduces write power by 5600× and 395×, and reduces read power by 4.1× and 3.1×. Read latency is 8% higher.



(a) Write power (for one column). (b) Read power.

(c) Read delay. (d) Normalized PDP (for read).

Fig. 7: Results of crossbar arrays and comparisons.

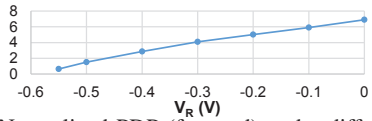


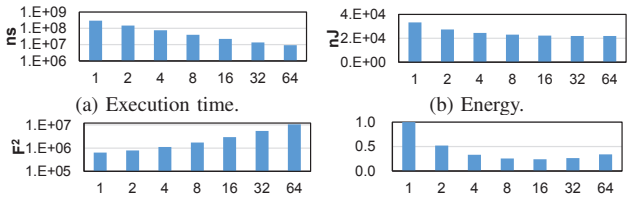
Fig. 8: Normalized PDP (for read) under different V_R.

For the CMOS-based design, we can also set HLs and $\overline{\text{HL}}$s to zero when programming, so the write power is extremely low due to the low programming power of SRAMs. The read power of the CMOS-based design is also the lowest because a MOSFET has lower I_{DS} than an FeFET when they have the same width. However, the read delay of the CMOS-based design is the longest. The FeFET-based design is the best in terms of read power-delay product (PDP).

Fig. 8 shows the normalized read PDP under different V_R values. Data is normalized to the CMOS-based design. The power-delay product increases linearly with V_R, so to obtain a low PDP, we should select a low V_R.

B. Results of Workload Mapping

Now we show the results of workload mapping and computation order. We take one convolution layer ($C_{\text{in}} = C_{\text{out}} = 512$, $W_O = H_O = 32$, and $W_F = H_F = 3$) in VGG-16 [32] as an example. A 64×64 FeFET-based crossbar array is used to compute this layer. We use 6 bits to store intermediate results (i.e., $B = 6$). In order to analyze the impact of the number of register rows (S, which is also the number of strides) for strided moves, Fig. 9 shows the results under different S values. The crossbar array, adders and registers are considered in these results. Note that the adders are shared so we only consider one row of adders, but the registers are not. The results of vertical moves are the same as those for strided moves when $S = 1$. The execution time is significantly reduced when S increases, indicating that the execution time is dominated by re-programming when the crossbar array is not large enough. The total energy is also reduced when S increases, but the reduction ratio is not high, indicating that the read energy is comparable to the write energy. The area



(a) Execution time. (b) Energy. (c) Area. (d) Normalized EDAP.

Fig. 9: Results of strided moves under different numbers of register rows (the horizontal axis is S).

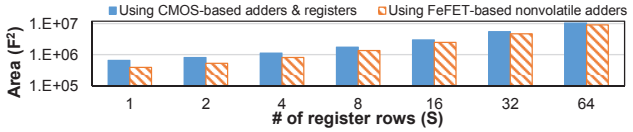


Fig. 10: Area comparison between the cases using CMOS- and FeFET-based adders and registers respectively.

significantly increases when S increases. To make the best tradeoff, we adopt the normalized EDAP, which shows that $S = 16$ is the best choice. In this case, strided moves reduce the EDAP by 4.2× compared with vertical moves. We have also tested other layers in VGG-16 and all tests suggest that $S = 16$ is the best in term of EDAP. In practice, we can test many cases and use voting to select the best S.

In the analysis above, adders and registers are CMOS based. We can also adopt FeFET-based nonvolatile adders [23] for intermediate result accumulation. Using FeFET-based nonvolatile adders will not significantly reduce energy, because registers/adders consume a negligible portion of the total energy. However, area can be saved, due to fused computational logic and storage. When using FeFET-based nonvolatile adders, the adders cannot be shared so S rows of adders are required. Fig. 10 compares the area between the two cases. FeFET-based nonvolatile adders save the area by 13%-40% when the number of register rows varies from 1 to 64.

In summary, our solution for the workload partitioning problem is a hardware-software co-optimization approach. On the software side, we propose strided moves to optimize the computation order; on the hardware side, we optimize the number of register rows to achieve the best tradeoff for performance, energy and area.

VI. Summary and Conclusion

We have proposed FeFET-based crossbars to build hardware accelerators for BCNNs. We have also discussed workload partitioning and mapping when the crossbar array is not large enough, and proposed a universal solution based on hardware-software co-optimization. We have presented both qualitative analysis and quantitative results to demonstrate that our FeFET-based crossbars can improve both the write and read power compared with RRAM-based crossbars. The extremely high saving ratio in the write power indicates that FeFETs will have great advantages in online training of CNNs, which will be considered in future work.

References

[1] M. Courbariaux *et al.*, "BinaryConnect: Training Deep Neural Networks with Binary Weights During Propagations," in *NIPS*, 2015, pp. 3123–31.

[2] M. Courbariaux, I. Hubara *et al.*, "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1," *arXiv preprint arXiv:1602.02830*, 2016.

[3] M. Rastegari *et al.*, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," in *ECCV*, 2016, pp. 525–542.

[4] P. Chi *et al.*, "PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory," in *ISCA*, June 2016, pp. 27–39.

[5] M. Cheng *et al.*, "TIME: A Training-in-memory Architecture for Memristor-based Deep Neural Networks," in *DAC*, 2017, pp. 26:1–26:6.

[6] B. Li *et al.*, "RRAM-Based Analog Approximate Computing," *TCAD*, vol. 34, no. 12, pp. 1905–1917, Dec 2015.

[7] B. Li, L. Xia *et al.*, "MErging the Interface: Power, area and accuracy co-optimization for RRAM crossbar-based mixed-signal computing system," in *DAC*, June 2015, pp. 1–6.

[8] L. Xia *et al.*, "Switched by input: Power efficient structure for RRAM-based convolutional neural network," in *DAC*, June 2016, pp. 1–6.

[9] P. Yao *et al.*, "Face classification using electronic synapses," *Nature Communications*, vol. 8, 2017.

[10] M. Prezioso *et al.*, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.

[11] G. W. Burr *et al.*, "Large-scale neural networks implemented with non-volatile memory as the synaptic weight element: Comparative performance analysis (accuracy, speed, and power)," in *IEDM*, Dec 2015, pp. 4.4.1–4.4.4.

[12] L. Ni *et al.*, "Distributed In-Memory Computing on Binary RRAM Crossbar," *ACM JETC*, vol. 13, no. 3, pp. 36:1–36:18, Mar. 2017.

[13] S. Yu *et al.*, "Binary neural network with 16 Mb RRAM macro chip for classification and online training," in *IEDM*, Dec 2016, pp. 16.2.1–.4.

[14] T. Tang *et al.*, "Binary convolutional neural network on RRAM," in *ASP-DAC*, Jan 2017, pp. 782–787.

[15] S. Salahuddin and S. Datta, "Use of negative capacitance to provide voltage amplification for low power nanoscale devices," *Nano letters*, vol. 8, no. 2, pp. 405–410, 2008.

[16] J. Muller *et al.*, "Ferroelectric hafnium oxide: A CMOS-compatible and highly scalable approach to future ferroelectric memories," in *IEDM*, Dec 2013, pp. 10.8.1–10.8.4.

[17] G. A. Salvatore *et al.*, "Low voltage Ferroelectric FET with sub-100nm copolymer P(VDF-TrFE) gate dielectric for non-volatile 1T memory," in *ESSDERC*, Sept 2008, pp. 162–165.

[18] M. Trentzsch *et al.*, "A 28nm HKMG super low power embedded NVM technology based on ferroelectric FETs," in *IEDM*, Dec 2016, pp. 11.5.1–11.5.4.

[19] S. George *et al.*, "Ncfet based logic for energy harvesting systems," *SRC TECHCON*, 2015.

[20] A. Aziz *et al.*, "Physics-Based Circuit-Compatible SPICE Model for Ferroelectric Transistors," *EDL*, vol. 37, no. 6, pp. 805–808, June 2016.

[21] "Predictive Technology Model." [Online]. Available: http://ptm.asu.edu/

[22] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *ArXiv preprint: 1502.03167*, Feb. 2015.

[23] X. Yin *et al.*, "Exploiting ferroelectric FETs for low-power non-volatile logic-in-memory circuits," in *ICCAD*, Nov 2016, pp. 1–8.

[24] X. Yin, M. Niemier *et al.*, "Design and benchmarking of ferroelectric FET based TCAM," in *DATE*, March 2017, pp. 1444–1449.

[25] S. George *et al.*, "Device Circuit Co Design of FEFET Based Logic for Low Voltage Processors," in *ISVLSI*, July 2016, pp. 649–654.

[26] S. George, K. Ma *et al.*, "Nonvolatile memory design based on ferroelectric FETs," in *DAC*, June 2016, pp. 1–6.

[27] D. Wang *et al.*, "Ferroelectric Transistor Based Non-Volatile Flip-Flop," in *ISLPED*, 2016, pp. 10–15.

[28] X. Li *et al.*, "Advancing Nonvolatile Computing With Nonvolatile NCFET Latches and Flip-Flops," *TCAS-I*, vol. PP, no. 99, pp. 1–13, 2017.

[29] C. Xu, D. Niu *et al.*, "Overcoming the challenges of crossbar resistive memory architectures," in *HPCA*, Feb 2015, pp. 476–488.

[30] C. Xu *et al.*, "Design implications of memristor-based RRAM cross-point structures," in *DATE*, March 2011, pp. 1–6.

[31] H.-L. Chang *et al.*, "Physical mechanism of HfO2-based bipolar resistive random access memory," in *VLSI-TSA*, 2011, pp. 1–2.

[32] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, 2014.