FPSA: A Full System Stack Solution for Reconfigurable ReRAM-based NN Accelerator Architecture

Yu Ji
jiy15@mails.tsinghua.edu.cn
Tsinghua University
Beijing, China

Youyang Zhang
Tsinghua University
Beijing, China

Xinfeng Xie
University of California
Santa Barbara, USA

Shuangchen Li
University of California
Santa Barbara, USA

Peiqi Wang
Tsinghua University
Beijing, China

Xing Hu
University of California
Santa Barbara, USA

Youhui Zhang*
zyh02@tsinghua.edu.cn
Tsinghua University
Beijing, China

Yuan Xie
yuanxie@ece.ucsb.edu
University of California
Santa Barbara, USA

Abstract

Neural Network (NN) accelerators with emerging ReRAM (resistive random access memory) technologies have been investigated as one of the promising solutions to address the *memory wall* challenge, due to the unique capability of *processing-in-memory* within ReRAM-crossbar-based processing elements (PEs). However, the high efficiency and high density advantages of ReRAM have not been fully utilized due to the huge communication demands among PEs and the overhead of peripheral circuits.

In this paper, we propose a full system stack solution, composed of a reconfigurable architecture design, Field Programmable Synapse Array (FPSA) and its software system including neural synthesizer, temporal-to-spatial mapper, and placement & routing. We highly leverage the software system to make the hardware design compact and efficient. To satisfy the high-performance communication demand, we optimize it with a reconfigurable routing architecture and the placement & routing tool. To improve the computational density, we greatly simplify the PE circuit with the spiking schema and then adopt neural synthesizer to enable the high density computation-resources to support different kinds of NN operations. In addition, we provide spiking memory blocks (SMBs) and configurable logic blocks (CLBs) in hardware and leverage the temporal-to-spatial mapper to utilize them to balance the storage and computation requirements of NN.

Owing to the end-to-end software system, we can efficiently deploy existing deep neural networks to FPSA. Evaluations show that, compared to one of state-of-the-art ReRAM-based NN accelerators, PRIME, the computational density of FPSA improves by 31×; for representative NNs, its inference performance can achieve up to 1000× speedup.

CCS Concepts • **Computer systems organization → Neural networks**; **Reconfigurable computing**; **Analog computers**; *Data flow architectures.*

Keywords neural networks accelerator, ReRAM, reconfigurable architecture

ACM Reference Format:
Yu Ji, Youyang Zhang, Xinfeng Xie, Shuangchen Li, Peiqi Wang, Xing Hu, Youhui Zhang, and Yuan Xie. 2019. FPSA: A Full System Stack Solution for Reconfigurable ReRAM-based NN Accelerator Architecture. In *2019 Architectural Support for Programming Languages and Operating Systems (ASPLOS '19), April 13–17, 2019, Providence, RI, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3297858.3304048

1 Introduction

Neural Networks (NNs) have achieved state-of-the-art performance benefits in a wide range of AI applications [4, 16, 26, 40, 42, 43], motivating the intensive studies on the design of NN accelerators to execute NN applications more efficiently.

*Corresponding author

ASPLOS '19, April 13–17, 2019, Providence, RI, USA

ACM ISBN 978-1-4503-6240-5/19/04…$15.00
https://doi.org/10.1145/3297858.3304048

ReRAM-based NN accelerator designs have been investigated as promising solutions due to the unique capability of performing efficient neural computing operations within ReRAM arrays [9, 17, 39, 41], which is called *computing-in-memory* or *processing-in-memory* (PIM) architecture enabled by the analog computing capability of ReRAM [17]. Existing ReRAM-based NN accelerators [9, 39, 41] have shown a significant speedup over their digital counterparts [5, 7, 8, 13] because ReRAM can integrate computation and memory in the same physical place, which reduces the data movement between memory and computing elements. ReRAM cells provide extremely high efficiency for dot-product computation, at high area density. It takes approximately $10ps$[1] for a 100×100 crossbar [47] to complete the vector-matrix multiplication The size of an ReRAM cell is approximately $4F^2$ [12], where F is the feature size of the integrated circuit process.

Existing ReRAM-based NN accelerators usually use ReRAM-crossbar as the basic building block to calculate analog vector-matrix multiplication, and put a lot of efforts on hardware design to enable NN computation. However, existing accelerators demonstrate far less efficiency and density than ReRAM's potential. The main bottleneck is communication.

Communication Bottleneck. Without loss of generality, by the analysis (details are given in Section 3) of one of the state-of-the-art ReRAM-based NN accelerators, PRIME [9], we found that as the performance of processing elements (PEs) is increased significantly by ReRAM-crossbars, the communication between these PEs becomes a new system bottleneck. Existing studies either use a memory bus [9, 41] or Network-on-Chip (NoC) [33, 39] for communication. The shared memory bus will inevitably become a bottleneck under the huge demand for data movement between PEs. For NoC, the transmission latency is usually high and the bandwidth is still not enough for ReRAM-based PEs.

The analysis further shows that even if we solve the communication bottleneck, the overhead of peripheral circuits still makes the real performance of PE far from potential.

Peripheral Circuit Overhead. Although ReRAM provides extremely high density, its peripheral circuits, such as analog-to-digital converters (ADCs) and digital-to-analog converters (DACs), occupy the majarity of a PE's area and processing latency, which seriously offsets the efficiency and density advantages. Some recent studies [39, 41] try to reduce the overheads, but, fundamentally, the issue is not solved. In addition, ReRAM crossbar is efficient when calculating vector-matrix multiplication. To support various and quickly evolving NNs, the peripheral circuits need to be more versatile in order to process a variety of operations, which worsens the problem.

To conquer these challenges, we propose an end-to-end full stack solution, which highly leverages software to use hardware resources efficiently, rather than complicating hardware. It is composed of a novel reconfigurable architecture for ReRAM-based NN accelerator, Field Programmable Synapse Array (FPSA), and the software system including neural synthesizer, spatial-temporal mapper, and placement & routing.

For communication, we optimize the communication subsystem with a reconfigurable routing architecture, which provides massive wiring resources for extremely high bandwidth and low latency and utilize them with the placement & routing tool. Due to this optimization, we can achieve about two-orders-of-magnitude speedup in comparison of PRIME.

For peripheral circuits, we employ spiking schema to simplify the PE circuit while still maintaining the functionality of vector-matrix multiplication and Rectified Linear Unit (ReLU) activation for artificial neural network (ANN). We leverage the neural synthesizer to make the NN computation more compact and enable our high density homogeneous hardware to support different kinds of NN operations in order to fully utilize the advantage of ReRAM. The latency and area of the entire PE is reduced by 94.90% and 36.63% respectively, which provides another order-of-magnitude speedup.

Last but not least, we introduce spiking memory blocks (SMBs) and configurable logic blocks (CLBs) in hardware as on-chip buffer and programmable logic. They are utilized by the spatial-to-temporal mapper to achieve optimized resource allocation and scheduling in order to balance the storage and computation requirements of NN, especially catering to the weight sharing property of convolutional neural networks (CNNs). It can lead to super-linear performance increase with more hardware resources.

In our design, the performance is no longer bounded by the *communication bottleneck*, and the *peripheral circuit overhead* is significantly reduced. Experiments show that the performance is increased by 1000× compared to PRIME [9], which is all due to the architectural and system improvements.

ReRAM-device variation is also considered: We propose a novel weight representation method, the *add* method, to decrease device variation exposed to NN models. It can approach the full precision accuracy for large-scale NNs.

The contributions of this paper are summarized as follows.

- We propose a full stack solution for ReRAM-based NN accelerator, including a reconfigurable architecture, FPSA, and the software hierarchy. The latter fully utilizes the various kinds of programmable resources provided by the former to deploy NN efficiently. Evaluations show that our approach can outperform a state-of-the-art ReRAM-based accelerator, PRIME, by up to 1000× for NN inference.
- We have observed that communication is the bottleneck of existing ReRAM-based NN accelerator and then propose to optimize it with a reconfigurable routing architecture to break this bound.

[1] It is the resistive-capacitive delay of just the crossbar circuits

- We make the PE design much more compact and efficient by leveraging the spiking schema. The latency is decreased by 19.6× and the density is improved by 1.6×.

Finally, we believe that it is a new design philosophy for ReRAM-based NN accelerators. Inspired by the spirit of the reduced instruction set computer (RISC) architecture of the conventional computer systems, our compact hardware design enables extremely high performance and can support rich NN functionalities with the software stack.

2 Background and Related Work

2.1 ReRAM-Based NN Acceleration

Neural Network applications are both memory-intensive and compute-intensive. Thus, there are a lot of NN accelerators [3, 5, 7, 8, 13, 15, 22, 23, 30, 32, 38] based on mature digital circuits to speedup NN computations.

To further increase the performance and eliminate other problems such as *memory wall*, quite a few studies on ReRAM based NN accelerators and neuromorphic hardware [2, 9, 18, 21, 24, 33, 36, 37, 39, 41, 44] have also been proposed.

Resistive random access memory, known as ReRAM, is a type of emerging non-volatile memory, which stores the information using its resistance. Prior work [17] shows that the ReRAM-based crossbar is very efficient at computing analog vector-matrix multiplications in the locations where the matrices are stored. As shown in Figure 1, there is a ReRAM cell in each intersection of the crossbar. An input voltage vector $\{V_i\}$ is applied to the rows and is multiplied by the conductance matrix of ReRAM cells $\{G_{ji}\}$. The resulting currents $\{I_j\}$ are summed across each column. The output current vector can be calculated by $I = GV$.

Existing studies on ReRAM-based NN accelerators [9, 39, 41] treat the ReRAM-crossbar as a very low-precision vector-matrix multiplication engine, and use it as the building block, combined with peripheral circuits, to construct NN accelerators. To support higher precision, these studies usually use the *splicing* method, which employs multiple cells for different bits of the high precision number and shift-add the partial sum of different bits to get the final result. For example, ISAAC conservatively uses 8 cells to represent one 16-bit cells; each cell represents 2 bits. PRIME [9] and PipeLayer [41] are modified from the ReRAM-based memory chip. Thus, their PEs are connected through the internal hierarchical memory bus. ISAAC [39] is a dedicated accelerator, which employs NoC.

2.2 Reconfigurable Architecture

Reconfigurable architecture provides much higher efficiency than general-purpose processors while providing more flexibility than Application Specific Integrated Circuits (ASICs). There are also some reconfigurable routing architectures designed for NN accelerators such as MAERI [27], but they target to the accelerators based on digital circuits. The capability is still far from the demands for ReRAM-based PEs.

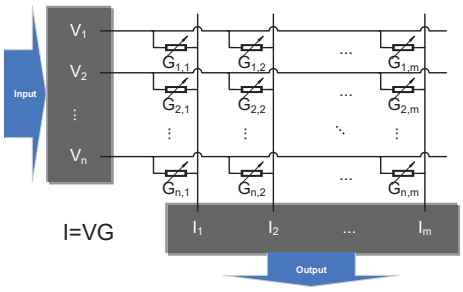


Figure 1. Vector-matrix multiplication with ReRAM crossbar

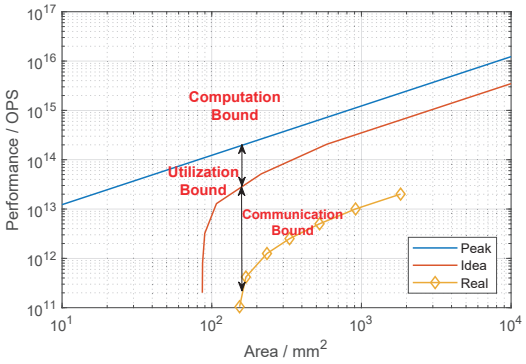


Figure 2. Performance vs. Area for the peak performance, the ideal case (with infinite bandwidth), and the real case for running VGG16 [40] on PRIME [9] (45*nm* process). The performance of the real case is bounded by communication.

FPGA is one of the most widely-used reconfigurable architectures, composed of many Configurable Logic Blocks (CLBs). The main function modules of CLB are Look-Up Tables (LUTs) that can be configured to achieve any arbitrary logic function. The routing architecture of an FPGA chip occupies up to 90% of the total area [14], and provides most of the reconfigurability. It consists of wires and programmable switches. The programmable switches use Connection Boxes (CBs) to configure the connection from CLBs to the routing network, and use Switch Boxes (SBs) to configure the connections from different wire segments. There have been many studies [10, 34, 45, 46, 48, 50] on using ReRAM to augment existing reconfigurable architectures. For example, ReRAM cells are used to replace SBs and CBs in FPGA [10] and to implement arbitrary logic function [50].

3 Motivation

We analyze the scalability and performance of PRIME [9][2], which uses memory bus as the communication subsystem;

[2]Thanks to the authors of PRIME. We got all of its implementation code.

we assume that its structure can scale-out linearly under 45nm process. A large scale CNN, VGG16 [40] for ImageNet [11], is employed as the NN application.

Based on the hardware configurations and NN requirements, we can get three performance bounds (Figure 2) as follows.

Computation Bound. It is the theoretical upper bound (which is defined as *peak performance* in this paper), the product of the PE number and the performance of one PE, as the total computation capability provided.

Utilization Bound. Usually, computation and communication capabilities are two important factors restricting performance improvement. But, even if the communication is ideal, the performance (called *ideal performance*) still cannot reach the peak value, caused by the following two utilization issues:

• *Temporal Utilization (Load Balance).* The first is the imbalance between storage and computation requirements of NN, especially for convolutional neural networks (CNNs). For example, the first two convolutional layers of VGG16 only occupy 0.028% of weight storage but consume 12.5% of computation because the weights are reused by 224×224 different regions of the input feature map, while the fully connected layers take 89.3% of storage but only consume 0.8% of computation.

In contrast, ReRAM-crossbars integrate computation and storage in the same physical place; thus a PE can only provide computing power commensurate with its storage capacity.

To map a neural network onto the ReRAM-based NN accelerator, the prerequisite is that there should be enough PEs for all the weight parameters. This mapping is quite unbalanced: about 0.028% of PEs should process 12.5% of computation and become the bottleneck, while the utilization of other PEs is low. This issue can be solved when more PEs are available: We can duplicate these layers' weights onto more PEs to speedup them significantly. For example, adding extra 0.028% of PEs for the first two layers can double the performance. That is why the first half of the *ideal performance* curve shows a super-linear increase. The curve will converge to linear scalability and approach the *peak performance* when different layers are balanced.

•*Spatial Utilization (Crossbar Mapping).* The fixed size of crossbars cannot match weight matrices of different scales perfectly, which also affects the PE utilization.

Between the two, the first is the main issue.

Communication Bound. In real cases with limited bandwidth, the utilization cannot be improved efficiently when more PEs are provided because the communication subsystem cannot fetch enough data in time for the PEs. This leads to a large gap with the ideal case.

Currently, PRIME has tried to balance the computation and communication requirements. However, due to its limited bus bandwidth, its real performance is far below the ideal value (two orders of magnitude lower than the latter).

Based on these observations, it is reasonable to improve the performance of ReRAM-based accelerators with the following methods in order.

1. **Improving Communication.** We should improve the communication subsystem to break the *communication bound*.
2. **Reducing Area.** We should reduce the area of a single PE to push the performance to the high-utilization region of the *utilization bound* for a given chip area.
3. **Reducing Latency.** We should reduce the latency of PEs to increase the peak performance (the *upper bound)* further.

Accordingly, we adopt the reconfigurable routing architecture first and then design simplified PE circuits to reduce area and latency, which are given in Section 4; the whole system software stack is proposed in Section 5.

4 Architecture Design

Figure 3 shows the overview of FPSA architecture. It contains three kinds of function blocks: ReRAM-based processing elements (PE) for computation, spiking memory block (SMB) for buffering, and configurable logic block (CLB) for controlling. These blocks are connected through a reconfigurable routing architecture. Functional blocks and the routing architecture are all programmable, which provide massive computation, buffering, controlling, and wiring resources for software to utilize.

To reduce the peripheral circuit overhead, we employ spiking schema to perform the vector-matrix multiplication. It uses the spike count to represent a high-precision number rather than the amplitude of an analog signal. The area and latency can be significantly reduced with this schema. In addition, the spiking memory block is customized to buffer spiking signals.

4.1 Routing Architecture

PEs and other function blocks are connected by the routing architecture and working in parallel in a pipelined manner. The pipeline clock cycle is bounded by the maximum latency of all pipeline stages, including the computation and communication latency. As mentioned before, the computation time has been significantly reduced by ReRAM-crossbar, which makes the communication a system bottleneck.

Therefore, we adopt the reconfigurable routing architecture widely used in FPGA chips, instead of the memory bus or NoC in existing NN accelerators. Compared to the memory bus and NoC that reuse physical channels for different traffic and provide flexible runtime data-path, the reconfigurable routing architecture assigns individual channels for each signal in the configuration phase and has a fixed runtime data-path (since the NN topology is fixed, the runtime flexibility is unnecessary). Furthermore, compared to the bus and NoC where the worst communication latency is not

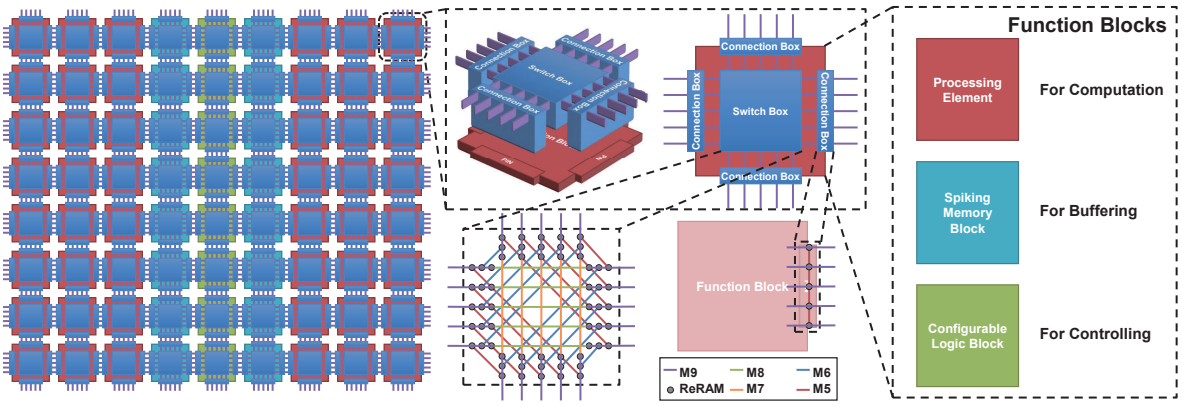


Figure 3. FPSA Architecture Overview. The function blocks are connected through the reconfigurable wiring network.

guaranteed, the maximum latency of critical path can be evaluated in advance.

One of the most widely used FPGA routing architectures is the island-style architecture: configurable logic blocks (CLBs) are connected to the wiring network through connection boxes (CBs) and different wiring segments are connected through switch boxes (SBs). Normally, the routing architecture consumes most of the FPGA chip area [14]. In our design, the area consumption would be greater because of more fan-in/outs in the ReRAM-based PEs than those of CLBs in normal FPGA.

To reduce this overhead, we adopt the previous work, mrFPGA [10], that employs ReRAM cells to construct CBs and SBs to reduce the area consumption. Figure 3 provides a detailed view of the routing architecture, in which SBs and CBs are placed over the function blocks. Specifically, the connections in SBs and CBs are decided by the resistance of the ReRAM cells. For example, an ReRAM cell with high resistance means that there is no connection between the two corresponding segments while low resistance is a pass. Figure 3 also provides the detailed wiring and layout inside CBs and SBs, which only use five metal layers from M5 to M9 without resource conflict. Functional blocks are connected to the wiring network through the CBs at four sides.

4.2 Processing Elements

We use spiking schema to simplify the peripheral circuits of PE. The inputs of the PE are digital spike trains that use the spike count to represent a number between 0 and 1. Although it requires 2^n spikes to represent a number of n bits, processing spikes is much more efficient than processing high-precision analog signals comprehensively.

The essential of the PE is an ReRAM crossbar followed by spiking neuron circuits. The input signal will be converted into a charging voltage and applied to each row of the crossbar. Then the resulting current of each column will be injected into the corresponding neuron circuit, which accumulates the current and issues a spike when the threshold voltage is reached.

In order to handle negative weights with the positive conductance values, we use two physical adjacent columns to represent one logic column of the weight matrix, one for the positive part and one for negative. The output spike train of the negative column will be subtracted from the positive one to get the final output.

Accordingly, the main components of a PE are charging units (one for each row), ReRAM-crossbar, neuron units (one for each column), and spike subtracters (one for every two columns). The overview of a PE is shown in Figure 4 Ⓐ.

Charging Unit. As shown in Figure 4 Ⓑ, since the input spike is a 1-bit signal, the DAC can be simplified to a transistor. When a spike signal arrives, the transistor will open and the charging voltage will be applied to this row.

ReRAM Crossbar. Figure 4 Ⓒ is the ReRAM crossbar. Each row connects to an input charging unit and each column connects to an output neuron unit. ReRAM cells are in the intersections of the crossbar.

Neuron Unit. It is an analog implementation of one widely used spiking neuron model, integrate-and-fire (IF) model. As shown in Figure 4 Ⓓ, it has a capacitor to integrate the current from the corresponding column. When its internal voltage reaches the threshold voltage, a spike signal will be stored in the S-R latch; the discharging unit will be turned on to discharge the capacitor until the voltage reaches the reset value. The discharging unit can also be triggered by a reset signal because we use the spike count in a sampling window to represent a number. Thus, a reset signal will be sent to clear internal states before a new sampling window begins.

Spike Subtracter. Figure 4 Ⓑ shows the circuit of the spike subtracter. It has two input spike trains from the corresponding two neuron units. The output is also a spike train, whose spike count is the different of the two inputs. The

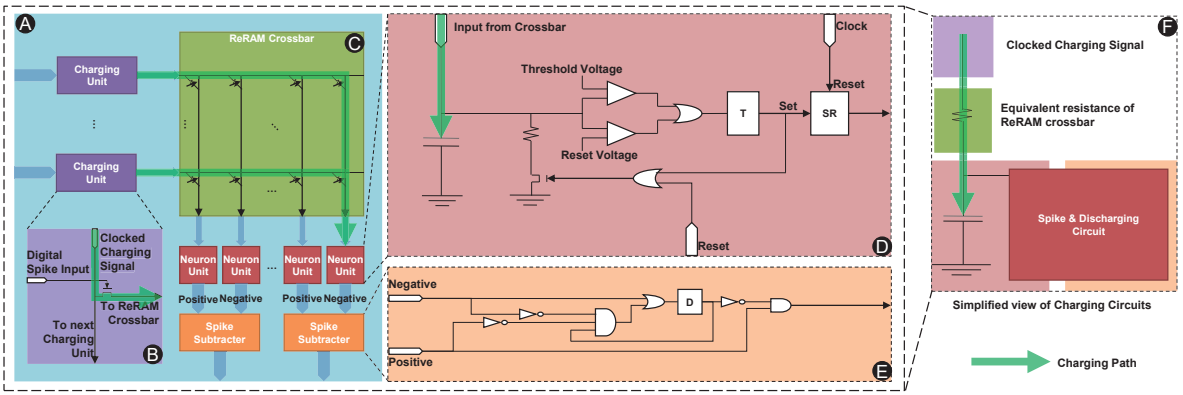


Figure 4. Overview of the Processing Element. The input is digital spike signals from routing architecture. The crossbar uses two columns for one output: one for the positive part and one for the negative. Neuron units integrate the output current from the corresponding crossbar-column and generate digital spikes. The spike subtracter computes the difference of the adjacent positive and negative columns. The green line represents the charging path of the capacitor of neuron unit. The simplified view of the charging circuits is on the right side of the figure.

working mechanism is that the spikes from the negative neuron unit will block the next spike coming from the positive neuron.

Although we use spiking schema in our circuit design, the computation achieved by the circuit is just a vector-matrix multiplication followed by the ReLU activation function; the precision depends on the size of the sampling window. The proof is as follows. The equivalent charging circuit is shown in Figure 4 Ⓕ . We denote the charging voltage from the voltage source as V_{dd}, the capacitance of neuron unit as C, and the charging time of each clock cycle as τ. For the j-th output neuron unit, the equivalent resistance of the ReRAM-crossbar is denoted as $R_j(t)$ at time t. We suppose that from the reset voltage V_{re}, the neuron unit's capacitor reaches the threshold V_{th} in the T-th cycle. In accordance with the model of charging a capacitor in an RC circuit, Equation 1 gives the capacitor's voltage U_T at the cycle T.

$$V_{dd} - U_T = (V_{dd} - U_{T-1})e^{-\frac{\tau}{R_j(T)C}} = (V_{dd} - V_{re})e^{-\frac{\tau}{C}\sum_{t=1}^{T}\frac{1}{R_j(t)}} \tag{1}$$

When U_T reaches the threshold V_{th} at the T-th cycle, we can derive Equation 2.

$$\sum_{t=1}^{T}\frac{1}{R_j(t)} = \frac{C}{\tau}\ln\frac{V_{dd} - V_{re}}{V_{dd} - V_{th}} \tag{2}$$

For convenience, we denote the right-hand side of Equation 2 as η because it is a constant. On the left-hand side, the equivalent resistance only counts the rows with spike inputs. Therefore, we can derive Equation 3 as follows, where $s_i(t)$ is the spike signal for the i-th row at time t and g_{ji} is the conductance of the cell at the intersection of the i-th row and the j-th column.

$$\sum_{t=1}^{T}\frac{1}{R_j(t)} = \sum_{t=1}^{T}\sum_{i}s_i(t)g_{ji} = \sum_{i}g_{ji}\sum_{t=1}^{T}s_i(t) = \eta \tag{3}$$

Suppose the size of the sampling window is Γ cycles. During this period, the spike counts of the i-th input row and the j-th output column are X_i and Y_j respectively. Thus, the voltage of the capacitor has reached the threshold for Y_j times and then we have Equation 4.

$$\sum_{i}g_{ji}\sum_{t=1}^{\Gamma}s_i(t) = Y_j\eta \tag{4}$$

By definition, X_i is the sum of $s_i(t)$ of the sampling window Γ. Thus, the relationship between the input and output spike count is shown in Equation 5.

$$Y_j = \sum_{i}\frac{g_{ji}}{\eta}X_i \tag{5}$$

Further, we connect two columns to one spike subtracter to support negative weight values. Suppose the corresponding spike counts and conductance values for positive and negative columns are Y_j^+, Y_j^- and g_{ji}^+, g_{ji}^-, respectively. The subtracter blocks Y_j^- spikes from the Y_j^+ if $Y_j^+ > Y_j^-$, or the output spike count is 0. Thus the final spike count from the j-th output port is shown in Equation 6.

$$Y_j = \max(Y_j^+ - Y_j^-, 0) = \mathrm{ReLU}(\sum_{i}\frac{g_{ji}^+ - g_{ji}^-}{\eta}X_i) \tag{6}$$

In conclusion, the difference from existing ReRAM-based accelerators that employ spiking schema (e.g. PipeLayer [41]) is that we directly charge the capacitor and transit spike trains between PEs. Thus, the overhead of current mirrors

and encoder/decoder for spike trains can be removed. Equation 6 shows that with this simplification we can still complete the vector-matrix multiplication followed by ReLU. In addition, owing to the area reduction, we do not need to reuse peripheral circuits for different rows and columns. They can process input and output of an ReRAM-crossbar in parallel. In contrast, existing ReRAM-based accelerators usually share ADCs and/or DACs to reduce the area overhead, which also leads to a corresponding increase in processing delay. (e.g. in ISAAC [39], 128 crossbar-columns share one ADC). Our approach achieves a good balance in terms of function, area cost and time delay. Quantitative evaluation will be given in Section 6.

4.3 Spiking Memory Block

As shown in Figure 3, in addition to the computation resources provided by PEs, we also have spiking memory blocks (SMBs) to provide on-chip buffer for the intermediate data.

Since the size of on-chip buffers has a significant impact on chip area, we only store the spike counts instead of the spike trains to fully use the buffers. The counters and spike generators are embedded inside the SMB to do the encoding and decoding between spike counts and spike trains; thus SMB can directly send and receive spike trains but only store the spike counts. The internal memory is indexed by bits so that it can fit any sampling window size (e.g., when the sampling window is 2^n, it can store the spike counts in the manner of n-bit by n-bit.

Although we heavily adopt ReRAM in our PE design and routing architecture, we still use SRAM for the SMB. ReRAMs are not suitable for buffers because they have low endurance (they can support about 10^{12} writes).

4.4 Configurable Logic Block

Further, we provide configurable logic blocks (CLBs) to provide logic resources for controlling as shown in Figure 3. The control signals for PEs and SMBs are generated by the CLBs.

We also use SRAMs to implement the LUTs in CLBs. Although ReRAM provides higher density than SRAM, it requires current sense amplifiers to read data, which consume a lot of area. Thus, its area efficiency is very poor when the capacity is small: A conventional 6-input LUT can be implemented with a 64-bit memory. According to NVSim [12], the area of a 64-bit SRAM cell is $35.129\mu m^2$ under $45nm$ process while the area of an ReRAM cell is $172.229\mu m^2$. Thus, CLBs contain multiple SRAM-based LUTs, flip-flops, and multiplexers to perform any logic function.

5 System Design

We highly leverage the software system to enable flexible functionality and high efficiency of FPSA architecture. Utill now, the hardware has provided massive computation, buffering, and controlling resources in the form of the three kinds of function blocks, as well as the massive wiring resources and configurable connections through the routing architecture. How to make full use of these hardware resources to fit the diversity of NN requirements is a complex problem, especially as we try to maintain the advantages of ReRAM (i.e. the high computational density of vector-matrix multiplication).

From a formal perspective, most deep learning frameworks [1, 6, 35] use computational graph (CG) as the programming model to represent NNs. Thus, the problem is how to efficiently map the software-level CG to the above reconfigurable resource pool.

We divide the problem into three independent sub-problems and design the software stack to solve them respectively, as shown in Figure 5. First, the neural synthesizer transforms the NN CG to make up the gap between the NN requirement and hardware functionality. Second, the spatial-to-temporal mapper gives the optimized allocation of PE-resources and the scheduling strategy for the above-mentioned output CG, including the corresponding control logic; all of them are collectively referred to as the function-block netlist. Finally, we place the netlist onto the FPSA chip and generate the routing.

5.1 Neural Synthesizer

Here the essential is to maintain the user-friendly programming interface and synthesize NN model into a hardware-friendly, compact representation for efficient execution.

Flexible NN Programming. Computational graph (CG) is widely used programming model in most deep learning frameworks. It is a graph that consists of many tensor operations and describes the data dependencies of the operations. There are hundreds of flexible and complex operations in most deep learning frameworks.

Efficient ReRAM Execution. The support of hundreds of operations in hardware is impractical. On the other side, our ReRAM-based PE can complete vector-matrix multiplication with ReLU function very efficiently (in Section 4.2). Therefore, the neural synthesizer is expected to synthesize the software CG into an equivalent CG only including operations that the hardware can support efficiently.

We adopt the existing NN compiler framework from Y. Ji et al [19, 20] to do the synthesis. They propose to transform a trained, software NN into an equivalent network that meets hardware constraints; one case study is to transform such a CG into a core-op graph (core-op is defined as an operation composed of a low-precision vector-matrix multiplication and ReLU). Namely, it can implement different kinds of operations with the core-op, and then fine-tune the model to retain the accuracy. The basic idea is to construct dedicated structures with core-ops to implement other operations or

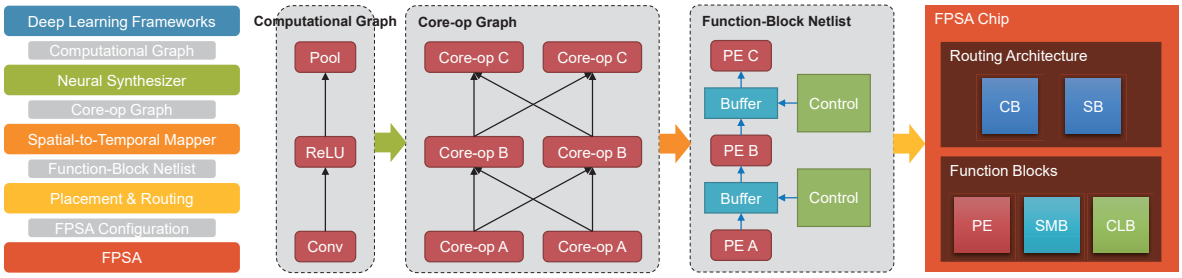


Figure 5. System stack of FPSA.

approximate them with multilayer perceptrons (MLPs). Further, large fully-connected layers or convolutional layers will be split into multiple small core-ops.

5.2 Spatial-to-Temporal Mapper

The output core-op graph only contains purely computational tasks. If we map CG nodes onto PEs directly, it will require extremely huge amount of PEs, which is impractical. For example, although a convolutional layer reuses its kernel weights for different regions of input feature map, its core-op graph contains individual core-ops for each region. Thus, we have to temporally map the core-op graph onto hardware with the on-chip buffering and controlling resources. Still taking the convolutional layer as an example, we can map all core-ops with shared weights onto one or more PEs and reuse the weights in a time-division-multiplexing manner. Accordingly, the mapper will generate an optimized netlist of function blocks for the core-op graph: PEs complete all the computation tasks, buffers hold the intermediate data, and control logic will be generated to schedule the execution. Further, the buffers separate the entire circuits into multiple pipeline stages and different pipeline stages process different samples in parallel. The mapping involves the following two sub-steps.

Resource Allocation. As discussed in Section 3, different layers reuse the weights for different times. We should assign more PEs to those layers that reuse weights more times. To do so, we have all the core-ops with the same weights into one group. The number of core-ops in one group is denoted as *reuse degree*. The iterations required to complete the computation of a group depends on the number of PEs assigned to that group. We first allocate one PE for each group to satisfy the minimum storage requirement. To balance the pipeline stages, we will assign extra PEs to those groups that require more iterations to complete if more PEs are available. The number of duplications (PEs) assigned to one group is referred as *duplication degree* of that group. We use the *duplication degree* of the group with the maximum *reuse degree* as the *duplication degree* of the entire model. With $n\times$ duplication degree, the temporal utilization bound is usually increased by $n\times$.

Scheduling. After the core-ops are assigned to PEs, we also need to schedule the execution order, insert buffers between PEs, and generate the control signal to get the netlist. We denote the core-op graph as $G = (V, E)$ where V is the node set and E is the edge set. A_v denotes the PE assigned to the core-op $v \in V$. s_v and e_v represent the start cycle and end cycle for executing the core-op v respectively. The following contraints should be satisfied.

• **Resource Conflict (RC).** Two core-ops cannot be executed synchronously if they are assigned to the same PE, which is shown in Formula 7.

$$e_v < s_u \text{ or } e_u < s_v \quad \text{if } A_v = A_u \tag{7}$$

• **No-Buffer Dependency (NBD).** If there is data dependency between node u and v, and if these two nodes are placed into directly connected PEs without buffers, the execution time of v needs to cover the one of u to receive the spike train generated by u, as shown in Formula 8.

$$s_v \le s_u + 1 \text{ and } e_v \ge e_u + 1 \quad \text{if } (u, v) \in E \tag{8}$$

• **Buffered Dependency (BD).** *Resource conflict* and *no-buffer dependency* may conflict; thus we add buffers between the two PEs to solve conflict. The buffers will store the firing rate of u and generate spikes for v when A_v is ready. This constraints is given by Formula 9.

$$s_v > e_u \quad \text{if } (u, v) \in E \tag{9}$$

• **Buffer Conflict (BC).** If two nodes u and v receive spike trains from the same port of one buffer, the buffer should provide spike train of sampling window Γ one-by-one. The timing should satisfy Formula 10.

$$e_v > e_u + \Gamma \text{ or } e_u > e_v + \Gamma \tag{10}$$

• **Sampling Window (SW).** Finally, the execution time of each core-op cannot be less than Γ as Formula 11.

$$s_v + \Gamma \le e_v \tag{11}$$

We can optimize all the s_v and e_v for a certain objective under these constraints. Here, we show a greedy algorithm in Algorithm 1 to minimize the buffer used and the latency.

The basic idea is to traverse the graph in topological ordering and try to connect PEs without buffer. If there is any conflict, a buffer from SMB should be inserted to separate

Algorithm 1 Scheduling algorithm

Require: $G = (V, E)$, A_v
  s_v, e_v is the start/end time of $v \in V$
  for $v \in V$ in topological ordering **do**
    Let v satisfy *NBD* and *SW*
    Increase s_v, e_v to satisfy *RC*
    if v does not satisfy *NBD* with u **then**
      Mark (u, v) requires buffer
      Increase s_v, e_v to satisfy *RC* and *BD*
    for u where $(u, v) \in E$ **do**
      if any (u, p) requires buffer **then**
        Insert buffer after u
        if the buffer requires extending fan-in/out **then**
          for w where $(u, w) \in E$ requires buffer **do**
            Increase s_w, e_w to satisfy *BC*
            for $q \in V$ between w and v **do**
              Increase s_q,$e_q$to satisfy all
        Increase s_u, e_u to satisfy all
      for $p \in V$ before u in reverse ordering **do**
        Increase s_p, e_p to satisfy all

them into different pipeline stages. Then we will check all the previous nodes and adjust them to ensure all constraints are satisfied.

When all s_v and e_v have been determined, the controlling signals can be generated accordingly with the CLBs.

5.3 Placement & Routing

The last step is to place all function blocks of the netlist onto physical units. Then the CBs and SBs in the routing architecture can be configured to connect the function blocks according to the topology of the netlist. The placement & routing problem is the same as the one for FPGA. We adopt the mature solution used in FPGA development tool-chain, which usually uses simulated annealing (SA) algorithm for the placement, and uses dijkstra's shortest path algorithm for the routing to minimize the latency of critical path.

6 Evaluation

We evaluate the FPSA architecture and its system stack with a set of typical NN applications. Specifically, we evaluate the contributions of the routing architecture and simplified PEs to the whole system improvement separately. Further, the scalability is evaluated when more resources are provided.

6.1 Experiment Configurations and Methodology

Benchmark. We evaluate our proposal on NN models of different scales, including MLP-500-100 for MNIST dataset [28] (an MLP with two hidden layers composed of 500 and 100 neurons), LeNet [29] for MNIST dataset, VGG17 for CIFAR-10 dataset [25], AlexNet [26], GoogLeNet [43], VGG16 [40],

Table 1. Parameters of function blocks under 45*nm* process

	Energy *pJ*	Area μm^2	Latency *ns*
PE (256 × 256)	29.094	22051.414	2.443
Charging Unit	0.001	2.246	0.070
×256	0.229	600.704	
ReRAM (256 × 512)	0.131	1061.683	0.000
×8	1.049	8493.466	
Neuron Unit	0.039	19.247	1.463
×512	19.861	9854.342	
Subtractor	0.031	12.121	0.910
×256	8.945	3102.902	
CLB (128× LUT)	3.106	5998.272	0.229
SMB (16Kb)	1.150	5421.900	0.578

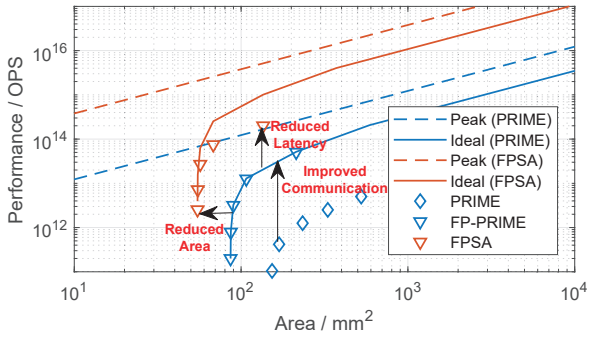


Figure 6. Comparison between PRIME, FP-PRIME (FPSA with PRIME's PE), and FPSA for VGG16.

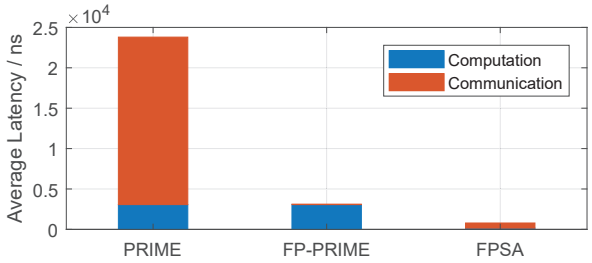
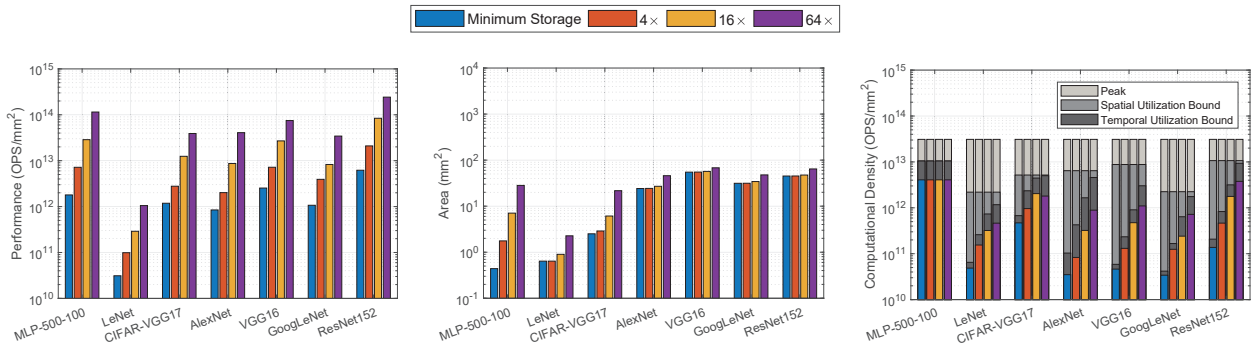


Figure 7. The breakdown of processing latency of one PE of PRIME, FP-PRIME, and FPSA (for VGG16).

Table 2. The comparison to PRIME for performing a vector-matrix multiplication of 8bit-weight, 6bit-I/O, and 256 × 256-scale.

	Area (μm^2)	Latency (ns)	Computational Density (OPS/mm^2)
PRIME	34802.204	3064.7	1.229T
FPSA	22051.414	156.4	38.004T
Improvement	-36.63%	-94.90%	30.92×



(a) Performance **(b)** Area **(c)** Computational Density

Figure 8. Scalabilty of FPSA. We show the area, computational density, and performance for all the benchmark models under different *duplication degrees*. (a) The performance increases significantly with the increase of *duplication degree*. (b) The area consumption does not increase much as performance. (c) The rest performance-increase comes from the better utilization since the *Temporal Utilization Bound* increases as more resources are available.

Model	MLP-500-100	LeNet	VGG17	AlexNet	VGG16	GoogleNet	ResNet152
Dataset	MNIST	MNIST	CIFAR-10	ImageNet	ImageNet	ImageNet	ImageNet
# of weights	443.0K	430.5K	1.1M	60.6M	138.3M	7.0M	57.7M
# of ops	886.0K	4.6M	333.4M	1.4G	30.9G	3.2G	22.6G
Throughput ($sample/s$)	129.7M	229.4K	117.4K	28.2K	2.4K	10.9K	10.8K
Latency (μs)	0.51	0.97	46.3	100.49	671.8	514.18	1106.4
Area (mm^2, $45nm$ **process**)	28.23	2.27	21.68	45.89	68.09	47.74	64.32

Table 3. The overall performance of FPSA for different NN models

and ResNet152 [16]. The last four are for the ImageNet dataset [11].

Baseline. We compare FPSA to state-of-the-art ReRAM-based accelerators, PRIME [9], ISAAC [39], and PipeLayer [41], especially PRIME (as detailed information is available). Previous studies already show great speedup over conventional digital circuits. For example, Eyeriss [7] achieves 35 frame/s throughput and 115.4ms latency for AlexNet on a chip of 12.25mm^2 under 65nm process *with* off-chip memory, while we achieve 28.2K frame/s and 100.49μs on 51.86mm^2 under 45nm process *without* off-chip memory. Most of the improvements come from device benefit. Thus, we only compare with ReRAM-based accelerators to show the improvements from the innovation at the architecture and system levels.

FPSA Configuration. The crossbar size is set to 256×512; the positive and negative values of each logic column is represented with two adjacent crossbar-columns respectively. Logically, the crossbar size is 256×256. At each intersection, we put 8 cells connected in parallel. Each cell can be set to 16 levels (4-bit), and we add up the values of 8 cells to represent an 8-bit weight. This is done for reliability reasons, which will be discussed in Section 7.2. We integrate 128 LUTs in one CLB to make the area and number of pins of one CLB similar to one PE. For SMBs, we choose SRAM with 16Kb capacity.

Simulation Setup. We use mrVPR tool for mrFPGA [10] as the placement & routing tool to evaluate the area consumption and critical path for communication. The mrVPR has two inputs: one is an architecture description file that contains the parameters of all the function blocks, and the other is a netlist composed of these blocks. We implement the neural synthesizer to generate the core-op graph and the spatial-to-temporal mapper to generate the function-block netlist for mrVPR. The parameters of function blocks are listed in Table 1. We use NVSim [12] to evaluate ReRAM-crossbar, sense amplifier, SMB and CLB, and use Synopsys Design Compiler for other peripheral circuits; all are under the 45nm process. The routing architecture is stacked over function blocks. According to the report from mrVPR, the area of the former is less. We build a simulator to evaluate the performance based on the reported routing result from mrVPR.

Methodology. To show the effects of the new routing architecture and simplified PEs, we first compare PRIME with FP-PRIME (FPSA's routing architecture with PRIME's PE) to show that the *communication bound* of PRIME can be broken. Then, FP-PRIME is compared with FPSA to show the further improvement from the new PE circuits. In addition, we evaluate FPSA with different models to give the overall performance.

6.2 Performance Improvement

Overall Comparison. In Figure 6, we compare PRIME, FP-PRIME, and FPSA for VGG16. FP-PRIME is composed of the FPSA routing architecture and PRIME's PEs, whose *peak performance* and *ideal performance* are the same as PRIME's. The performance improvements comes from the three aspects listed in Section 3: *Improving Communication*, *Reducing Area*, *Reducing Latency*.

• *Improved Communication.* Comparing PRIME and FP-PRIME in Figure 6, we can see that by introducing the reconfigurable routing architecture, FP-PRIME can break the *communication bound*. Its performance is very close to the ideal case (the gap looks negilible in the logarithmic axes).

• *Reduced Area & Latency.* Comparing FP-PRIME and FPSA, we can further increase the performance due to the area & latency reduction of our PE design.

Combining these together, we can achieve up to 1000× speedup with the same area consumption.

Communication Improvement. In Figure 7, we show the average latency of computation and communication of one PE for VGG16. The communication takes most of latency of PRIME. By introducing the reconfigurable routing, the communication latency is reduced to 59.4ns, which is negilible compared to the computation time, 3064.7ns. By further simplifying the peripheral circuits of PE, the computation time is reduced to 156.4ns, while the communication time increases to 633.9ns because we transmit the spike trains directly instead of spike counts. The communication overhead is simply the reason for the gap between the ideal case and the real case for FPSA in Figure 6. It can be improved by adding buffers: Currently, the input spike signal of the charging unit is hold by its source PE. If we add more buffers between the source and target PEs, the latency could be reduced, but it will also decrease the density advantage of current FPSA design. We will discuss more about the effect of transmitting spike trains in Section 7.1.

Area & Latency Reduction. In Table 2, we compare the area and latency of one PE in PRIME and those in FPSA. The area is reduced by 36.63% and the latency is reduced by 94.90%, which leads to the overall improvement on computational density by 31×. The major improvements are from latency reduction because we do not need to share simplified peripheral circuits among different rows and columns. The computational density is 38.004$TOPS/mm^2$, which is higher than PRIME [9] (1.229$TOPS/mm^2$), PipeLayer [41] (1.485$TOPS/mm^2$), and ISAAC [39] (0.479$TOPS/mm^2$).

6.3 Scalability & Utilization

We test the performance of FPSA under 1×, 4×, 16×, and 64× *duplication degrees* (defined in Section 5.2) for all the benchmark models, results in Figure 8. The detailed performance for the 64× case is listed in Table 3.

In Figure 8a, with 4×, 16×, and 64× *duplication degree*, the geometric mean of the performance improvement is 3.06×, 10.88×, and 38.65×, respectively. In contrast, the increase of the geometric mean of area consumption is only 1.25×, 1.85×, and 3.73×, respectively. Especially, for the last four ImageNet models, the area consumption is only increased by 1.003×, 1.074×, and 1.504× on average.

The reason for the super-linear scalability is the increased utilization when more resources are available. In Figure 8c, we show the peak computational density, the spatial utilization bound (due to the imperfect crossbar mapping), the temporal utilization bound (due to the unbalanced workload), and the real computational density. The two bounds depend on the property of the models: There is no weight sharing in the MLP model, so its workload is balanced and the two bounds coincide with each other. For CNN models, when more resources are available, the spatial utilization bounds do not change (we will discuss how to improve this bound in Section 7.3). But the temporal utilization bound will increase significantly, which provides the super-linear scalability (as long as the communication bound is not hit)

7 Discussion

Despite overall improvements, there are also some other considerations that affect our design details.

7.1 Spiking Schema

Spiking schema has been used in existing design, e.g. PipeLayer [41], to reduce the overhead of ADC and DACs, but there is a significant different between our work and theirs. We transmit spike trains directly through the routing architecture while they transmit the spike counts. Despite the saved overhead of encoder/decoder circuits, it can also reduce end-to-end latency and on-chip buffers.

As discussed in Section 5.2, when two PEs are connected directly without buffers, the post-PE can start computation only 1 cycle after the pre-PE starts (the No-Buffer Dependency (NBD)), and we only need 1-bit buffer to store current spike. If we transmit the spike counts, the post-PE should wait for at least 2^n cycles (the sampling window for n-bit number) until the pre-PE finish all its computation, and then start its computation. In addition, it needs n-bit buffer to store the spike count. Thus, by transmitting the spike trains directly, we can gain up 2^n× end-to-end latency reduction for NBD and n× buffer consumption saving. The drawback is that we will generate 2^n-bit traffic for an n-bit number, which is the reason for the increased communication latency from FP-PRIME to FPSA in Figure 7. But compared to the original latency of PRIME, it is negligible. We list them in Table 3: the latency for VGG16 is only 671.8μs while PRIME's is 102.0ms.

7.2 Device Variation and NN accuracy

ReRAM devices are not ideal. Due to the programming overhead and the intrinsic working mechanism of ReRAM cells, its conductance value cannot be programmed to the exact value as expected; the conductance value also has cycle-to-cycle variation [49]. The device variation will inevitably lead to inaccurate results even if we set a tight margin between levels. The reason is that, in the ReRAM-crossbar based computing, there is no explicit read to quantize the obtained conductance, and all currents (with errors) from cell along the same column will accumulate. Some software approaches, e.g. Vortex [31], have been proposed to make NN models more robust to variation. We have adopted these methods in our neural synthesizer, but as the inherent fault tolerance of NN is limited, for relative large variation, the effect is limited. Thus, from the architecture perspective, we should also leverage more cells for one weight value to reduce the variation exposed to software level. Without loss of generality, suppose the conductance of an ReRAM cell is a random variable obeys a normal distribution $N(\mu, \sigma^2)$ rather than a number. We use *normalized deviation*, which is the ratio between the standard deviation and the value range, to measure the variation exposed to software.

The existing splicing method. Most existing architecture studies [9, 39] employ the *splicing method*, which uses multiple cells for different bits of a number, to increase the representation precision of ReRAM. Suppose we use two n-bit cells to form a number of $2n$-bit cell, one for the high n bits and one for the low n bits. Their conductance values are H and L, respectively: $H \sim N(h, \sigma^2)$ and $L \sim N(l, \sigma^2)$ where h and l are the expected values of the high n bits and low n bits, respectively. The number should be expressed as $2^n H + L \sim N(2^n h + l, (2^n\sigma)^2 + \sigma^2)$. Its *normalized deviation* is $\sqrt{2^{2n}+1}\sigma/(2^{2n}-1)$, which is almost equal to the ratio of one-cell case, $\sigma/(2^n - 1)$. Namely, it has little improvement on accuracy.

The new add method. We propose the *add method* that will add the conductance values evenly to increase precision and reduce variation. Considering the general case that n cells ($X_1, \ldots, X_n$ and $X_i \sim N(x_i, \sigma^2)$) are joined together by coefficient $a_1, \ldots, a_n$. Then the number is expressed as $\sum_i a_i X_i \sim N(\sum_i a_i x_i, \sum_i (a_i\sigma)^2)$. The *normalized deviation* is decreased by $\sum_i |a_i| / \sqrt{\sum_i a_i^2}$. According to Cauchy inequality, the deviation decrease would reach its maximum value $\sqrt{n}$ when $|a_1| = \ldots = |a_n|$.

Figure 9 shows the effect of the two methods on the accuracy of VGG16. The variation data is derived from real fabricated ReRAM cells [49]. PRIME use two 4-bit cells to form an 8-bit weight value with *splicing*. The accuracy drops to 70% of the full precision accuracy. In our design, we use 16 4-bit cells, 8 for positive and 8 for negative to form an 8-bit weight value with *add*. The accuracy is close to full precision accuracy.

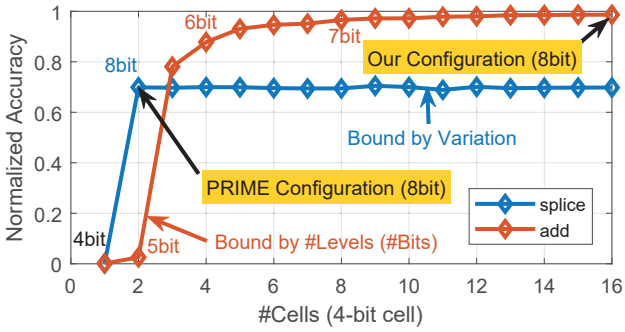


Figure 9. The normalized accuracy of VGG16 (normalized by the full precision accuracy) for the *splice* and *add* method with different number of cells used (4-bit for each cell).

7.3 Spatial Utilization

The *Spatial Utilization Bound* comes from the fact that weight matrices cannot fit crossbars perfectly. Moreover, we find that the neural synthesizer aggravates this situation. It introduces many small-scale weight matrices to implement operations such as reduction and max pooling. For example, in GoogleNet, after synthesis the pooling operations occupy 67.2% of PEs, which leads to the large gap between the *peak performance* and the *spatial utilization bound* in Figure 8c. To improve the utilization, from the hardware perspective, we could introduce different scales of PE to fit weight matrices better. From the software perspective, a future task is to find a better set of operations supported by hardware than the core-op.

8 Conclusion

By analyzing the bottlenecks and bounds for ReRAM-based NN acceleration, we propose a full system design of ReRAM-based NN accelerator, from the circuit level to the architectural and system level. Owing to the software system and massive hardware resources, it can support the function diversity and optimized execution of NN models on the proposed compact and efficient ReRAM PEs, achieving up to 1000× speedup compared to an existing ReRAM-based design, PRIME. Last but not least, the computational density, $38 TOPS/mm^2$, is also much higher than counterparts.

Acknowledgments

Thanks for the support from Beijing Innovation Center for Future Chip, the support of the Science and Technology Innovation Special Zone project, China, and the support of HUAWEI project. This work was also supported by NSF grant CCF 1500848, 1719160, 1725447, 1730309, 1740352, SRC nCORE NC2766-A, and CRISP, one of six centers in JUMP, a SRC program sponsored by DARPA.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.*, Kimberly Keeton and Timothy Roscoe (Eds.). USENIX Association, 265–283. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[2] Shyam Prasad Adhikari, Changju Yang, Hyongsuk Kim, and Leon O. Chua. 2012. Memristor Bridge Synapse-Based Neural Network and Its Learning. *IEEE Trans. Neural Netw. Learning Syst.* 23, 9 (2012), 1426–1435. https://doi.org/10.1109/TNNLS.2012.2204770

[3] Jorge Albericio, Patrick Judd, Tayler H. Hetherington, Tor M. Aamodt, Natalie D. Enright Jerger, and Andreas Moshovos. 2016. Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing. In *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*. IEEE Computer Society, 1–13. https://doi.org/10.1109/ISCA.2016.11

[4] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Awni Y. Hannun, Billy Jun, Tony Han, Patrick LeGresley, Xiangang Li, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Sheng Qian, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Chong Wang, Yi Wang, Zhiqian Wang, Bo Xiao, Yan Xie, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. 2016. Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016 (JMLR Workshop and Conference Proceedings)*, Maria-Florina Balcan and Kilian Q. Weinberger (Eds.), Vol. 48. JMLR.org, 173–182. http://jmlr.org/proceedings/papers/v48/amodei16.html

[5] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Architectural Support for Programming Languages and Operating Systems, ASPLOS '14, Salt Lake City, UT, USA, March 1-5, 2014*, Rajeev Balasubramonian, Al Davis, and Sarita V. Adve (Eds.). ACM, 269–284. https://doi.org/10.1145/2541940.2541967

[6] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. *CoRR* abs/1512.01274 (2015). arXiv:1512.01274 http://arxiv.org/abs/1512.01274

[7] Yu-Hsin Chen, Joel S. Emer, and Vivienne Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*. IEEE Computer Society, 367–379. https://doi.org/10.1109/ISCA.2016.40

[8] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. 2014. DaDianNao: A Machine-Learning Supercomputer. In *47th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2014, Cambridge, United Kingdom, December 13-17, 2014*. IEEE Computer Society, 609–622. https://doi.org/10.1109/MICRO.2014.58

[9] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory. In *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*. IEEE Computer Society, 27–39. https://doi.org/10.1109/ISCA.2016.13

[10] Jason Cong and Bingjun Xiao. 2011. mrFPGA: A novel FPGA architecture with memristor-based reconfiguration. In *Proceedings of the 2011 IEEE/ACM International Symposium on Nanoscale Architectures, NANOARCH 2011, San Diego, CA, USA, June 8-9, 2011*. IEEE Computer Society, 1–8. https://doi.org/10.1109/NANOARCH.2011.5941476

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*. IEEE Computer Society, 248–255. https://doi.org/10.1109/CVPRW.2009.5206848

[12] Xiangyu Dong, Cong Xu, Yuan Xie, and Norman P. Jouppi. 2012. NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory. *IEEE Trans. on CAD of Integrated Circuits and Systems* 31, 7 (2012), 994–1007. https://doi.org/10.1109/TCAD.2012.2185930

[13] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: shifting vision processing closer to the sensor. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture, Portland, OR, USA, June 13-17, 2015*, Deborah T. Marr and David H. Albonesi (Eds.). ACM, 92–104. https://doi.org/10.1145/2749469.2750389

[14] Varghese George. 2000. *Low energy field-programmable gate array.* Ph.D. Dissertation. University of California, Berkeley.

[15] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*. IEEE Computer Society, 243–254. https://doi.org/10.1109/ISCA.2016.30

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 770–778. https://doi.org/10.1109/CVPR.2016.90

[17] Miao Hu, John Paul Strachan, Zhiyong Li, Emmanuelle M. Grafals, Noraica Davila, Catherine Graves, Sity Lam, Ning Ge, Jianhua Joshua Yang, and R. Stanley Williams. 2016. Dot-product engine for neuromorphic computing: programming 1T1M crossbar to accelerate matrix-vector multiplication. In *Proceedings of the 53rd Annual Design Automation Conference, DAC 2016, Austin, TX, USA, June 5-9, 2016*. ACM, 19:1–19:6. https://doi.org/10.1145/2897937.2898010

[18] Giacomo Indiveri, Bernabé Linares-Barranco, Robert A. Legenstein, George Deligeorgis, and Themistoklis Prodromakis. 2013. Integration of nanoscale memristor synapses in neuromorphic computing architectures. *CoRR* abs/1302.7007 (2013). arXiv:1302.7007 http://arxiv.org/abs/1302.7007

[19] Yu Ji, Youhui Zhang, Wenguang Chen, and Yuan Xie. 2018. Bridge the Gap between Neural Networks and Neuromorphic Hardware with a Neural Network Compiler. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2018, Williamsburg, VA, USA, March 24-28, 2018*, Xipeng Shen, James Tuck, Ricardo Bianchini, and Vivek Sarkar (Eds.). ACM, 448–460. https://doi.org/10.1145/3173162.3173205

[20] Yu Ji, Youhui Zhang, Shuangchen Li, Ping Chi, Cihang Jiang, Peng Qu, Yuan Xie, and Wenguang Chen. 2016. NEUTRAMS: Neural network transformation and co-design under neuromorphic hardware

constraints. In *49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2016, Taipei, Taiwan, October 15-19, 2016*. IEEE Computer Society, 21:1–21:13. https://doi.org/10.1109/MICRO.2016.7783724

[21] Sung Hyun Jo, Ting Chang, Idongesit Ebong, Bhavitavya B Bhadviya, Pinaki Mazumder, and Wei Lu. 2010. Nanoscale memristor device as synapse in neuromorphic systems. *Nano letters* 10, 4 (2010), 1297–1301.

[22] Norman P. Jouppi, Cliff Young, Nishant Patil, David A. Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24-28, 2017*. ACM, 1–12. https://doi.org/10.1145/3079856.3080246

[23] Duckhwan Kim, Jaeha Kung, Sek M. Chai, Sudhakar Yalamanchili, and Saibal Mukhopadhyay. 2016. Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory. In *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*. IEEE Computer Society, 380–392. https://doi.org/10.1109/ISCA.2016.41

[24] Hyongsuk Kim, Maheshwar Pd. Sah, Changju Yang, Tamás Roska, and Leon O. Chua. 2012. Neural Synaptic Weighting With a Pulse-Based Memristor Circuit. *IEEE Trans. on Circuits and Systems* 59-I, 1 (2012), 148–158. https://doi.org/10.1109/TCSI.2011.2161360

[25] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning multiple layers of features from tiny images*. Technical Report. Citeseer.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger (Eds.). 1106–1114. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks

[27] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2018, Williamsburg, VA, USA, March 24-28, 2018*, Xipeng Shen, James Tuck, Ricardo Bianchini, and Vivek Sarkar (Eds.). ACM, 461–475. https://doi.org/10.1145/3173162.3173176

[28] Yann LeCun. 1998. The MNIST database of handwritten digits. *http://yann.lecun.com/exdb/mnist/* (1998).

[29] Yann LeCun et al. 2015. LeNet-5, convolutional neural networks. *URL: http://yann.lecun.com/exdb/lenet* (2015), 20.

[30] Robert LiKamWa, Yunhui Hou, Yuan Gao, Mia Polansky, and Lin Zhong. 2016. RedEye: Analog ConvNet Image Sensor Architecture for Continuous Mobile Vision. In *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*. IEEE Computer Society, 255–266. https://doi.org/10.1109/ISCA.2016.31

[31] Beiye Liu, Hai Li, Yiran Chen, Xin Li, Qing Wu, and Tingwen Huang. 2015. Vortex: variation-aware training for memristor X-bar. In *Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11, 2015*. ACM, 15:1–15:6. https://doi.org/10.1145/2744769.2744930

[32] Dao-Fu Liu, Tianshi Chen, Shaoli Liu, Jinhong Zhou, Shengyuan Zhou, Olivier Temam, Xiaobing Feng, Xuehai Zhou, and Yunji Chen. 2015. PuDianNao: A Polyvalent Machine Learning Accelerator. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '15, Istanbul, Turkey, March 14-18, 2015*, Özcan Özturk, Kemal Ebcioglu, and Sandhya Dwarkadas (Eds.). ACM, 369–381. https://doi.org/10.1145/2694344.2694358

[33] Xiaoxiao Liu, Mengjie Mao, Beiye Liu, Hai Li, Yiran Chen, Boxun Li, Yu Wang, Hao Jiang, Mark Barnell, Qing Wu, and Jianhua Yang. 2015. RENO: a high-efficient reconfigurable neuromorphic computing accelerator design. In *Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11, 2015*. ACM, 66:1–66:6. https://doi.org/10.1145/2744769.2744900

[34] Hadi Owlia, Parviz Keshavarzi, and Abdalhossein Rezai. 2014. A novel digital logic implementation approach on nanocrossbar arrays using memristor-based multiplexers. *Microelectronics Journal* 45, 6 (2014), 597–603. https://doi.org/10.1016/j.mejo.2014.04.014

[35] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS-W*.

[36] Yuriy V. Pershin and Massimiliano Di Ventra. 2010. Experimental demonstration of associative memory with memristive neural networks. *Neural Networks* 23, 7 (2010), 881–886. https://doi.org/10.1016/j.neunet.2010.05.001

[37] Mirko Prezioso, Farnood Merrikh-Bayat, Brian Hoskins, Gina C. Adam, Konstantin K. Likharev, and Dmitri B. Strukov. 2014. Training and Operation of an Integrated Neuromorphic Network Based on Metal-Oxide Memristors. *CoRR* abs/1412.0611 (2014). arXiv:1412.0611 http://arxiv.org/abs/1412.0611

[38] Brandon Reagen, Paul N. Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David M. Brooks. 2016. Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators. In *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*. IEEE Computer Society, 267–278. https://doi.org/10.1109/ISCA.2016.32

[39] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars. In *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*. IEEE Computer Society, 14–26. https://doi.org/10.1109/ISCA.2016.12

[40] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2014). arXiv:1409.1556 http://arxiv.org/abs/1409.1556

[41] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. 2017. PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning. In *2017 IEEE International Symposium on High Performance Computer Architecture, HPCA 2017, Austin, TX, USA, February 4-8, 2017*. IEEE Computer Society, 541–552. https://doi.org/10.1109/HPCA.2017.55

[42] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada,*

Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger (Eds.). 3104–3112. http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks

[43] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 1–9. https://doi.org/10.1109/CVPR.2015.7298594

[44] Andy Thomas. 2013. Memristor-based neural networks. *Journal of Physics D: Applied Physics* 46, 9 (2013), 093001.

[45] Ioannis Vourkas, Angel Abusleme, Vasileios G. Ntinas, Georgios Ch. Sirakoulis, and Antonio Rubio. 2016. A Digital Memristor Emulator for FPGA-Based Artificial Neural Networks. In *1st IEEE International Verification and Security Workshop, IVSW 2016, Sant Feliu de Guixols, Spain, July 4-6, 2016*. IEEE, 1–4. https://doi.org/10.1109/IVSW.2016.7566607

[46] Wei Wang, Tom T. Jing, and Brian Butcher. 2010. FPGA based on integration of memristors and CMOS devices. In *International Symposium on Circuits and Systems (ISCAS 2010), May 30 - June 2, 2010, Paris, France*. IEEE, 1963–1966. https://doi.org/10.1109/ISCAS.2010.5537010

[47] Lixue Xia, Peng Gu, Boxun Li, Tianqi Tang, Xiling Yin, Wenqin Huangfu, Shimeng Yu, Yu Cao, Yu Wang, and Huazhong Yang. 2016. Technological Exploration of RRAM Crossbar Array for Matrix-Vector Multiplication. *J. Comput. Sci. Technol.* 31, 1, 3–19. https://doi.org/10.1007/s11390-016-1608-8

[48] Qiangfei Xia, Warren Robinett, Michael W. Cumbie, Neel Banerjee, Thomas J. Cardinali, J. Joshua Yang, Wei Wu, Xuema Li, William M. Tong, Dmitri B. Strukov, Gregory S. Snider, Gilberto Medeiros-Ribeiro, and R. Stanley Williams. 2009. Memristor-CMOS Hybrid Integrated Circuits for Reconfigurable Logic. *Nano Letters* 9, 10 (2009), 3640–3645. https://doi.org/10.1021/nl901874j arXiv:https://doi.org/10.1021/nl901874j PMID: 19722537.

[49] Peng Yao, Huaqiang Wu, Bin Gao, Sukru Burc Eryilmaz, Xueyao Huang, Wenqiang Zhang, Qingtian Zhang, Ning Deng, Luping Shi, H-S Philip Wong, and He Qian. 2017. Face classification using electronic synapses. *Nature Communications* 8 (2017).

[50] Yue Zha and Jing Li. 2018. Liquid Silicon-Monona: A Reconfigurable Memory-Oriented Computing Fabric with Scalable Multi-Context Support. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2018, Williamsburg, VA, USA, March 24-28, 2018*, Xipeng Shen, James Tuck, Ricardo Bianchini, and Vivek Sarkar (Eds.). ACM, 214–228. https://doi.org/10.1145/3173162.3173167