A Ferroelectric FET based Processing-in-Memory Architecture for DNN Acceleration

Yun Long, *Student member, IEEE*, Daehyun Kim, *Student member, IEEE*, Edward Lee, *Student member, IEEE*, Priyabrata Saha, *Student member, IEEE*, Burhan Ahmad Mudassar, *Student member, IEEE*, Xueyuan She, *Student member, IEEE*, Asif Islam Khan, *Senior member, IEEE*, and Saibal Mukhopadhyay, *Fellow, IEEE*

***Abstract*—This paper presents a Ferroelectric FET (FeFET) based processing-in-memory (PIM) architecture to accelerate inference of deep neural networks (DNNs). We propose a digital in-memory vector-matrix multiplication (VMM) engine design utilizing the FeFET crossbar to enables bit-parallel computation and eliminate analog-to-digital conversion in prior mixed-signal PIM designs. A dedicated hierarchical network-on-chip (H-NoC) is developed for input broadcasting and on-the-fly partial results processing, reducing the data transmission volume and latency. Simulations in 28nm CMOS technology show 115x and 6.3x higher computing efficiency (GOPs/W) over desktop GPU (Nvidia GTX 1080Ti) and ReRAM based design, respectively.**

***Index Terms*—Ferroelectric FET, Processing in memory, DNN**

I. Introduction

THE wide-spread adoption of deep neural networks (DNNs) in solving complex problems in various domains have inspired the design of many dedicated hardware accelerators [1-3]. However, as DNN models become deeper and require more parameters, the time/energy cost of moving data between memory and logic starts limiting the efficiency of the computing. Memory rich architecture integrates large amount of on-chip memory to reduce the DRAM access [2, 3]. Near-memory-processing (NMP) architecture embeds logic engines within off-chip memory to reduce the cost of data-movement [4, 5]. A more aggressive approach is to directly perform computation inside memory, often referred to as the processing-in-memory (PIM) architectures [6-11]. The PIM architectures are designed to perform vector-matrix-multiplication (VMM) within the memory (i.e. VMM-in-memory). The current summation at the bit-line is used to perform multiplication-accumulation (MAC) operation, resulting in very high throughput. The resistive random access memory (ReRAM) crossbar based in memory VMM engine design has shown promise of high energy-efficiency, thanks to the zero-leakage storage, high-density (1T cells), and non-volatility of ReRAM devices [6-8]. However, recent studies have noted challenges associated with ReRAM's high write power, long read latency, and challenges in driving large crossbar arrays with many parallel ReRAM devices [11]. Moreover, due to the nature of analog computation for current summation, analog/digital conversion (ADC and DAC) is necessary for data conversion, leading to high power/area overhead. As an alternative, some prior PIM based designs utilize DRAM [9] or SRAM [10] to performs basic logic operation (such as NOR and AND) inside the memory rather than current summation, therefore, avoiding ADC/DAC. However, such approaches suffer from large leakage current and reduced compute density due to their low level logic abstraction.

This paper presents a ferroelectric FET (FeFET) based PIM architecture for *in-memory* vector-matrix computation to accelerate DNN inference. Our design is built on three core concepts:

- We employ FeFET as the basic memory cell. Compared with ReRAM, FeFET presents much less read latency (less RC delay) and ultra-low programming energy while keeps similar density and non-volatility.
- We exploit gate-driven operation of FeFET to design all-digital VMM engine, eliminating ADC/DAC while ensuring high throughput.
- We present a scalable micro-architecture by connecting multiple *VMM* engines using a hierarchical network-on-chip (H-NoC) with in-router accumulator, reducing the data transmission volume and latency.

A chip-scale architecture is developed using our VMM-in-memory fabric coupled with specialized functional blocks, on-chip storage. A dedicated execution flow and software-hardware interface is developed to improve the system flexibility. The proposed system is implemented in 28nm CMOS technology. The area, power, and frequency of operation are simulated using post-layout analyses of blocks and network. The cycle-level simulation is used for application level performance analyses for different Convolutional and Recurrent Neural Network (CNN and RNN) models. Our design demonstrates 115x and 6.3x higher computing efficiency (GOPs/W) over GPU (Nvidia GTX 1080Ti) and ReRAM based PIM design, respectively.

This material is based on work supported in part by National Science Foundation (NSF) (#1810005). All authors are with School Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332 USA. Corresponding author: Yun Long. Email: yunlong@gatech.edu.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JXCDC.2019.2923745, IEEE Journal on Exploratory Solid-State Computational Devices and Circuits

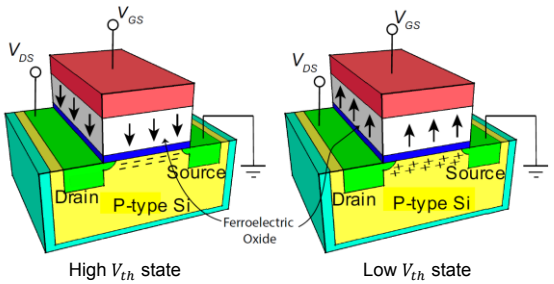


Fig. 1. (a) FeFET structure and its on/off state.

II. FeFET Background

Various memory techniques have been explored for PIM based designs including DRAM [9], eDRAM [2, 3], SRAM [10], ReRAM [6, 7]. Among these memory techniques, ReRAM attracted lots of research attentions recently and shows promise for very high energy-efficiency. However, recent studies also note challenges associated with ReRAM's high write power, long read latency, and power/area overhead for driving large arrays [11].

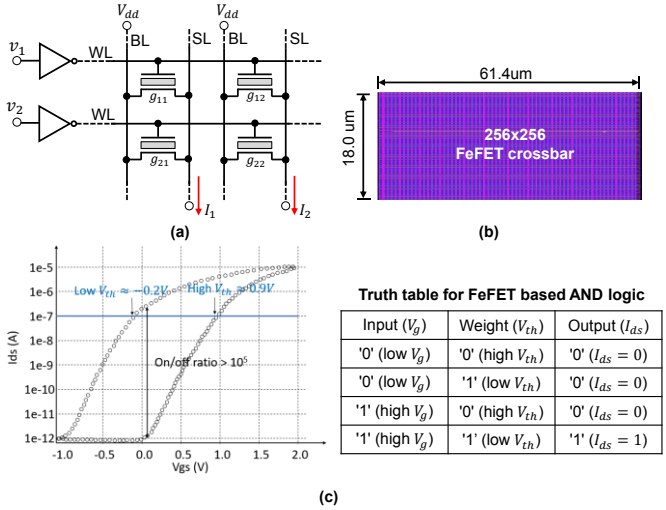
To address the challenges presented in ReRAM, we propose to use FeFET as an alternative memory solution for PIM architecture. FeFET is a transistor in which the ferroelectric oxide layer is included in the gate dielectric stack, as shown in Fig. 1. A ferroelectric oxide is an insulator which exhibits a spontaneous electric polarization in the absence of electric field. The direction of the polarization can be switched by applying a voltage larger than the coercive voltage on the gate terminal of FeFET [12]. When the polarization is pointing downwards, channel is in inversion, bringing the transistor into the 'ON' state (i.e. low Vth state). Similarly, if the polarization is pointing upwards, channel is in accumulation which gives the transistor 'OFF' state (i.e. high Vth state).

It has already been demonstrated that Hafnium oxide FeFET has good temperature stability, writing endurance, data retention and switching speed/energy [12-14]. The ultra-low writing energy due to the unique electrical field effect switching mechanism is the most prominent feature which distinguish FeFET from other emerging technologies. Further, unlike ReRAM which presents as resistive loads for reading, FeFET is gate-driven (i.e. capacitive load), eliminating the large RC delay in ReRAM case and reducing the read latency.

Besides utilizing FeFET as non-volatile memory [12-14], there have been a few recent works exploring FeFET based logic (AND, OR, etc.) design [15], oscillator design [16], spiking neural network [17], and binary neural network acceleration (using 4 FeFET cells for XNOR logic) [18]. These works focus on device/crossbar modeling and lack of system/architecture level design.

III. FeFET Crossbar Design

With FeFET crossbar as the core memory element, e*ach cell in the crossbar performs a 1-bit multiplication*. Fig. 2(a) shows the configuration of the FeFET crossbar, where gate, drain, source of the transistors are connected to WL, BL and source line (SL), respectively. Fig. 2(b) shows the corresponding



Truth table for FeFET based AND logic

Input (V_g)	Weight (V_{th})	Output (I_{ds})
'0' (low V_g)	'0' (high V_{th})	'0' ($I_{ds}=0$)
'0' (low V_g)	'1' (low V_{th})	'0' ($I_{ds}=0$)
'1' (high V_g)	'0' (high V_{th})	'0' ($I_{ds}=0$)
'1' (high V_g)	'1' (low V_{th})	'1' ($I_{ds}=1$)

Fig. 2. (a) Configuration of FeFET crossbar. (b) Layout view of a 256x256 crossbar under 28nm technology. (c) Measured Ids-Vgs characteristics from a FeFET device. Measurement data are extracted from [13] with the transistor size 30nm x 80nm. FeFET based 1-bit multiplication (i.e. AND logic).

layout view of a 256x256 crossbar under 28nm technology (the layout is based on normal MOSFET). The left side of Fig. 2(c) shows the measured Ids-Vgs characteristics [13]. Two distinct threshold voltage (-0.2V and 0.9V) are observed and the on/off ratio is more than 10^5. One should note that with our FeFET crossbar configuration, device can be programmed in a row-wise fashion [19].

For computation, weights are stored as transistor channel conductance (i.e. threshold voltage) and input vectors are used to drive WLs (i.e. transistor gate). We employ the FeFET based AND logic [15] to perform the 1-bit multiplication. One-bit of weight is encoded as high V_{th} or low V_{th}, representing either 0 or 1, respectively; similarly, 1 bit of input vector can be encoded as high or low WL voltage (V_{gs}). When the input bit is 0 (i.e. low V_{gs}), the current is always 0 with either high V_{th} or low V_{th} since the transistor is turned off. On the other hand, if the input bit is 1 (i.e. high V_{gs}), the transistor is still off when V_{th} is high but turns on when V_{th} is low. The large on/off ratio of FeFET, thanks to its steep sub-threshold slope (<60 mV/Dec) [20], creates large difference between the output '1' current and output '0' current.

A key advantage of our design is that now the WL connects to transistor's gate, which is a capacitive load. Therefore, there is no word line voltage drop issue (RC delay) as in the ReRAM scenario. Moreover, the drain voltage is fixed at 1V, which is the system supply voltage. One should note that the read disturb effect (minor polarization loop caused by reading operation) is less concerned and not explored in our desgin since both the pulse width and amplitude of the gate voltage is much smaller than the switching voltage reported in recent works [13, 21].

IV. VMM Engine Micro-Architecture

In this section, we present two configurations of the all-digital VMM engine to realize an ADC free design.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JXCDC.2019.2923745, IEEE Journal on Exploratory Solid-State Computational Devices and Circuits

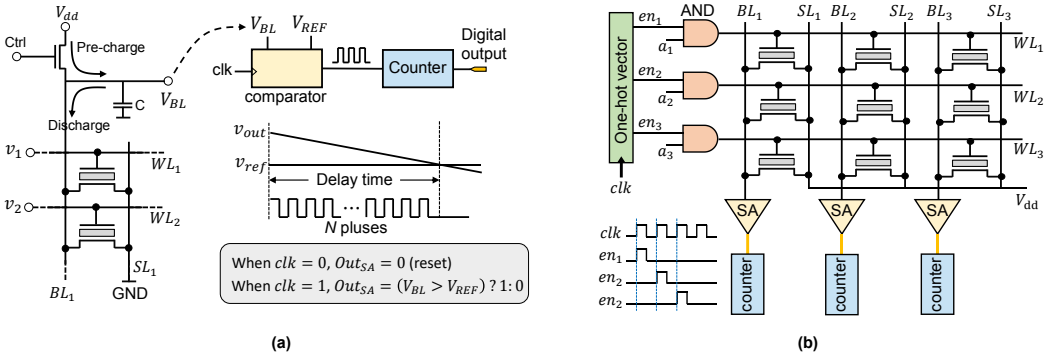


Fig. 3. (a) Configuration 1: SA + counter based TDC VMM engine design. (b) Configuration 2: Row-by-row read and accumulation based VMM engine design.

A. *Configuration 1: SA and Counter based TDC*

We propose a pre-charge/discharge approach based VMM engine design as shown in Fig. 3(a). First, the BL is pre-charged to the supply voltage V_{dd}. Then, during computing, depending on how many devices in the same column are turned on, the BL voltage drops with different speed. The comparator (sense-amplifier based) is used to sample the difference between the reference voltage V_{REF} and V_{BL} periodically (controlled by a clock signal clk). When clk is low, the output is 0 (reset). When clk is high, the output of comparator is 1 if $V_{BL} > V_{REF}$, or 0 if $V_{BL} < V_{REF}$. Therefore, within 1 clock cycle, if V_{BL} is larger than the reference voltage, the comparator generates a pulse; if not, the output of comparator remains 0. A counter accumulates the number of pulses from comparator. Basically, with a simple sense amplifier and counter, we realize the time to digital converting (TDC). Simulation in 28nm CMOS shows that our design consumes 2.7x less power than using ADC [7], while achieving same speed.

B. *Configuration 2: Row-by-Row read and accumulation*

The ADC (or TDC in configuration 1) is inevitable if multiple WLs are simultaneously activated. To eliminate it, we propose to activate single WL per clock cycle. As shown in Fig. 3(b), The first part of WL peripheral is a clock-driven one-hot vector unit. At the first clock cycle, the enable signal for the top WL (en_1) is turned on. Then, at the second clock cycle, en_1 disabled and en_2 is enabled, and so on. The second part of WL peripheral is a logic gate which performs AND operation between the enable signal (en_i) and input vector (a_i). Only when both en_i and a_i are high, the value stored in corresponding memory cells b_i are sensed out. At BL peripheral, sense amplifier (SA) is employed to sense out the value (either 0 or 1) and send it to the counter at each clock cycle. Essentially, the MAC operation is performed with N cycles where N is the number of rows in the memory array.

While sacrificing the parallelism of analog computing, our design is faster than ADC based approach. This is due to the fact that for prior ADC based design, the speed of ADC is the major throughput bottleneck. For example, in ISAAC [7], a 1.3 Giga-samples-per-second (GSps) SAR ADC is employed and shared by a memory crossbar. It takes 100ns to convert the analog values for a 128×128 (the crossbar size in ISAAC) memory array. In our design, it also takes 128 clock cycles to perform the same computing (loop through all the rows). For a 128×128 FeFET array (2KB), our simulation indicates the internal clock frequency can go up to 4GHz (reading frequency, not programming) in 28nm technology, resulting 25.6 ns to perform the MAC operation, ~4x faster than ADC based solution.

C. *Comparison of the two configurations*

While the proposed two configurations have very different reading/sensing scheme, the circuit level implementation are similar. To be more specific, the sensing is realized by a sense amplifier (called comparator in configuration 1) and a counter. In essence, with the same system implementation, we can realize these two configurations simply by changing the control signal (i.e. the WL activations) patterns.

In terms of speed, the latency for configuration 1 is determined by the clock frequency (reference voltage V_{REF} needs change accordingly) of the comparator and counter. For a FeFET crossbar with N rows, the counter needs to wait at least N clock cycle to ensure the output has N levels. Similarly, the latency for configuration 2 is determined by the memory clock as the sensing is performed row-by-row. For a N rows crossbar, N clock cycles are necessary to get the final output.

In terms of accuracy, configuration 2 is preferred. In configuration 1, even though the sensing is digital, the BL discharge is still in analog domain. Further, the device variation, cell leakage, nonlinearity of BL discharge and temperature/voltage fluctuation can introduce computing error. On the other hand, the MAC in configuration 2 is in digital domain, providing much better fidelity in terms of computing accuracy.

V. Data Communication Network

A. *VMM engine for large scale matrix operation*

Fig. 4 illustrates the methodology to partition a large matrix-matrix multiplication across multiple VMM engines. Assuming the 1 VMM engine can hold parameters of size s×s, the weight matrix is then partitioned into several small segments with the granularity of s×s. In total, n×m VMM engine will be used (Fig. 4). Similarly, the input matrix is first transposed, partitioned and sequentially fed into the corresponding VMM engines.

From Fig. 4, we observe that each input segment is shared across multiple VMM engines horizontally (e.g. VMM11,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JXCDC.2019.2923745, IEEE Journal on Exploratory Solid-State Computational Devices and Circuits

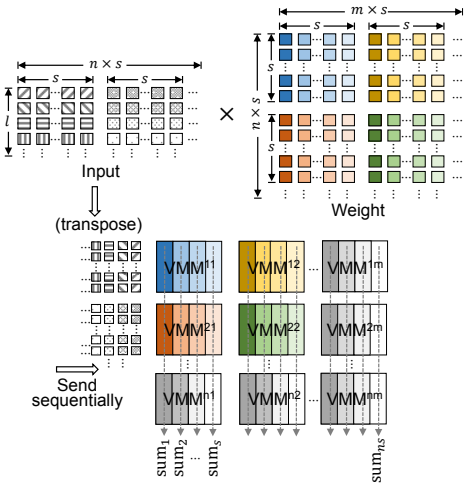


Fig. 4: Matrix partition and mapping to multiple VMM engines. Different color and shade are used to help tracking the input and weight mapping.

VMM12, till VMM1m). We call it as **row-wise input sharing**. On the other side, partial results generated from the same column of multiple VMM engines should be summed together vertically (e.g. VMM11, VMM21, till VMMn1 in Fig. 4) since they belong to the same column in the original weight matrix. We call it as **column-wise output summation**.

B. Hierarchical Network-on-Chip (H-NoC) Design

We propose a hierarchical NoC to address the discrepency between row-wise input sharing and column-wise output summation (Fig. 5). At the bottom level, 4 VMM engines share a router. Then, 4 such routers are connected to a router in the higher level. Since the VMM engines organized in a hierarchical fashion, a system with N level of routers can accommodate up to 4^N VMM engines. Fig. 5(b) shows the router design, containing five input/output ports and corresponding I/O buffers. A 5×5 switching matrix is equipped to route input/output ports and the routing is based on

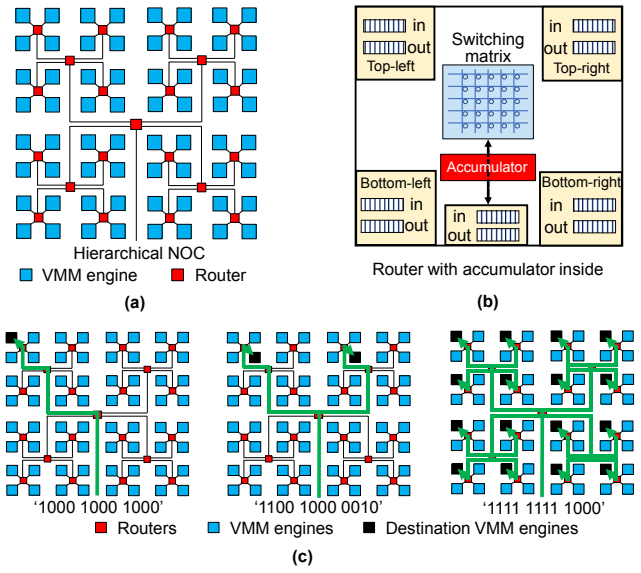


Fig. 5: (a) Hierarchical network-on-chip. (b) Router design with accumulator integrated. (c) Three different data forwarding patterns and corresponding addresses, including one-to-one forwarding and broadcasting.

store-and-forward (SAF) approach. Distinguished from conventional router designs, we insert a computing block (i.e. accumulator) inside the router to enable on-the-fly partial results summation. The benefits of H-NoC are in two-fold.

First, H-NoC realizes efficient row-wise input sharing. Fig. 5(c) illustrates three different data forwarding patterns. The first example shows the one-to-one forwarding. The top-level router decodes the first 4-bit of a packet (each bit represents the on/off of *top-left*, *top-right*, *bottom-right*, *bottom-left* output ports, e.g. '1000' means the packet goes to its top-left branch) and sent the packet to its sub-level router. Then the sub-level router decodes the next 4-bits and repeats until the packet arrives the designated VMM engine at the top-left corner. Besides one-to-one forwarding, the packet can be broadcast. As shown in the last example of Fig. 5(c), since the first 4-bit address is '1111', the top-level router broadcasts the packet to its sub-level routers in four directions. This process repeats and finally a single packet is assigned to 16 distributed VMM engines simultaneously.

A case study is used to illustrate how the *row-wise input sharing* benefits from the input broadcasting. As shown in Fig 6(a), a large weight matrix is first partitioned into several segments (we show $2 \times 8 = 16$ segments, more details about matrix partition and mapping are discussed in supplementary materials). Then, we map W_{11}, W_{21}, W_{31}, W_{41} to 4 VMM engines sharing the same router node ❹. Then, input vectors are sent to corresponding VMM engines (input sharing and reuse). For example: I_4 (in blue) should go to two VMM engines which store W_{41} and W_{42}. Conventionally, this requires two packets and two cycles since there are two destination VMM engines. With H-NoC, this can be done with a single packet and one cycle. As shown in Fig. 6(b), router ❶ decode the first 4-bit address (1100) and then broadcasts I_4 to its sub-level router at top-left and top-right directions (i.e. sends packet to routers ❷ and ❸). These two routers then decode the next 4-bits (1000) and sent the packet to their top-left router ❹ and ❻. Finally, the packet goes to the bottom-right leaf VMM engines of router ❹ and ❻.

Second, **H-NoC is dedicated for efficient column-wise partial results summation**. Enabled by the in-router accumulator, the results summation is performed on-the-fly, i.e. output summation happens during data transmitting. Again, we use the case in Fig. 6 as an example. It takes two steps to get the

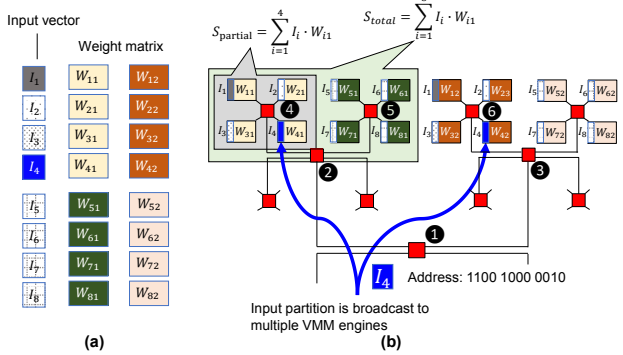


Fig. 6: A case study to illustrate how data are mapped via H-NoC. We use different color and shade to help tracking the input and weight mapping.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JXCDC.2019.2923745, IEEE Journal on Exploratory Solid-State Computational Devices and Circuits

summation ($S_{total} = \sum_{i=1}^{8} I_i \cdot W_{i1}$). First, router ❹ and ❺ works independently and parallelly, each receiving four partial results from the connected VMM engines and summing the partial data utilizing the built-in accumulator (i.e. $S_{partial} = \sum_{i=1}^{4} I_i \cdot W_{i1}$ and $S_{partial} = \sum_{i=5}^{8} I_i \cdot W_{i1}$). Router ❷ then accumulates the partial results from ❹ and ❺ and sends the final summation to global buffer. Therefore, rather than sending each partial result to the global buffer as separate packets, only 1 packet is sent to the global buffer leveraging the on-the-fly/parallel processing enabled by H-NOC.

Depending on how many VMM engines are involved for one matrix computing, this process repeats until all the partial results are summed together. As routers in the same level are working in parallel, the worst-case latency is limited to $4 \times$ number of router levels, since it takes 4 clock cycles for a router to accumulate partial results from its 4 branches.

VI. Chip-Scale Architecture

A. System Architecture

Fig. 7(a) shows the system architecture with VMM engines interconnected with H-NoC. We implement several fixed function units to support computation that cannot be accelerated within the VMM-engines, such as element-wise multiplication and activation functions. This ensures our design has the flexibility to support various CNN and RNN models. For example, multiplier array and adder array are used for element-wise multiplication and addition, respectively. Additionally, they combined together can be used to compute the Taylor series of some special functions such as sigmoid. Since our design accelerate DNN inference, both adder and multiplier have 16-bit fixed point precision. Pooling processor is used to perform average or max pooling and ReLU is for ReLU layer. There are several other fixed function units, such as max value search and divider (used in batch normalization layer). The last component in the system architecture is the micro-processor which fetches/decodes the instructions and coordinates the data accessing and transmission.

B. Execution Flow

The first layer of AlexNet is used as an example to illustrate the execution flow (Fig. 7(b)). The input feature maps to this layer is $224 \times 244 \times 3 \times 1$, representing the image width, height, RGB channels, and mini-batch size, respectively. The Convolutional kernel size is $3 \times 11 \times 11 \times 96$, corresponding to the number of input channels, kernel width/height, and output channels, respectively. The micro-processor calculates how many memory sub-arrays are required to perform the computation. Assuming the memory sub-array size is 256×256, each set of the convolution kernel (i.e. $3 \times 11 \times 11$) contains 363 weight parameters, and thus, $\lceil 363/256 \rceil = 2$ crossbar arrays to perform the dot-production for one kernel. Further, kernels can be padded horizontally to achieve parallelism. Since there are 96 kernels, in total we need $96 \times 8 \div 256 \times 2 = 6$ memory sub-arrays, given that each element is a 8-bit number (256 devices in a row can hold 32 8-bit numbers). Then, the micro-processor will wrap the received data into discrete packets (the first few bits of a packet contains the routing address) and dispatch them to the target memory sub-array locations via H-NoC. After the computing is done, results are collected back and sent to the activation/pooling function units through system bus.

Eventually, the output feature maps are generated and stored to the memory for temporary storage since they will be used as inputs for the next layer. Preferably, these data are stored in on-chip memory if there are available space; alternatively, if all the on-chip memory sub-arrays are programmed with weights, the micro-processor will offload the temporary data off-chip. Typically, a single layer parameter size is less than 2MB, the offload of temporary data rarely happens for our benchmark CNNs and RNNs

C. Software/hardware interface

A software/hardware interface is designed to bridge the gap between software and hardware, letting users easily deploy their applications without specific hardware knowledge. As shown in Fig. 7(c), the runtime system takes DNN definition file (as well as pre-trained model if available) as input, sets the computing model and running precision, and performs layer-wise interpretation to translate the high-level DNN model definition to the instructions we developed for the proposed system. There are three types of instructions: **control, layers, and parameter specification**. Instructions are 64-bit with the first 6-bit as Opcode. Control instructions are used to define computing precision, set running mode (only inference available now, supporting training is our future work) and write

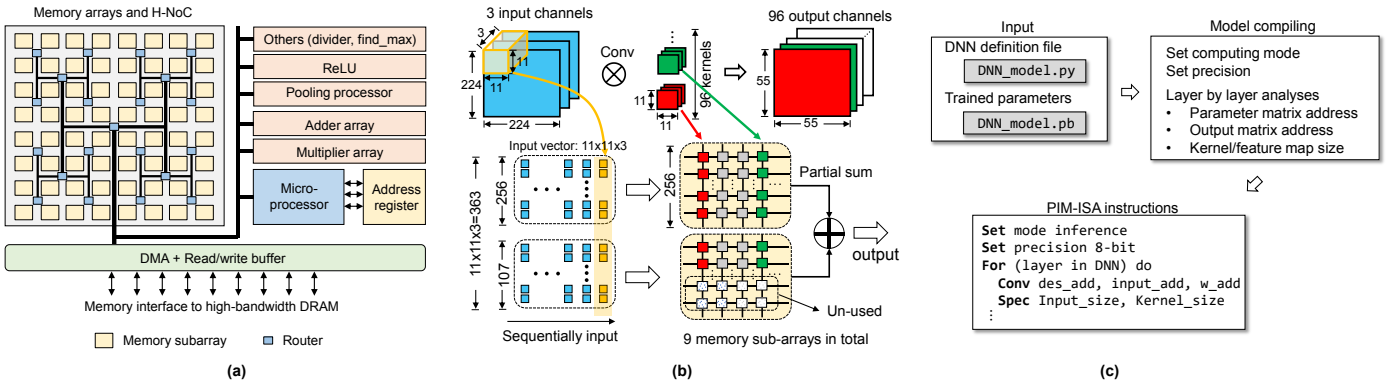


Fig. 7: The chip-scale architecture of the FeFET-based PIM design: (a) system architecture, and (b, c) execution flow.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JXCDC.2019.2923745, IEEE Journal on Exploratory Solid-State Computational Devices and Circuits

address register. Layers instructions define the layer and where the weight and activation should be fetched from. Parameter specification instructions are always attached to the layer instructions, specifying more information about the layer defined by the previous instruction. For example, to define the computation of a convolutional layer, we need two instructions, one for layer specification which defines the layer type and where we should read the weight and input from (i.e. weight and input address); the other one defines the convolutional kernel size, input/output feature map depths, etc. Details about instruction design and more examples are available at supplementary materials.

VII. Simulation Results

A. Prototype design

The prototype design contains 2048 VMM engines organizing with a 6 levels H-NoC. Each VMM engine contains a 256x256 FeFET memory crossbar together with the peripheral circuitry. In addition, at system level, we implemented several functional blocks such as multiplier/adder arrays, pooling processor, and ReLU units. We performed SPICE simulation with 28nm CMOS technology (normal MOSFET model with calibrated threshold voltage and transistor size to mimic the I-V characteristic of real FeFET measurement data [13]) using extracted netlist of the crossbar together with the WL drivers and SAs to estimate power and latency of the memory sub-array. The SPICE simulation is then coupled with synthesized digital blocks (such as counters, H-NoC, functional blocks and controller) to form a completed chip-level modeling. While setting the system clock to be 1GHz, the FeFET memory sub-array can run at a higher clock frequency. Therefore, for configuration 1 (SA + counter based TDC approach), we set the clock to be 2GHz. Similarly, for configuration 2 (row-by-row read and accumulation approach), the memory crossbar also has an internal 2GHz clock. The off-chip memory bandwidth is set to be 512GB/s which is same with TPU-v2 [22]. The key design specification and the layout view for a VMM engine is presented in Fig. 8. One should note that we use the same circuit implementation but changing the WL activation pattern (parallel versus row-by-row access) to realize the two different

Design specification for the prototype implementation

Configurations	Config 1 (SA+counter based TDC)	Config 2 (row-by-row accumulation)
Computing domain	Mixed signal	Digital
# of VMM engine (memory capacity)	2048 256x256 FeFET crossbar (16 MB)	
Peak throughput	16.38 TOPs (@ clk_mem: 2 GHz with 8-bit precision)	
DRAM bandwidth	512 GB/s	
VMM power *	5.8 mW	
System power	18.2 W	
Technology	28 nm CMOS	

* For config 1, we ignore the power for reference voltage generation.

VMM engine layout view

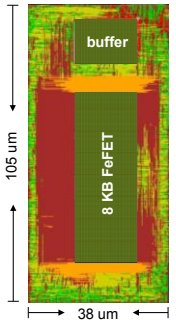


Fig. 8: Design specification for the prototype implementation and the layout view for one VMM engine. Two configurations share the same circuit implementation.

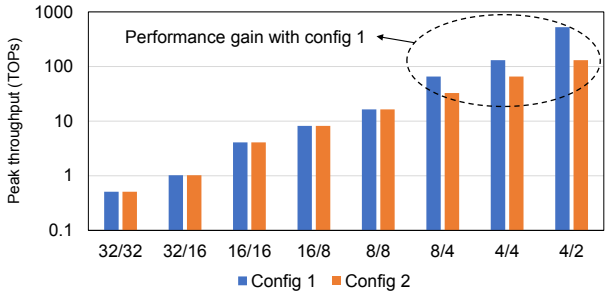


Fig. 9. Peak throughput with different weight and activation precision. 32/16 means the weight is 32-bit while the activation is 16-bit.

configurations. Also, we ignore the power overhead of the reference voltage generation circuit in configuration 1 since it can be shared across the system. Therefore, the power of VMM engine for these two configurations are also similar.

B. Benchmarks and precision

We have 5 different types of DNN models (AlexNet, GoogleNet, VGG-16, VGG-19, and LSTM) with varying parameter size and computing complexity (i.e. GOPs).

We also explore the system peak performance with different bitwidth of DNN weights and activations. As illustrated in Fig. 9, with less bit-precision, the throughput is higher. We also note that when the activation is less than 8-bit (which is the precision for 256 rows FeFET crossbar), configuration 1 becomes faster as it takes a smaller number of clock cycles to accumulate the sensing result (A more detailed analyses for this observation can be found in supplementary material).

In the following experiments, we assume weights and activations have 8-bit precision because we observe that for the benchmark DNN models, 8-bit is good enough to ensure the inference accuracy. One should note that the state-of-the-art DNN models (such as ResNet [23] and MobileNet [24]) typically requires 16-bit or even floating point for the best accuracy. Supporting flexible bit-precision and floating point operation is our future work.

C. Performance analyses

First, for data transmission efficiency, we compare our H-NoC design with the naive approach (no input broadcasting/reuse or output on-the-fly processing) and ISAAC-like design (using two stage hierarchical buffer for output accumulation) [7]. Fig. 10(a) shows the data (input, weights, and internal temporary data) transmission latency for processing one image using 4 different benchmark CNNs. On average, our design reduces the latency by 14.5x and 6.7x over the naive approach and ISAAC-like design across the benchmark CNNs, respectively.

Second, we analyze the power efficiency of FeFET VMM engines and compare with ReRAM baseline design, as illustrated in Fig. 10(b). We first consider using ADC in the BL peripherals and insert buffer to drive the WL (i.e. resistive load). With a simple technology replacement from ReRAM to FeFET (using the same peripherals), we observe that the FeFET based design achieves only 1.2x power reduction because the power consumption on the peripherals dominated. With the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JXCDC.2019.2923745, IEEE Journal on Exploratory Solid-State Computational Devices and Circuits

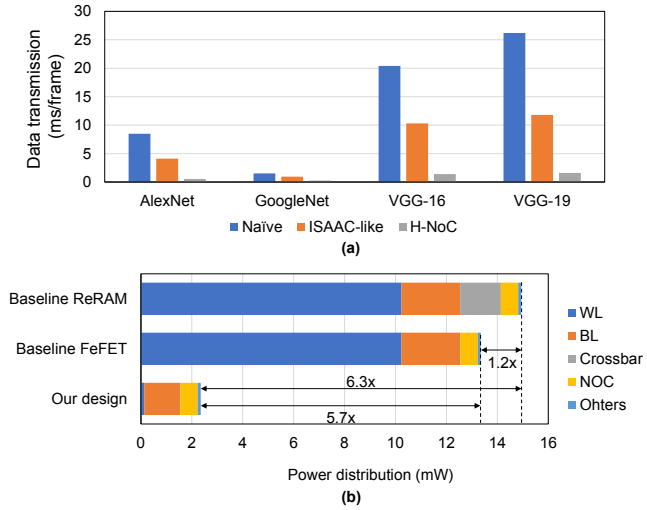


Fig. 10. System performance improvements for (a) H-NoC for data transmission and (b) VMM engine for computation.

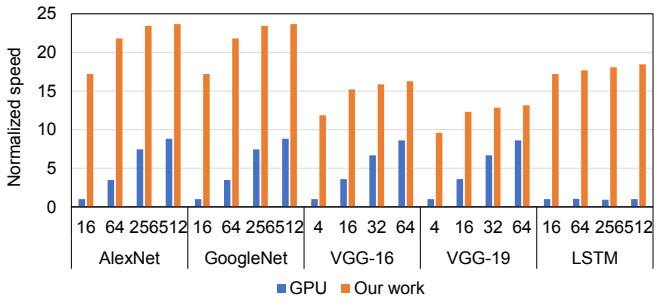


Fig. 11. Normalized inference speed of desktop GPU and our design for DNN models with varying batch sizes. For VGG-16 and VGG-19, we choose smaller batch sizes to avoid the *run out of memory* error.

optimized digital-like peripherals (i.e. replace the power-hungry ADC with SA + counter and also eliminate the WL buffer since FeFET is a capacitive load), significant power efficiency improvement is observed (another 5.7x). In total, with the cross-cutting solutions combining emerging device technologies and circuit innovations, FeFET based VMM engine demonstrates 6.3x power efficiency over the baseline ReRAM design.

We then evaluate the overall system performance using our benchmark DNN models and compared with the measured data from desktop GPU (Nvidia GTX 1080Ti with 11.3 TFLOPs throughput and 250 W power). Fig. 11 shows the speed (normalized) comparison for different DNNs under varying batch sizes. We don't differentiate the two VMM engine configurations because they have similar throughput when using 8-bit precision. We observe that our design outperforms GPU solution by 8.4x in terms of frames per second (fps). Additionally, desktop GPU's power is 13.7x higher than out work, resulting up to 115x computing efficiency (GOPs/W) improvement with our design.

D. *Computing accuracy*

The device variation of FeFET can potentially impact the computing accuracy. Similar with prior ReRAM based design, we use Gaussian noise to represent the stochastic device variation [25]. We calibrate our device variation model with experimental FeFET data in recent published works [13, 14]. The typical variation (the standard deviation: σ) varies from 1% to 20%.

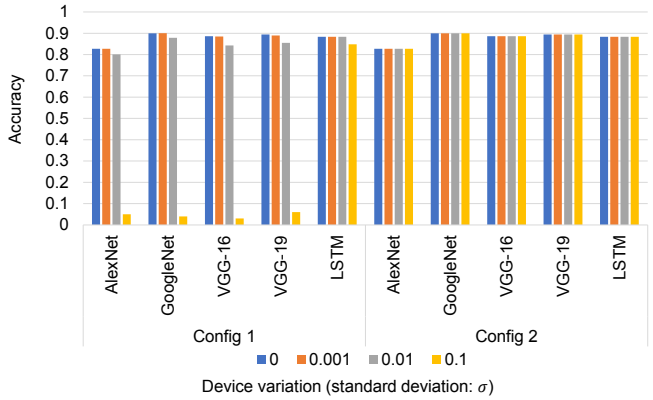


Fig. 12. Computing accuracy under different level of device variation. The variation is characterized with Gaussian noise.

Fig. 12 shows the classification accuracy deterioration under device variation considering the two proposed VMM engine configurations. As mentioned earlier, the first configuration eliminates the ADC but still perform part of the computing in mixed-signal domain, thus, it is more vulnerable to device variation. On the other side, thanks to the large on/off ratio of FeFET, the second configuration demonstrates good robustness towards the device noise.

E. *Comparison with other works*

We perform a detailed comparison between existing DNN accelerators implemented with ASIC [2, 22], NMP [4, 5], and PIM architecture [7, 10]. For ASIC based solution, we consider DaDianNao [2], which integrates large amount of on-chip eDRAM to store DNN parameters and TPU-v2 [22], the second generation of tensor processor unit from Google. For NMP, we investigate DeepTrain [5], a novel architecture which integrates logic layer into the high-bandwidth DRAM. We also compare with ISAAC [7], a pioneer work for ReRAM based PIM architecture for DNN acceleration. For SRAM based design, we evaluate a recent work, neural-cache [10], a bit-serial logic-in-memory based DNN accelerator architecture. At last, we compare with a recent FeFET based design [26] which utilize FeFET as analog synapse (each device stores 5-bit) for DNN training acceleration. The key design features are summarized in Table I. In terms of computing efficiency, we evaluate from two different aspect, namely, array-level and system level. For array-level efficiency, only the energy consumed by the array (device and crossbar peripherals) is considered.

As a conclusion, our design achieves the state-of-the-art performance with 896 GOPs/W computing efficiency using 8-bit precision. Our design outperforms other PIM architectures by leveraging the following merits: (1) The key horse power comes from the VMM engine which eliminates the slow/power-hungry ADC, plus the high memory clock frequency. (2) H-NoC helps to reduce the data movement latency for both input reuse and output collection. (3) With FeFET as the memory

TABLE I
PERFORMANCE COMPARISON WITH OTHER DNN ACCELERATORS

	Technology	Hardware	Parameter storage	Power (W)	Area (mm^2)	Efficiency (array-level)	Efficiency (system-level)	Peak throughput
DaDianNao [2]	28 nm	ASIC	eDRAM (on-chip)	20.1	67.7	--	286 GOPs/W	5.7 TOPs
TPU-v2 [20]	--	ASIC	DRAM	~ 250	--	--	180 GOPs/W	45 TOPs
DeepTrain [5]	15 nm	NMP	DRAM	7.2	--	--	566 GOPs/W	7.2 TOPs
ISAAC [7]	28 nm	PIM	ReRAM	65.8	85.4	604 GOPs/W	381 GOPs/W	25.1 TOPs
Neural Cache [10]	28 nm	PIM	SRAM	52.9	--	529 GOPs/W	--	28 TOPs
Analog-FeFET [24]	--	PIM	FeFET	--	--	840 GOPs/W	--	--
Our work	28 nm	PIM	FeFET	18.2	49.6	1234 GOPs/W	896 GOPs/W	16.38 TOPs

cell, we also benefit from the dense cell structure and low read latency/write energy.

VIII. CONCLUSION

In this work, we propose FeFET based PIM architecture to accelerate DNN inference. With FeFET as the basic memory cell and ADC free VMM engine design, the computing efficiency is significantly enhanced. A dedicated hierarchical network-on-chip is developed to realize fast and parallel data communication. As FeFET continues to mature towards a commercial technology, we show the pathway to a high-efficient architecture that successfully leverages unique properties of this technology to accelerate challenging data-intensive computing applications.

REFERENCES

[1] S. Han *et al.*, "EIE: efficient inference engine on compressed deep neural network," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, 2016, pp. 243-254: IEEE.

[2] Y. Chen *et al.*, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 609-622: IEEE Computer Society.

[3] Z. Du *et al.*, "ShiDianNao: Shifting vision processing closer to the sensor," in *ACM SIGARCH Computer Architecture News*, 2015, vol. 43, no. 3, pp. 92-104: ACM.

[4] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, 2016, pp. 380-392: IEEE.

[5] D. Kim, T. Na, S. Yalamanchili, and S. Mukhopadhyay, "DeepTrain: A Programmable Embedded Platform for Training Deep Neural Networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 37, no. 11, pp. 2360-2370, 2018.

[6] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ACM SIGARCH Computer Architecture News*, 2016, vol. 44, no. 3, pp. 27-39: IEEE Press.

[7] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News,* vol. 44, no. 3, pp. 14-26, 2016.

[8] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, 2017, pp. 541-552: IEEE.

[9] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Drisa: A dram-based reconfigurable in-situ accelerator," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 288-301: ACM.

[10] C. Eckert *et al.*, "Neural cache: bit-serial in-cache acceleration of deep neural networks," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, 2018, pp. 383-396: IEEE Press.

[11] Y. Long *et al.*, "A ferroelectric FET based power-efficient architecture for data-intensive computing," in *Proceedings of the International Conference on Computer-Aided Design*, 2018, p. 32: ACM.

[12] J. Muller, T. S. Boscke, U. Schroder, R. Hoffmann, T. Mikolajick, and L. Frey, "Nanosecond Polarization Switching and Long Retention in a Novel MFIS-FET Based on Ferroelectric HfO2," *IEEE Electron Device Letters,* vol. 33, no. 2, pp. 185-187, 2012.

[13] M. Trentzsch *et al.*, "A 28nm HKMG super low power embedded NVM technology based on ferroelectric FETs," in *2016 IEEE International Electron Devices Meeting (IEDM)*, 2016, pp. 11.5. 1-11.5. 4: IEEE.

[14] H. Mulaosmanovic *et al.*, "Novel ferroelectric FET based synapse for neuromorphic systems," in *2017 Symposium on VLSI Technology*, 2017, pp. T176-T177: IEEE.

[15] A. Aziz *et al.*, "Computing with ferroelectric FETs: Devices, models, systems, and applications," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 1289-1298: IEEE.

[16] Z. Wang, S. Khandelwal, and A. I. Khan, "Ferroelectric oscillators and their coupled networks," *IEEE Electron Device Letters,* vol. 38, no. 11, pp. 1614-1617, 2017.

[17] Z. Wang *et al.*, "Experimental Demonstration of Ferroelectric Spiking Neurons for Unsupervised Clustering," in *2018 IEEE International Electron Devices Meeting (IEDM)*, 2018, pp. 13.3. 1-13.3. 4: IEEE.

[18] X. Chen, X. Yin, M. Niemier, and X. S. Hu, "Design and optimization of FeFET-based crossbars for binary convolution neural networks," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 1205-1210: IEEE.

[19] S. F. Mueller, "Development of HfO2-Based Ferroelectric Memories for Future CMOS Technology Nodes," PhD thesis 2014.

[20] M. Lee *et al.*, "Physical thickness 1. x nm ferroelectric HfZrOx negative capacitance FETs," in *2016 IEEE International Electron Devices Meeting (IEDM)*, 2016, pp. 12.1. 1-12.1. 4: IEEE.

[21] H. Mulaosmanovic, T. Mikolajick, and S. Slesazeck, "Accumulative polarization reversal in nanoscale ferroelectric transistors," *ACS applied materials & interfaces,* vol. 10, no. 28, pp. 23997-24002, 2018.

[22] *Google TPU-v2* https://cloud.google.com/tpu/docs/system-architecture.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770-778.

[24] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861,* 2017.

[25] B. Gao *et al.*, "Ultra-low-energy three-dimensional oxide-based electronic synapses for implementation of robust high-accuracy neuromorphic computation systems," *ACS nano,* vol. 8, no. 7, pp. 6998-7004, 2014.

[26] M. Jerry *et al.*, "Ferroelectric FET analog synapse for acceleration of deep neural network training," in *2017 IEEE International Electron Devices Meeting (IEDM)*, 2017, pp. 6.2. 1-6.2. 4: IEEE.