A Peripheral Circuit Reuse Structure Integrated with a Retimed Data Flow for Low Power RRAM Crossbar-based CNN

Keni Qiu*, Weiwen Chen*, Yuanchao Xu*, Lixue Xia†, Yu Wang†, Zili Shao‡
*Capital Normal University, Beijing, China
†Tsinghua University, Beijing, China
‡Hong Kong Polytechnic University, Hong Kong, China

***Abstract*—Convolutional computations implemented in RRAM crossbar-based Computing System (RCS) demonstrate the outstanding advantages of high performance and low power. However, current designs are energy-unbalanced among the three parts of RRAM crossbar computation, peripheral circuits and memory accesses, and the latter two factors can significantly limit the potential gains of RCS. Addressing the problem of high power overhead of peripheral circuits in RCS, this paper proposes a Peripheral Circuit Unit (PeriCU)-Reuse scheme to meet power budgets in energy constrained embedded systems. The underlying idea is to put the expensive ADCs/DACs onto spotlight and arrange multiple convolution layers to be sequentially served by the same PeriCU. In the solution, the first step is to determine the number of PeriCUs which are organized by cycle frames. Inside a cycle frame, the layers are computed in parallel inter-PeriCUs while sequentially intra-PeriCU. Furthermore, a layer retiming technique is exploited to further improve the energy of RCS by assigning two adjacent layers within the same PeriCU so as to bypass the energy consuming memory accesses. The experiments of five convolutional applications validate that the PeriCU-Reuse scheme integrated with the retiming technique can efficiently meet variable power budgets, and further reduce energy consumption efficiently.**

I. Introduction

Power consumption and device size have become critical concerns in embedded computing system design where research explorations are often conducted in the power-constrained domains. Convolutional Neural Networks (CNN)-based methods are characteristic of computational-intensive and resource-consuming, and thus are challenging to be integrated into embedded systems. The prior studies show that convolutional computations which consist of numbers of multiply and accumulate (MAC) operations, serve as the most computationally expensive portion, namely 90% of computations in CNN [1]. The recent research has proposed that the emerging metal-oxide resistive switching random-access memory (RRAM) crossbar is able to do MAC operations through its natural analog computing in-situ in the cells where data are stored with extremely low energy [2]. Compared to the manner of executing MAC operations in GPU, ASIC and FPGA [3], [4], the RRAM crossbar-based CNN algorithms [1] demonstrate two highlights. First, the CNN algorithms can enjoy the advantages of a high density low power data store in RRAM device. Second, the MAC operations in CNN can be naturally completed just in the cells where CNN weights are stored, with no necessity of specific ALUs as well as weights transferring to ALUs. This paper targets RRAM crossbar-based Computing System (RCS) design for CNN. Specifically, this RCS offers a solution to meet constrained power budget of a system and further optimize the energy.

The RRAM crossbar network of a RCS accomplishes MAC in analog, and thus analog-to-digital and digital-to-analog converters (ADCs/DACs) are required in the mixed-signal system to bridge the digital part and the RRAM crossbar-based analog data processing unit. However, ADCs/DACs not only take up most of the chip area, but also consume much more power than RRAM devices. Furthermore, the current design is energy-unbalanced among the three parts of computation, peripheral circuits and memory access. It is observed that the peripheral circuits cost the most significant portion of power consumption in a RCS. The memory access also costs a big portion. As a result, the potential efficiency gains of RCS are significantly limited by the peripheral circuits and memory accesses [5], [6].

Addressing the problem of high power overhead of Peripheral Circuit Unit (*PeriCU*) and memory accesses in RCS, this paper proposes a *PeriCU-Reuse* scheme to meet low power budgets. The underlying idea of this study is to put the expensive peripheral circuits onto spotlight and arrange multiple convolution layers to be sequentially served by the same PeriCU. In the PeriCU-Reuse solution, the first step is to determine a hybrid structure which is organized by multiple PeriCUs. Then a layer retiming technique considering layer dependencies is conducted to save memory accesses under a smart layer schedule. The experiments validate that the PeriCU-Reuse scheme can efficiently meet constrained power budgets and the retiming technique can further reduce energy consumption.

In summary, this work makes the following main contributions.

- An idea of sequentially reusing the power consuming peripheral circuits is proposed to offer a solution to meet constrained power budgets of RCS.
- Given a power budget, a hybrid structure is proposed to organize convolutional layers to operate with multiple PeriCUs. The layers are computed in parallel inter-PeriCUs while sequentially intra-PeriCU. This computation is iteratively forwarding in a manner of cycle frames.
- A memory access-aware layer retiming technique considering layer dependencies is proposed to improve the energy efficiency through bypassing memory accesses.

The remainder of the paper is organized as follows. Section II introduces the RRAM crossbar-based convolution-

978-3-9819263-0-9/DATE18/©2018 EDAA

al computations. In Section III, the observations of power consumptions on peripheral circuits and memory accesses are presented. A motivational example is demonstrated to validate the efficacy of the PeriCU-Reuse idea. In Section IV, the solution details of the proposed PeriCU-Reuse scheme are presented. Section V discusses the experimental results. Conclusions are given in Section VI.

II. Background of RRAM crossbar-based CNN

The CNN is comprised of multiple computation layers that run in sequence. The previous work has proved that convolution operations take up over 90% of total computation [1]. Fig. 1(a) illustrates an example of a convolutional layer. A convolutional layer accepts P feature maps as input. In terms of each input feature map, a $K \times K$ kernel (filter) is employed to convolve with it at a sliding stride. During each shift, weights of the kernel are multiplied to overlapping values of input maps. And then resulting products are added-up together, generating the output feature maps.

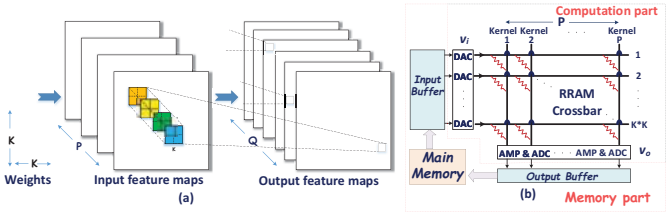

Fig. 1. (a) Convolutional layers in CNN; (b) RRAM crossbar-based convolutional computation.

The RRAM provides a promising solution to improve the performance and energy efficiency for future CNN accelerators [1], [7]. As illustrated in Fig. 1(b), the RRAM device is exploited to build cross-point structure, known as *RRAM crossbar array*, which can accomplish matrix-vector multiplication or vector-vector inner product naturally. The underlying support of this highlight arises from the observation that the voltage across a resistor is given by the product of the current passing through it and the resistor, thereby bringing an in-situ computing in memory. Note that we generally configure *ALL* the kernels on RRAM crossbar networks. In this way, RRAM can enjoy the advantage of in-situ matrix-vector multiplication with no necessity to consider the problems of expensive weight reconfigurations at runtime.

III. Observations and Motivations

In this section, we first point out that power is mostly consumed by peripheral circuits around RRAM crossbars when implementing convolutional computations on RCS. Then an example is illustrated to show that the power consumption of the peripheral circuits can be significantly reduced by reusing the expensive circuits. Motivated by this idea, a hybrid structure is proposed to reduce power consumption of RCS. Furthermore, potential of reducing memory accesses is observed by scheduling the layers in this hybrid structure. These two motivations lead to the contributions of this work.

A. An observation on HG of RCS

Although great performance improvement ($\sim \times 2360$) and energy reduction ($\sim \times 895$) have been reported regarding the architecture for RRAM-based convolutional computation [7], it does not mean that the current design is perfect. It can be seen that, besides RRAM crossbar, other digital and analog units such as ADCs, DACs and sensing amplifiers are also involved in RCS. In contrary to the high density and energy efficient RRAM crossbar, ADCs/DACs not only take up most of the chip area, but also consume much more power than RRAM crossbar. The large size and high power of peripheral circuits fundamentally limits further efficiency gains of a RCS.

Fig. 2 demonstrates the simulation results of power consumption breakdowns for a *HG* RCS in PSPICE. HG is a hand gesture recognition algorithm with two convolutional layers [8]. The parameter details are described in Section V. We can obtain the following observations.

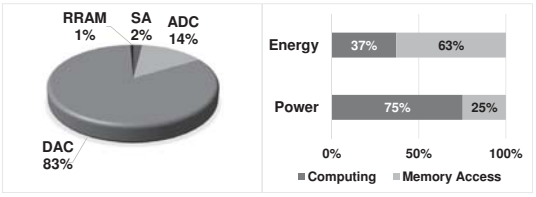

Fig. 2. (a) Power consumption of different analog components; (b) Power consumption of computation part and memory part of HG RCS.

- The performance of RCS outperforms that of FPGA implementation by thousands of times, which can well satisfy the requirements of computations on embedded systems.
- The power ratio of ADCs/ DACs to computation part of RCS is larger than 90%, presenting a very high overhead on peripheral circuits.
- The power and energy ratios of memory accesses to the entire RCS is 75% and 37%, also presenting a big overhead.

These insights inspire a design hint of trading performance off a satisfied power consumption.

B. Motivation 1: Peripheral circuit reusing and a hybrid structure

Fig. 3 illustrates our idea by comparing two RCS designs. It is known that the layers in one work item cannot process in parallel because one layer's inputs are fed by its previous layer's output. Therefore, a batch of work items are naturally processed in a *layer-level pipeline* fashion in order to achieve high throughput. Fig. 3(a) indicates that the two layers processed in parallel are from different work items of *M* and *M+1*. In this design, each layer's crossbar is served by a separate set of peripheral circuits to guarantee full parallelism. It can be seen that the *layer-to-RRAM crossbar* mapping is a conventional *one-to-one* manner.

Fig. 3(b) depicts a novel idea of design where the MAC computations of two different layers share the same peripheral circuits. The two layers are implemented sequentially. In other words, multiple layers' crossbar are served by one set of peripheral circuits. In this way, the power consumption can be reduced significantly. Note that *SEL* signals are needed to select the crossbar which should be activated. In this structure,

the shared Peripheral Circuit Unit (*PeriCU*) should provide sufficient resource to support the largest scale of kernels and feature maps for the layers that share the same PeriCU.

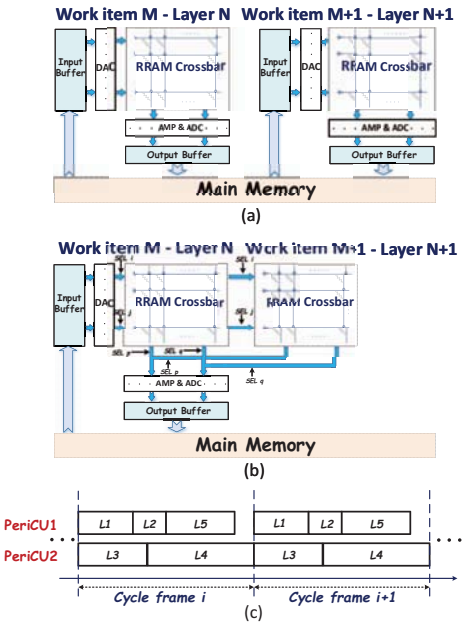

Fig. 3. (a) Design with a separate set of peripheral circuits for each layer; (b) Design with shared PeriCU for two layers; (c) The hybrid structure.

It can be found that the performance degrades resulting from non-paralleled performing with the PeriCU-Reuse design. However, it does not mean a disaster for the following reasons. First, the performance of RCS has already been boosted by the RRAM crossbar-based processing-in-memory (PIM) scheme. Therefore, even a degraded performance is also able to meet system requirements. Second, the non-full paralleled RCS does not mean a totally sequential layer implementation manner. A hybrid structure of fusing parallel and sequential executions is an ideal solution to meet both performance requirement and power budget. Fig. 3(c) shows a hybrid structure of implementing five convolutional layers. There are two PeriCUs to serve the five layers. That is, *L1, L2* and *L5* are served by PeriCU1 and *L3* and *L4* are served by PeriCU2. In this way, layers belonging to the same PeriCU are executed in sequence while layers belonging to different PeriCUs are executed in parallel. It exhibits a segmenting execution manner for multiple work items in the hybrid structure. Each segment execution is called a *Cycle Frame*. A cycle frame consists of multiple PeriCUs where each PeriCU consists of multiple convolutional layers which are operated sequentially to reuse the peripheral circuits.

Motivation 1 *In view of the foregoing potentials, this paper is motivated to propose a hybrid structured accelerator for convolutional computations through reusing the power consuming peripheral circuits in RCS.*

C. Motivation 2: memory access bypassing and layer scheduling

In the hybrid structure, it can be observed that if two adjacent layers of one work item are assigned to be served by the same PeriCU, these two layers can exchange data between the input and output buffers directly in a fine-granularity and thus bypassing the expensive memory accesses.

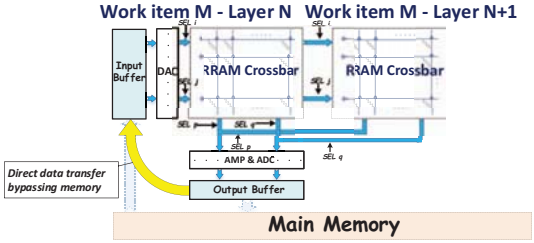

Fig. 4. A PeriCU-Reuse design with data transfer bypassing memory.

It is assumed that *Layer N* and *Layer N+1* are two adjacent layers of work item *M* in Fig. 4. The output of *Layer N* can be directly fed to *Layer N+1* such that data transferring from *Layer N-output buffer* to memory and from memory to *Layer N+1-input buffer* can be bypassed. Here the interleaved operation of *Layer N* and *Layer N+1* can be performed in a finer granularity. That is, instead of performing *Layer N* first and then *Layer N+1*, in the fine-grained operating mode, *Layer N+1* can start immediately when the output data of *Layer N* are already prepared for the first operation in *Layer N+1*. Therefore, layer scheduling in the hybrid structure can bring opportunities of bypassing memory accesses.

Motivation 2 *In the proposed hybrid structure, this paper is further motivated to propose a layer scheduling scheme by assigning adjacent layers in the same PeriCU to save memory accesses.*

The end objective of this paper is to meet some prespecified power budget and further optimize energy consumption. The concrete solution addresses the following two critical issues. The first is how to build the hybrid structure under a power budget. The second is how to schedule multiple layers in the hybrid structure to save memory accesses.

IV. PeriCU-Reuse Scheme

In this section, we present a low power RCS design for a resource-constrained lightweight embedded systems. The idea is to reuse the power consuming peripheral circuits (*PeriCU-Reuse*) in a hybrid structure consisting of parallel and sequential executions for inter-PeriCUs and intra-PeriCU respectively. The key issue is how many PeriCUs should be reused. Furthermore, motivated by the observation that memory accesses between two layers can be bypassed if these two adjacent layers in the same PeriCU are from the same work item, layer scheduling is studied as a further step to reduce energy by saving memory accesses. This step is directed by a Layer Flow Graph (LFG) model.

A. Energy and power models

Targeting to build the PeriCU-based hybrid structure and calculate the optimal layer schedule, we first derive the energy and power models of RCS. The relevant parameters are listed in Table I.

Energy Model Energy consists of two parts: the computation energy of RRAM crossbars and the memory access energy. Equation 1 expresses the energy consumption for performing *num* work items' convolutional computations.

TABLE I
KEY PARAMETERS IN A RCS.

Parameters	Descriptions
P_{comp}	power of RRAM crossbar, DACs/ADCs and analog circuits
T_{num}	execution time of *num* work items' convolutional computations
BA_{buff}	times of buffer accesses
E_{buff}	energy consumption for each buffer access
MA_{mem}	times of memory accesses
E_{DRAM}	energy consumption for each memory access
BW_{DRAM}	DRAM bandwidth
E_{total}	total energy of convolutional computations
P_{total}	total power of convolutional computations
P_{comp}^{ave}	average power on computing part

$$E_{total} = \sum P_{comp} \cdot T_{num} + BA_{buff} \cdot E_{buff} + MA_{mem} \cdot E_{DRAM} \quad (1)$$

Note that T_{num} consists of the time of computations, buffer accesses and memory accesses of executing *num* work items. It is assumed that data transferring and computations are performed in parallel with a *double buffering* scheme, thus T_{num} can be represented by the maximum value of computation time T_{comp} and data transferring time T_{mem}: $T_{num} = max\{T_{comp}, T_{mem}\}$.

Basically, T_{comp} varies under different structures. In the conventional *one-to-one* mapping structure where each set of peripheral circuits serves one separate layer's computation, T_{comp} can be determined by the layer pipelining execution. In the proposed hybrid structure where each PeriCU serves multiple layers' computation, T_{comp} is determined by the cycle frame-based execution.

Power Model The power model can be derived from the energy model as below.

$$P_{total} = P_{comp}^{ave} + (BA_{buff} \cdot BW_{buff} + MA_{mem} \cdot BW_{DRAM}) \cdot T_{num} \quad (2)$$

Besides the power consumption on RCS, the power is also closely correlated to the memory/buffer accesses and the operation time.

B. PeriCU Arrangement

This work proposes a hybrid structure consisting of multiple PeriCUs which are organized by cycle frames. Each cycle frame consists of C PeriCUs where $1 \leq C \leq L$ for a CNN with L convolutional layers as shown in Fig. 3(c). The layer assignment is fixed in each cycle frame such that cycle frames are executed iteratively to process batches of convolutional layers. In the hybrid structured cycle frame, layers of each PeriCU are processed sequentially while layers of different PeriCUs are executed in parallel.

Factor C Given a power budget, the number of PeriCUs (referred as *Factor C*) can be determined based on the proportion of the peripheral circuits in a RCS system.

Supposing the power budget is around $PeriC_{pb}$ percentage (e. g. 50%) of that operated in a conventional *one-to-one* $PeriC_{pb}$ mapping manner, the factor C can be calculated as below.

$$C = \lceil PeriC_{pb} \cdot L \rceil \quad (3)$$

Because we can further reduce the power by saving memory accesses, the above derivation is conservative and safe to meet the specified budget.

A naive cycle assignment It is known that each cycle frame sequentially processes its layers, with each layer having its own data in one cycle frame execution. In a simply safe way, a layer in each PeriCU only consumes data generated during previous cycle frame. For example, the output produced by *L1* in cycle frame i will be used as input for *L2* in cycle frame $i+1$. This means processing one work item requires 5 cycle frames of time for the convolutional computations with five convolutional layers. This is a simple and direct way to guarantee layer dependence within a cycle frame. Supposing *L4, L5* are of the highest computation demand in an application with five convolutional layers, Fig. 5 shows a naive cycle arrangement with a factor $C = 2$ where *L1, L2* and *L5* are assigned to PeriCU1 and *L3* and *L4* are assigned to PeriCU2. $W_i/L1$ means executing Layer 1 of work item i.

C. Layer Flow Graph (LFG) Modeling

As Fig. 3(b) indicates, since layers in one PeriCU are executed sequentially, which inherently matches the forwarding execution fashion of CNN, it is beneficial to put two adjacent layers of one work item in the same PeriCU to relieve the load of memory access. We are motivated to optimize layer scheduling inside cycle frames so as to further reduce energy and improve performance. It is known that cycle frames are uniform and they are executed in a loop form as shown in Fig. 3(c). This work proposes a retiming technique to schedule the layers in the uniform cycle frames. To this end, we first need to model layer dependency of a cycle frame using a Layer Flow Graph (LFG).

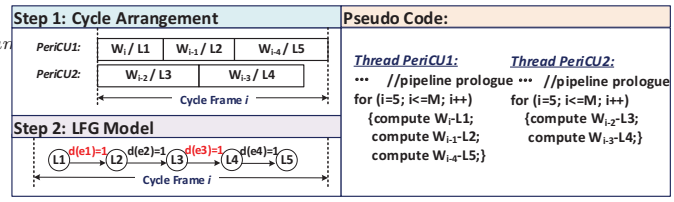

Fig. 5. The naive PeriCU-Reuse scheme.

Definition: Layer Flow Graph (LFG) *A Layer Flow Graph (LFG) G=<V, E, d> is an edge weighted directed graph, where V is the set of layers and E is the set of dependency edges. For an edge* $u \xrightarrow{e} v$, *its delay vector d(e) represents any delay vector between the tail computation node u and the head computation node v.*

In the LFG example in Fig. 5, *d(e1)=1* implies the *L2* must wait 1 iteration of cycle frame before it can use the input of *L1*. If there is no edge between two nodes, it implies there is no direct input dependence between these two layers. The pseudo codes illustrate the execution of the two PeriCUs.

D. Layer Retiming

In the naive assignment, each layer of a cycle frame only consumes data generated from the previous cycle frame. In this way, since any layers inside one PeriCU are from different work items, the benefits of memory bypassing as indicated in Fig. 4 can not be enjoyed. If we adopt the idea of scheduling

two adjacent layers of one work item to the same PeriCU, the energy consuming memory accesses can be reduced and the prologue/epilogue of the cycle frame sequence can be shortened as well. Motivated by the above idea, this work proposes a loop retiming technique to schedule the layers in a cycle frame so as to reduce memory accesses and save energy.

Retiming principles Loop retiming allows rearranging the layers of a cycle frame by moving the layers across the original iteration boundary. A retiming vector r is a function that redistributes the nodes in one iteration. The retiming vector $r(u)=\omega$ of a node u represents the offset between the original iteration containing u and the one after retiming. A *legal retiming vector* should preserve the original data dependency direction. *In the LFG model, specifically, the retimed edge delay should remain non-negative value to preserve correct dependency.* After a legal retiming, a new LFG G^r is created, and each iteration still has one execution of each layer in G^r.

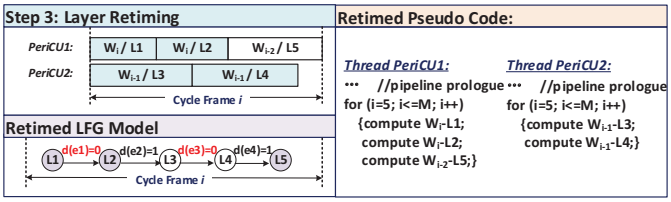

Fig. 6. The retimed PeriCU-Reuse scheme with retiming vectors *r(L1)=-1, r(L4)=-1, r(L5)=-1*.

For the example in Fig. 6, the retimed code and the retimed LFG are depicted in *Step 3*. For the retimed node, all the edge delays connected to it are changed. *Its outgoing edge delays add the retiming vector and its incoming edge delays minus the retiming vector.* It can be observed that the retimed schedule allows more zero delays in a cycle frame, resulting in memory access bypassing, further leading to power and energy saving.

Energy-aware retiming scheme The optimal retiming problem for the loop can be addressed as determining a legal retiming vector for each node (layer) so as to achieve the minimal energy consumption. The reduced memory accesses can be represented as below.

$$MA^{redu} = \sum_{tail-layer\ w/d=0} MA^{L-in} + \sum_{head-layer\ w/d=0} MA^{L-out} \quad (4)$$

In the above equation, MA^{redu} denotes the total number of reduced memory accesses. $\sum_{tail-layer\ w/d=0} MA^{L-in}$ and $\sum_{head-layer\ w/d=0} MA^{L-out}$ denote the number of reduced from-memory accesses and the number of reduced to-memory accesses when assigning two adjacent layers (*d=0*) of the same work item in one PeriCU. The energy saving problem can be formulated as follows.

Objective: *Maximize* MA^{redu}
Subject to:
(1) $d^r(e) = \omega + \delta(t) - \delta(h)$
(2) $d^r(e) \geq 0$ (L_u, $L_v \in$ one PeriCU)
(3) $d^r(e) > 0$ (L_u, $L_v \notin$ one PeriCU) (5)

In the constraints, ω denotes the original delay of edge e before retiming, i.e. $d(e) = \omega$, and δ_t and δ_h denote the retiming vectors for the tail node and the head node of edge e respectively. By solving this ILP problem, the optimal retiming solutions for the layers can be obtained. For the example in Fig. 5, the optimal retiming solution is: *r(L1)=-1, r(L4)=-1, r(L5)=-1* as shown in Fig. 6.

Note that the retiming is conducted based on a certain cycle arrangement. The sub-optimal retiming solution may vary for different cycle arrangements. In order to find a global optimal retiming solution, we calculate the optimal retiming vectors for each layer by searching the full cycle arrangement space.

V. Experimental Evaluation

We evaluate the proposed PeriCU-Reuse approach integrated with the retiming optimization to implement convolutional computations of CNN algorithms. Given a power budget, we first use the proposed cycle arrangement method to determine the number of PeriCUs of a cycle frame. Directed by the LFG model, the layer retiming strategy is applied to obtain the optimal layer schedule in a cycle frame.

A. Experimental Setup

Detailed parameters of circuits and RRAM crossbar are summarized in Table II [1]. The maximum amplitude of input voltage is set to 0.5V to achieve an approximate linear I-V relationship of RRAM devices. The amplitude of the write pulse is set to -1.2V and the read pulse is set to 0.1V. The pulse width is set to 5ns. The maximum acceptable deviation of RRAM conductance state is set to 1%. An off-chip DDR3 DRAM with access energy E_{DRAM} of 70pJ/bit and bandwidth of 12.8GB/s [9] are adopted in this work. The simulation results of the computation amount of each kernel are achieved with PSPICE, and the results of the complete convolution computations of one work item as well as a batch of work items are derived from MATLAB. We apply our design to five practical CNN algorithms where the convolutional layers and the concerned kernels are listed in Table III.

TABLE II
Experimental Setup [1] [9].

Technology Node	65nm
RRAM Resistance Range	500Ω - 500kΩ
R_s	2kΩ
AMP	4.8mW
ADC	8bit, 35mW
DAC	8bit, 40mW
Frequency	800MHz
E_{DRAM}	70nJ/bit
BW_{DRAM}	12.8GB/s

Three versions of results are evaluated: the traditional one-to-one mapping design (referred as *Conventional version*), the PeriCU-Reuse design with naive layer assignment (*Naive PeriCU-Reuse version*) and the PeriCU-Reuse designs with memory access-aware retiming optimization (*retimed PeriCU-Reuse version*). The *Naive PeriC-Reuse* design only considers cycle arrangement in a PeriCU-Reuse manner, but no retiming to re-schedule the layers in a cycle frame. The *retimed PeriCU-Reuse* design takes a further step to retime layers of a cycle frame so as to optimize energy.

TABLE III
SEVERAL PRACTICAL CNN ALGORITHMS.

Workloads	Layers	Kernels	Layer Size
PV	Input		1@50×50
	C1	8@6×6×1	8@45×45
	C3	12@3×3×8	12@20×20
	C5	16@3×3×12	16@8×8
	C6	10@3×3×16	10@6×6
	C7	6@3×3×10	6@4×4
FR	Input		1@32×32
	C1	4@5×5×1	4@28×28
	C3	16@4×4×4	16@10×10
LeNet	Input		1@32×32
	C1	6@5×5×1	6@28×28
	C3	16@5×5×6	16@10×10
HG	Input		1@28×28
	C1	6@5×5×1	6@24×24
	C3	12@4×4×6	12@8×8
AlexNet	Input		3@224×224
	C1	48@11×11×3	48@55×55
	C3	128@5×5×48	128@27×27
	C5	192@3×3×128	192@13×13
	C6	192@3×3×192	192@13×13
	C7	128@3×3×192	128@13×13

B. Results and Analysis

We compare the three versions by evaluating average power and energy. The power budgets are set as 50% for those benchmarks with two layers and 40% and 60% for those benchmarks with five layers compared to the *conventional version*. According to Equation 3, the calculation of the factor C is safe enough to meet power budgets. For the benchmarks *FR*, *LeNet* and *HG* with two convolutional layers, C is set as one to meet the 50% power budget. For the benchmarks *PV* and *AlexNet* with five convolutional layers, C is set as two or three to meet the 40% or 60% budgets respectively. All the results are normalized to the *conventional version*.

1) Average power consumption: It can be seen that the proposed PeriCU-Reuse approach can meet the power budget requirements by appropriate cycle arrangement for all algorithms in Fig. 7. The reason is that the layers served by the same PeriCU are executed sequentially instead in a parallel-style, thereby offering an intriguing solution to deploy power-consuming parts. As mentioned above, the hybrid structure is defined by the factor C which is determined by the largest scales among the layers. Actually the average power consumption can be further smaller. The average power consumption of the *retimed PeriCU-Reuse* version is reduced by 85.3% and 14.9% over the *Conventional* version and the *Naive PeriC-Reuse* version respectively across all the power budgets. *AlexNet* (C=3) exhibits the most power reduction with its *retimed PeriCU-Reuse* version.

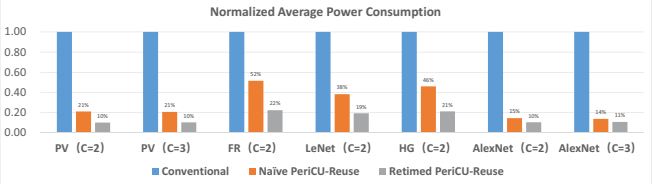

Fig. 7. Average power consumption evaluation.

2) Energy consumption: Since we trade performance off the power consumption to meet a given power budget, the total energy of the *Naive PeriCU-Reuse* version is similar to the *conventional* version. Nevertheless, the *retimed PeriCU-Reuse* version can reduce energy because memory access bypassing is enable to save energy and latency of memory accesses between two adjacent layers of the same work item in one PeriCU. Table IV shows that 40.6% energy can be saved by the *retimed PeriCU-Reuse* version.

TABLE IV
ENERGY REDUCTION OF THE *retimed PeriCU-Reuse* VERSION.

Workloads	PV2	PV3	FR	LeNet	HG	AlexNet2	AlexNet3
Ene. Redu. (%)	48.2	40.4	20.5	29.5	29.6	61.6	54.4

VI. CONCLUSION

The peripheral circuits account the most power consuming parts in a RCS, which limits the gains from adopting the efficient RRAM crossbar-based convolutional computing. This paper proposes a PeriCU-Reuse scheme to achieve low power by sequentially reusing the power consuming peripheral circuits such as ADCs/DACs. A hybrid structure organized by the concept of cycle frame is proposed to arrange multiple convolutional layers on PeriCUs under a given power budget. A layer retiming technique is proposed to schedule the layer delays in cycle frames and further reduce energy by mitigate memory accesses. The experimental results show that the PeriCU-Reuse scheme offers a feasible design to meet power requirements while optimizing energy efficiency of RCS.

ACKNOWLEDGMENT

This work is supported by the grants from Beijing Advanced Innovation Center for Imaging Technology, Beijing Innovation Center for Future Chip, National Natural Science Foundation of China [Project No. 61502321, 61622403] and the Project of Beijing Municipal Education Commission [Project No. KM201710028016].

REFERENCES

[1] L. Xia, T. Tang, W. Huangfu, M. Cheng, X. Yin, B. Li, Y. Wang, and H. Yang, "Switched by input: Power efficient structure for RRAM-Based convolutional neural network," in *DAC'16*, 2016, pp. 125:1–125:6.

[2] B. Li, Y. Wang, Y. Chen, H. H. Li, and H. Yang, "ICE: Inline calibration for memristor Crossbar-Based computing engine," in *DATE'14*, 2014, pp. 1–4.

[3] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A Machine-Learning supercomputer," in *Micro'14*, 2014, pp. 609–622.

[4] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-Based accelerator design for deep convolutional neural networks," in *FPGA'15*, 2015, pp. 161–170.

[5] B. Li, L. Xia, P. Gu, Y. Wang, and H. Yang, "MErging the interface: Power, area and accuracy co-optimization for RRAM crossbar-based Mixed-Signal computing system," in *DAC'15*, 2015, pp. 1–6.

[6] P. Panda, A. Sengupta, S. S. Sarwar, G. Srinivasan, S. Venkataramani, A. Raghunathan, and K. Roy, "Cross-layer approximations for neuromorphic computing: From devices to circuits and systems," in *DAC'16*, 2016, pp. 1–6.

[7] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *ISCA'16*, 2016, pp. 27–39.

[8] H. I. Lin, M. H. Hsu, and W. K. Chen, "Human hand gesture recognition using a convolution neural network," in *CASE'14*, 2014, pp. 1038–1043.

[9] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei, "Deep convolutional neural network architecture with reconfigurable computation patterns," *IEEE TCAD*, vol. PP, no. 99, pp. 1–14, 2017.