An Energy-efficient Deep Belief Network Processor Based on Heterogeneous Multi-core Architecture with Transposable Memory and On-chip Learning

Jiajun Wu, *Member*, *IEEE*, Xuan Huang, *Member*, *IEEE*, Le Yang, *Member*, *IEEE*, Jipeng Wang, *Member*, *IEEE*, Bingqiang Liu, *Member*, *IEEE*, Ziyuan Wen, *Member*, *IEEE*, Juhui Li, Guoyi Yu, *Member*, *IEEE*, Kwen-Siong Chong, *Senior Member*, *IEEE* and Chao Wang, *Senior Member, IEEE*

***Abstract*—With the growing interest of edge computing in the Internet of Things (IoT), Deep Neural Network (DNN) hardware processors/accelerators face challenges of low energy consumption, low latency, and data privacy issues. This paper proposes an energy-efficient processor design based on Deep Belief Network (DBN), which is one of the most suitable DNN models for on-chip learning. In this study, a thorough algorithm-architecture-circuit design optimization method is used for efficient design. The characteristics of data reuse and data sparsity in the DBN learning algorithm inspires this study to propose a heterogeneous multi-core architecture with local learning. In addition, novel circuits of transposable weight memory and sparse address generator are proposed to reduce weight memory access and exploit neuron state sparsity, respectively, for maximizing the energy efficiency. The DBN processor is implemented and thoroughly evaluated on Xilinx Zynq FPGA. Implementation results confirm that the proposed DBN processor has excellent energy efficiency of 45.0 pJ per neuron-weight update, which has been improved by 74% against the conventional design.**

***Index Terms*—Edge computing, Deep Belief Network (DBN), on-chip learning, algorithm-architecture-circuit co-design, data reuse, data sparsity, heterogeneous multi-core architecture, transposable memory.**

I. Introduction

With the breakthrough of Deep Neural Networks (DNN) [1], advanced Artificial Intelligence (AI) models have been attracting significant attention across almost all domains of technology. Among the existing AI models, the DNN model [1] has shown superior performance in many applications of everyday lives, such as object recognition [2] and speech processing [3]. However, in the power-limited AI applications such as edge computing in the AI Internet-of-Things (AIoT), existing power-hungry AI platforms like Graph Processing Unit (GPU) have an unacceptable drawback due to the large energy consumption of DNN processing [4]. Moreover, local DNN training is required to address the problems of latency, privacy data and personal security on edge devices. Recently, various DNN accelerators or processors with on-chip learning capability have been proposed for AIoT edge devices [5]-[6]. These hardware designs have not paid much attention to the memory hierarchy optimization to improve energy efficiency by reducing external memory access or data movement during on-chip learning and inference. Nevertheless, previous work [7] has reported that DNN processing requires frequent access to power-hungry external memories for supporting complex network structures. In general, off-chip data movement is 2-3 orders of magnitude more expensive in terms of energy consumption compared to on-chip data movement. Thus, for targeting energy-constrained AIoT edge devices, it is essential to implement an energy-efficient DNN accelerator with a light-computation model as well as realization of local storing of DNN parameters to reduce DNN computation and accesses of off-chip memory, respectively.

As one of DNN models, Deep Belief Network (DBN) model [8] has been attracting significant attention in AIoT edge computing applications. The DBN is suitable for implementing hardware accelerators for AIoT edge devices due to three major reasons. Firstly, the input data and activations of DBN are binary, which significantly reduces both the amounts of model computation and sizes of memories compared with other float-point or high-precision fixed-point DNN models. Secondly, a DBN learning algorithm is actually divided into local learning of several Restricted Boltzmann Machines (RBM). This kind of

Manuscript received xxx, 2021; revised xxx XX, 2021; accepted xxx, 2021. This work was partially supported by National Key R&D Program of China (2019YFB1310001), National Natural Science Foundation of China (61974053) and Fundamental Research Funds of the Central Universities (2019KFYXJJS049). This paper was recommended by Associate Editor XXX.XXX. (*Corresponding Author: Chao Wang*).

J. Wu, X. Huang, L. Yang, J. Wang, B. Liu, Z. Wen and G. Yu are with the School of Optical and Electronic Information, Huazhong University of Science and Technology, Wuhan City, Hubei Province, China 430074. (*J. Wu and X. Huang contributed equally to this paper*).

J. Li is with the Nations Innovation Pte Ltd Singapore, Singapore.

K-S. Chong is with the Temasek Laboratories, Nanyang Technological University, Singapore.

C. Wang is with the School of Optical and Electronic Information, Huazhong University of Science and Technology, and also with Wuhan National Laboratory of Optoelectronics, Wuhan City, Hubei Province, China 430074 (chao_wang_me@hust.edu.cn; chao.wang.1978@hotmail.com).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JETCAS.2021.3114396, IEEE Journal on Emerging and Selected Topics in Circuits and Systems

learning rule limits dataflow of update signals locally and reduces the external memory access so as to significantly reduce the energy consumed by data movement globally as required by the most popular CNN/MLP models with backpropagation (BP) algorithms transferring the data across different layers. Last but not least, conventional DBN models using unsupervised learning are very suitable for edge AIoT devices, as unsupervised learning reduces the need for labeled data that is difficult to access for in AIoT devices.

In recent years, researchers have developed a few DBN hardware accelerators/processors with on-chip learning, which can be classified into Application Specific Integrated Circuit (ASIC) based designs [5], [9]-[10] and Field Programmable Gate Array (FPGA) based designs [11]-[14]. These works have focused on hardware-level to accelerate the DBN processing by exploiting parallelism over processing elements (PEs) and proposing specific calculation circuits for multiply-and-accumulate (MAC) operations. However, data reuse of transposed weights and data sparsity of neuron states during learning and inference phases of DBN processing have not been investigated and exploited to improve energy efficiency of DBN hardware accelerator designs.

Therefore, we investigate the data reuse of transposed weights and data sparsity of neuron states by algorithm-level analysis of DBN, and propose an energy-efficient DBN processor with a heterogeneous multi-core architecture and novel circuits of transposable weight memory and sparsity address generator, which is extended from on our previous work [15]. The contributions of this study can be summarized at the three design hierarchical levels from the algorithm-architecture-circuit co-design perspective:

Algorithm Level Analysis- Data Reuse of Local Transposed Weights and Sparsity of Neuron States: The data flow during DBN learning and inference has been analyzed at algorithm level to exploit the data parallelism and locality. The local DBN learning is performed on multiple two-layer RBM components that constitute the DBN model. From our previous work in [15], it has been found that local weights between the two layers of a RBM component can be perfectly transposed reuse during different local learning phases to avoid frequent memory access between chip and off-chip memory. Data sparsity of the neuron states has also been investigated at the algorithm level analysis for the circuit design.

Architecture Level Design - Heterogeneous Multi-core Architecture: To efficiently exploit the aforementioned data parallelism and locality, as well as data reuse of transposable weights, a heterogeneous multi-core architecture is proposed. By dividing the weights between two layers into a number of data blocks, the DBN processing is mapped into highly parallel computational cores and constrained into local data movement of the proposed multi-core architecture.

Circuit Level Design - Transposable Local Memory and Sparsity Address Generator: To exploit the data reuse of local weights found from the algorithm-level analysis, transposable local memory is designed for reducing multiple memory R/W access between different phases of learning from the conventional weight memory organization. In addition, a sparse address generator (SAG) is also designed to skip reading the local weights that do not contribute to the MAC operations due to the observed sparsity of neuron states (i.e., a large number of zero neuron states).

By the above algorithm design analysis and architecture-circuit co-design, the proposed DBN processor can significantly achieve energy efficiency improvement. FPGA implementation and evaluation are carried out to confirm the proposed DBN processor design has achieved 75.4% higher energy efficiency.

The rest of this paper is organized as follows. Section II introduces the DBN model and presents the DBN algorithm-level analysis for proposed hardware design. Details of the architecture-circuit co-design are given in Section III, which include efficient data mapping, multi-core architecture and specific circuit design. Section IV presents FPGA implementation results with hardware resources, learning speed and energy consumption. Finally, Section V makes a conclusion.

II. Algorithm-level Analysis of Deep Belief Network

A. The Structure and Learning Rule of DBN

Fig. 1 shows the DBN network topology, which is similar to fully connected multilayer perceptron (MLP). A DBN model consists of M unsupervised layers and an output layer. In the former M layers, every adjacent two layers form several pairs (e.g., M-1 pairs in Fig. 1) and weights of the pairs are trained as unsupervised learning rule. Unsupervised learning reduces the need for labeled data, which is important for intelligent IoT edge devices. Each pair is regarded as a Restricted Boltzmann Machine (RBM) [16]. All neuron states in DBN layers are binarized i.e., 0 or 1. Thus, training and test dataset of DBN model are also binarized, such as binarized images and sequences. This character makes computations of DBN learning simpler, which also fits the applications for AI edge devices. The small-scale classifier in the last layer is used for enhancing classification performance and it is trained as supervised learning rule after all the RBMs finish their unsupervised learning.

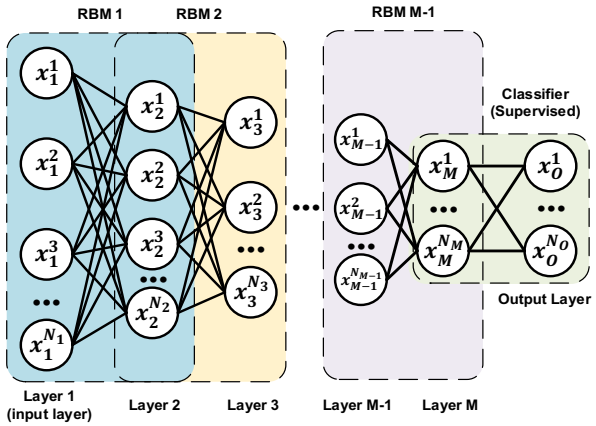


Fig. 1. An unsupervised DBN model consisting of M fully connected layers with N_k neurons in the k-th layer ($k = 1, 2, ..., M$) and an output layer with N_O neurons. The first layer is the input layer which receives input binarized data. The last layer is the output layer that generates recognition/classification results in inference possessing.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JETCAS.2021.3114396, IEEE Journal on Emerging and Selected Topics in Circuits and Systems

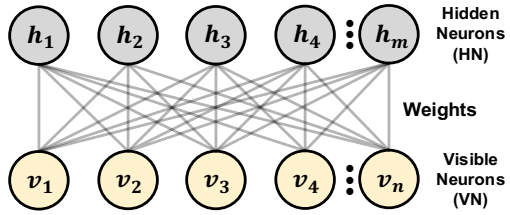


Fig. 2. Structure of Restricted Boltzmann Machine (RBM) with *n* visible neurons and *m* hidden neurons, which is the basic component of DBN unsupervised learning.

The learning process of DBN is divided into unsupervised learning of each RBM. Fig. 2 presents the structure of a $n \times m$ RBM model with n visible neurons ($v_1 \sim v_n$), m hidden neurons ($h_1 \sim h_m$) and connection weights. The RBM model works as a fully connected encoder-decoder pair. Visible neurons (VN) are initialized with input binarized vector and hidden neurons (HN) are sampled and generated by visible neurons. With iterations of the unsupervised learning rule, a well-trained RBM model is able to reconstruct a vector of visible neurons, which is as same as possible with the input vector.

As Fig. 3 illustrates, the most popular algorithm for RBM unsupervised learning is Contrastive Divergence (CD) [8]. In phase 1, assume one training data is input as binarized vector and distributed in visible neurons $v_1^0 \sim v_n^0$ (superscript 0 indicates the number of iterations). In phase 2, the visible neurons will be used to generate hidden neurons, which is named forward generation (FG). In this phase, weighted sums from visible neurons to hidden neurons are calculated firstly, as the following equation,

$$z_j^{h^0} = \sum_{i=1}^{n} v_i^0 \times w_{i,j} \tag{1}$$

where $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, m$ and $w_{i,j}$ stands for the weight between i-th visible neuron and j-th hidden neuron. Assuming the connection weights are formulated as matrix $\boldsymbol{W}$ with m rows and n columns, and initial column vector is $\boldsymbol{V^0}$ (formed with $v_1^0 \sim v_n^0$), Equation (1) can be written as matrix form:

$$\boldsymbol{Z^{H^0}} = \boldsymbol{W V^0} \tag{2}$$

After getting the weighted sums, hidden neuron states will be generated by following equation,

$$h_j^0 = \begin{cases} 1, & \sigma(z_j^{h^0}) \geq RN \\ 0, & \sigma(z_j^{h^0}) < RN \end{cases} \tag{3}$$

where RN is a random number and it ranges from 0 to 1. The $\sigma(x)$ in Equation (3) is an activation function. It is commonly selected as sigmoid function, which is given by

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{4}$$

This random generation process for neuron states is named Gibbs Sampling. Then, the CD algorithm will use the generated hidden neurons ($h_1^0 \sim h_m^0$) to reconstruct visible neurons. This phase is backward reconstruction (BR). Similar to FG, in BR phase the weighted sums are calculated firstly as:

$$z_i^{v^1} = \sum_{j=1}^{m} h_j^0 \times w_{i,j} \tag{5}$$

or matrix form based on row vector $\boldsymbol{H^0}$ (formed with $h_1^0 \sim h_m^0$):

$$\boldsymbol{Z^{V^1}} = \boldsymbol{H^0 W^T} \tag{6}$$

Next, new visible neuron states $v_1^1 \sim v_n^1$ are reconstructed as

$$v_i^1 = \begin{cases} 1, & \sigma(z_i^{v^1}) \geq RN \\ 0, & \sigma(z_i^{v^1}) < RN \end{cases} \tag{7}$$

The results can be written as a column vector $\boldsymbol{V^1}$ (formed with $v_1^1 \sim v_n^1$). After reconstructing new visible neuron states, CD algorithm takes FG phase again to generate new hidden neuron states $h_1^1 \sim h_m^1$ and corresponding row vector $\boldsymbol{H^1}$, as Equation (8)-(10) illustrate:

$$z_j^{h^1} = \sum_{i=1}^{n} v_i^1 \times w_{i,j} \tag{8}$$

$$\boldsymbol{Z^{H^1}} = \boldsymbol{W V^1} \tag{9}$$

$$h_j^1 = \begin{cases} 1, & \sigma(z_j^{h^1}) \geq RN \\ 0, & \sigma(z_j^{h^1}) < RN \end{cases} \tag{10}$$

There are 4 types of vectors of neuron states after 2 FG phases and 1 BR phase. They are $\boldsymbol{V^0}$, $\boldsymbol{H^0}$, $\boldsymbol{V^1}$ and $\boldsymbol{H^1}$, as the 4 different colors shown in Fig. 3. After that, CD algorithm will take these vectors to update the weight matrix $\boldsymbol{W}$. Assuming update value of weights are written as matrix $\Delta\boldsymbol{W}$ which has same scale with $\boldsymbol{W}$. This matrix is given by:

$$\Delta\boldsymbol{W} = \delta(\boldsymbol{V^0 H^0} - \boldsymbol{V^1 H^1}) \tag{11}$$

where δ is a constant value which stands for learning rate of CD algorithm. Finally, $\boldsymbol{W}$ is updated by $\Delta\boldsymbol{W}$ and the matrix $\boldsymbol{W'}$ in last iteration:

$$\boldsymbol{W} = \boldsymbol{W'} + \Delta\boldsymbol{W} \tag{12}$$

After getting final $\boldsymbol{W}$, one iteration of the CD algorithm completes, and another training data will be input to visible neurons for next iteration. The learning iteration is repeated until convergence, which denotes either the maximum number of iteration times is reached, or the Euclidean norm of $\boldsymbol{V^1}$ and $\boldsymbol{V^0}$ (i.e. $\|\boldsymbol{V^1}, \boldsymbol{V^0}\|_2$) is lower than the predetermined threshold[8], and then this RBM finishes its unsupervised learning. In inference process, RBM only generates $\boldsymbol{H^0}$ based on input data $\boldsymbol{V^0}$. The training datasets of the next RBM are the inference results of the current RBM. For instance, in Fig. 1, input data of the RBM 1 is from the training datasets, while the input data of RBM 2 is from the inference results of RBM 1. In other words, the DBN unsupervised learning is processed by each RBM, and the classifier in the output layer will use the inference results of RBM M-1 for supervised learning itself.

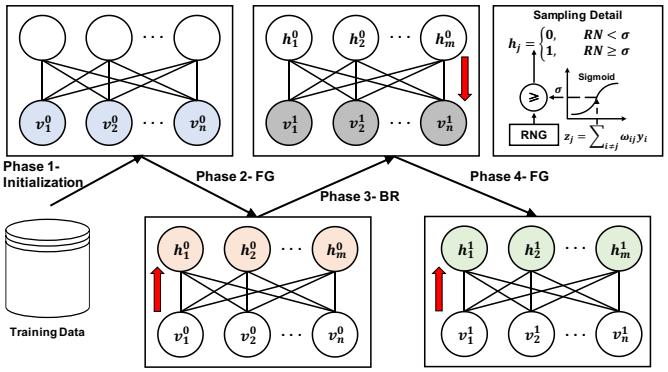


Fig. 3. Process of contrastive divergence (CD) algorithm before updating, which needs two phases of forward generation (FG) and one phase of backward reconstruction (BG).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JETCAS.2021.3114396, IEEE Journal on Emerging and Selected Topics in Circuits and Systems

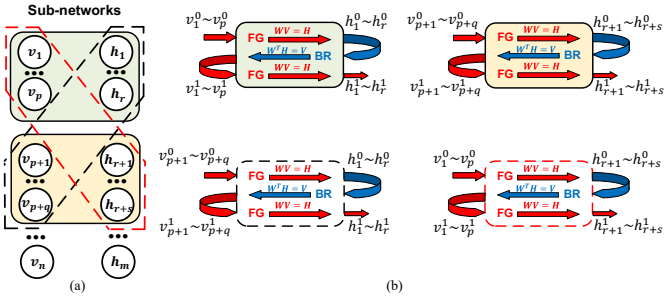


Fig. 4. (a) Sub-networks division of RBM unsupervised learning and (b) explanation of weight transposed reuse. All the weights blocks are reused in FG and BR phases.

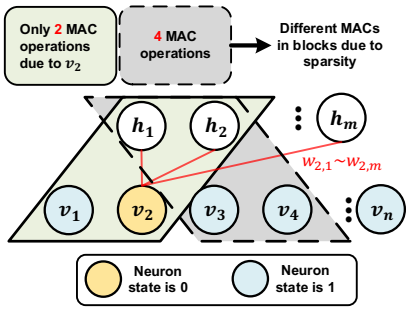


Fig. 5. Illustration of data sparsity in RBM unsupervised learning. Note that we only consider neuron state sparsity in this work.

B. Data Flow Analysis of DBN

Thorough algorithm-level analysis can greatly enhance efficiency of hardware design. Therefore, we make an effect on algorithm-level data flow to guide architecture- and circuit-level optimizations, which has not been thoroughly explored in previous works. The data characters of DBN unsupervised learning can be summarized as transposed reuse and sparsity.

1) Data Transposed Reuse: Motivated by numerous DNN accelerators/processors works mentioned in Section I, we find that the key optimization point in hardware level is parallel computation. For fully connected models such as MLP and RBM, the most computationally intensive process is weighted sums in Equation (1), (5) and (8). To accelerate this process in parallel, it is natural to break up the RBM into a series of smaller sub-networks and the weights are accordingly divided into several weight blocks, so that all the sub-networks operate in parallel. Fig. 4 (a) takes an example to illustrate the division of sub-networks, in which v_1~v_{p+q}, h_1~h_r and corresponding weights $w_{1,1}$~$w_{p+q,r+s}$ are divided into four different sub-networks. Note that the shown subnetworks in Fig. 4 (a) are only for illustration. In fact, the neurons v_1~v_{p+q} and h_1~h_{r+s} are connected to many subnetworks. The solid line boxes are to distinguish the neurons partitioning whereas the dashed line boxes are to distinguish the weights partitioning. In such data flow, each sub-network calculates partial sums itself, and all partial sums will be accumulated to get the final weighted sum, as Equation (13), (14) presents.

$$z_1^{h^0} = \sum_{i=1}^{p} v_i^0 \times w_{i,1} + \sum_{j=p+1}^{p+q} v_i^0 \times w_{i,1} + \cdots + v_n^0 \times w_{n,m} \quad (13)$$

$$\begin{cases} Partial\ sum\ 1 = \sum_{i=1}^{p} v_i^0 \times w_{i,1} \\ Partial\ sum\ 2 = \sum_{j=p+1}^{p+q} v_i^0 \times w_{i,1} \\ \cdots \end{cases} \quad (14)$$

Based on this network-division strategy, the weight blocks can be reused individually in FG and BR phases, as Fig. 4(b) illustrates. The four sub-networks are mapped into different computation units in hardware, with different local weights (e.g., the first sub-network includes $w_{1,1}$~$w_{p,r}$ while the second sub-network includes $w_{p+1,1}$~$w_{p+q,r}$). This characteristic inspires us to store these weight blocks locally, so that all the weights can be efficiently reused, avoiding frequent communication with off-chip memory. Besides, it can be concluded from Equation (2), (6) and (9) that in the FG phase, the weight matrix is $\boldsymbol{W}$ but in BR phase, it should be transposed to $\boldsymbol{W^T}$. This data character gives a challenge to hardware designers on how to efficiently transposed reuse the local weights without complex transpose operations.

2) Data Sparsity: Another important optimization point in hardware accelerators is data sparsity. Data sparsity, by definition, is the presence of many zero values in data. Take Equation (1) as an example, this aforementioned weighted sum process is composed of multiply-and-accumulate (MAC) operations. The weighted sum $z_i^{h^0}$ consists of n times of multiplies (i.e. $v_j^0 \times w_{i,j}$) and n times of accumulations. However, a fairly large number of these operations do not contribute to the final weighted sums because there are many zero neuron states in both visible and hidden layers.

Fig. 5 illustrates the data sparsity in the RBM model with an example. In this figure, v_1~v_2 and h_1~h_2 form sub-network #1, while v_3~v_4 and h_1~h_2 form sub-network #2. It can be found that the RBM weights in red line connections (i.e. $w_{2,1}, w_{2,2}, \ldots, w_{2,m}$) do not contribute to the weighted sums because $v_2 = 0$, in other words, they are invalid weights in MAC operations. The sub-network #1 only takes two valid MACs but the sub-network #2 should take four MACs. As a result, different sub-networks cost different computation time due to sparsity. How to efficiently take advantage of data sparsity to accelerate weighted sum processes, is another optimization point in hardware design.

III. Architecture-Circuit Co-Design of Proposed DBN Processor

A. Multi-core Architecture Design

Fig. 6 presents the proposed energy-efficient architecture with on-chip local learning to accelerate DBN/RBM unsupervised learning based on the data characters analyzed in Section III-B. This heterogeneous architecture consists of two processing engines, which are a multi-core partial sum engine (MPSE) and an accumulation & sampling engine (ASE). The MPSE is responsible for calculating the partial sums of different sub-network in the learning and inference process. The ASE is used for accumulating the partial sums calculated by MPSE and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JETCAS.2021.3114396, IEEE Journal on Emerging and Selected Topics in Circuits and Systems

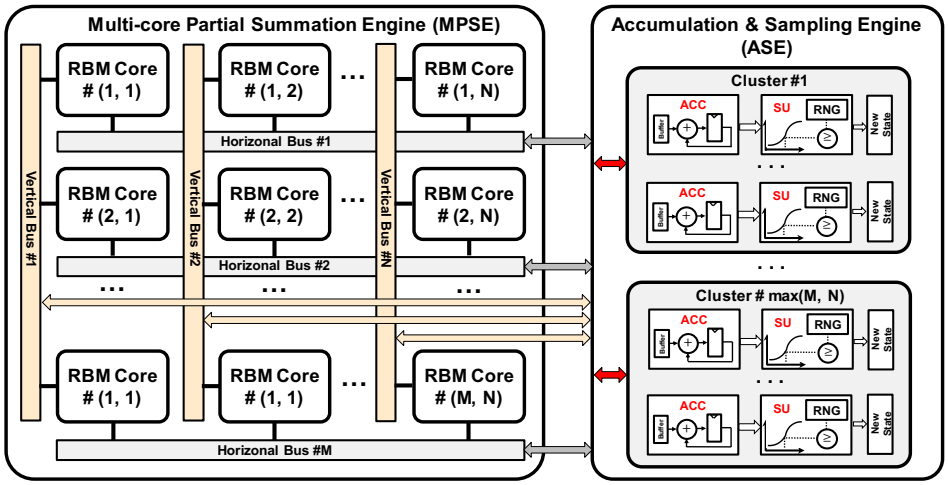


Fig. 6. The proposed heterogeneous multi-core architecture for unsupervised learning of RBM. This architecture consists of two processing engines, which are a multi-core partial sum engine (MPSE) and an accumulation & sampling engine (ASE).

sampling new neuron states. In MPSE, M×N RBM cores form a transposable multi-core architecture. Each RBM core calculates the partial sum of a sub-network, storing the corresponding weight block and neuron states locally. Based on the local weights, the local learning rule is also operated in each RBM cores respectively. Considering the characteristics of the transposed data reuse, we design a transposable architecture with horizontal bus and vertical bus, which are respectively responsible for the partial sum data movements in the FG and BR phases. RBM cores in the same row are connected to same horizontal bus (e.g. RBM Core #(1, 1), RBM Core #(1, 2), ... and RBM Core #(1, N) are connected to Horizontal Bus #1), while the RBM cores in the same column are connected to same vertical bus (e.g. RBM Core #(1, 1), RBM Core #(2, 1), ... and RBM Core #(M, 1) are connected to the Vertical Bus # 1).

When an RBM core completes partial sum calculation of the sub-network, the result will be sent to the ASE. Each cluster in the ASE operates the accumulation and sampling for one row or one column in the MPSE. Parallel accumulators (ACC) in the cluster receive the partial sums and get the total weighted sums. The partial sums transferred from the same row or column are accumulated in a cluster. After that, the sampling unit (SU) will operate Gibbs Sampling as Equation (3), (7) and (10) based on the calculated total weighted sums. SU is made up of a sigmoid function module, a random number generator (RNG) and a comparator. The new neuron states generated by the SU will be sent back to the RBM cores. Next, the neuron states stored in each RBM core will be refreshed by the new states and cache the old states for the weight update as Equation (11). The sigmoid function module in the ASE is implemented using piecewise linear approximation strategy based on look up tables (LUT). According to the previous work, piecewise linear approximation is one of the most cost-saving and accurate nonlinear function implementation strategies [17]. Please note that since the FG and BR phases are not performed at the same time, the two phases share the clusters in the ASE to reduce hardware overhead.

The timing diagram of the MPSE and ASE workloads are presented in Fig. 7. Due to the crucial data dependence of DBN on-chip learning, it is impossible to implement a pipeline between different input data. Take the image dataset as an example, the learning process of the next image cannot start before the current image finishes its learning process because the weights are updated in each iteration. As Fig. 7 illustrates, there are many idle states in one iteration, which causes PE utilization degradation in the processor. In the future, we will explore the Globally-Asynchronous-Locally-Synchronous (GALS) method in ASIC design to optimize the proposed processor [18]. Specifically, the local clocks of RBM cores can be designed to be independent, and the local gated clocks will be shut off to save energy when the cores are idle. This method can reduce the impact caused by unbalanced workloads of different cores.

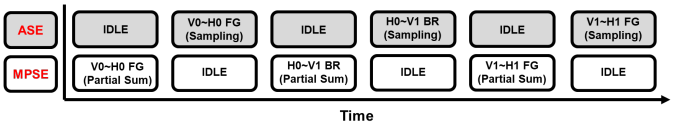


Fig. 7. Timing diagram of the MPSE and ASE workloads.

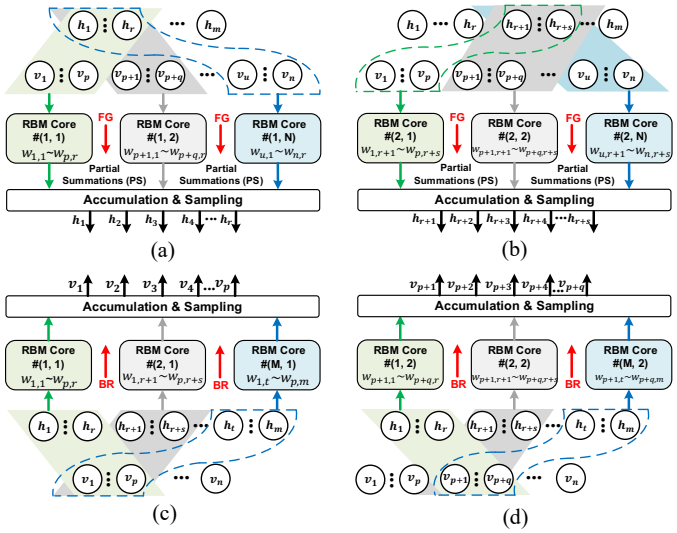


Fig. 8. Data mapping of the proposed multi-core DBN architecture.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JETCAS.2021.3114396, IEEE Journal on Emerging and Selected Topics in Circuits and Systems

Fig. 8 takes an example to illustrate the data mapping and movement in the proposed architecture. This RBM model scale is $m \times n$. First, this RBM is divided into M×N sub-networks and the weights are accordingly divided into M×N blocks and mapped into M×N RBM cores. Each RBM core calculates the partial sum of the sub-network in parallel. In Fig. 8 (a), all RBM cores in the first row calculate the partial sums of hidden neurons $h_1 \sim h_r$ respectively. Specifically, $w_{1,1} \sim w_{p,r}$ block is mapped to RBM Core #(1, 1), $w_{p+1,1} \sim w_{p+q,r}$ block is mapped to RBM Core #(1, 2), etc. Finally, all the partial sums calculated by the first row will be sent to cluster #1 in ASE for accumulation and sampling. It is worth mentioning that weighted sums $z_1^{h^0} \sim z_r^{h^0}$ and the new states of $h_1 \sim h_r$ will be parallel calculated due to the parallel ACCs and SUs in one ASE cluster, as Fig. 6 shows. Similarly, as shown in Fig. 8 (b), the RBM cores in the second row are used to calculate the partial sums of hidden neurons $h_{r+1} \sim h_{r+s}$. In the same way, the partial sums are sent to cluster #2 for accumulation and sampling, and so on to all rows. The same goes with the BR phase, as shown in Fig. 8 (c), the first column of RBM cores calculate partial sum of $v_1 \sim v_p$, and cluster #1 is responsible for accumulating and sampling the new state of these visible neurons. In Fig. 8 (d), the second column of RBM Cores are for partial sums of $v_{p+1} \sim v_{p+q}$, and so on to all columns. Since the FG and BR phases share the clusters, the number of clusters in the ASE is determined by the larger value of the row and column numbers.

The proposed architecture and data mapping method make full use of the data characters in RBM unsupervised learning. Since all RBM weights are stored and updated locally in the learning process, this architecture significantly reduces data transfer between off-chip memory and on-chip local memory. In addition, compared with the popular Network-on-chip (NoC) based DNN accelerators [19], this design constrains the data movements to specific local buses (horizontal bus and vertical bus). In theory, the complexity and energy consumption of data transmission are lower, which is very suitable for the edge computing scenario targeted by DBN. Moreover, this design proposes an efficient heterogeneous architecture to separate the parallel part (partial sum calculation) and serial part (accumulation and sampling) according to their respective characters, deploying them in different hardware engines.

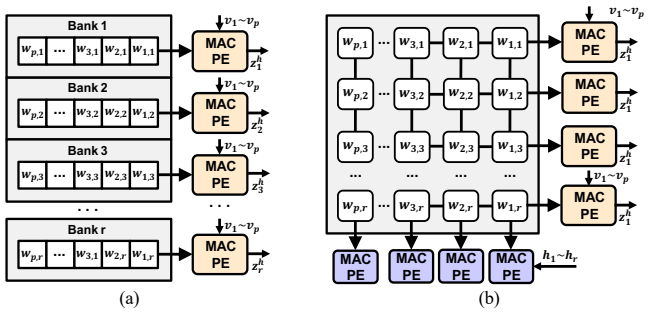


Fig. 9. (a) Conventional method for parallel MAC operations and (b) novel method based on transposed weight memory. Note that in Fig. 9 (b), the MAC PEs in different colors can actually be reused in hardware design, and it is for convenience explanation to display two sets of PEs in this figure.

B. Key Circuit-level Design

In this part, we will introduce the proposed high-efficient circuit-level design in each RBM core. According to the algorithm-level analysis in Section III, the circuit-level optimization should focus on two major challenges: How to design processing elements and local memory to support transposed reuse of local weights, and how to utilize neuron states sparsity to improve throughput and energy efficiency.

1) Parallel PE Array and Transposed Weight Memory: In the circuit-level optimization, our design aims to efficiently calculate the partial sum of each sub-network, i.e. to accelerate the MAC operations. Generally, methods for accelerating MAC operations can be divided into three categories: Parallel processing element (PE) array [20], systolic array [21] and bit-serial processing [22]. Although systolic arrays and bit-serial processing have been proven effective in general DNN accelerators, they do not take full advantage of data sparsity, which is exactly an important optimization point in the DBN model. Therefore, in the design of RBM core, the parallel PE array strategy is selected for acceleration.

Fig. 9 (a) shows the conventional parallel PE array method to accelerate MAC operations. The key point of this design is to expand the number of memory banks, making PE and memory bank correspond to each other to achieve the effect of parallel reading and operation [23]. Assuming that $v_1 \sim v_p$, $h_1 \sim h_r$ and their weights are mapped into this RBM core as a sub-network. The calculation flow of FG phase is as Fig. 9 (a) presents. For the first PE and Bank 1 of memory, $w_{1,1}$, $w_{2,1}$, ..., $w_{p,1}$ are read out in order to join MAC operations with v_1, v_2, ..., v_p respectively, then the result is for z_1^h (i.e. partial sum of hidden neuron h_1). The same goes for all the PEs, and each PE is responsible for the partial sum of a hidden neuron. This parallel MAC strategy significantly increases the calculation speed. Moreover, based on this strategy, energy-efficient calculation is possible for data sparsity optimization because MAC PE can skip reading the invalid weights when corresponding neuron states are 0.

However, when it comes to BR phase, this strategy faces challenge due to transposed reuse of RBM weights, as Fig. 4 shows. Obviously, the memory architecture in Fig. 9 (a) does not support parallel reading and computation in column dimension due to conventional SRAM architecture (e.g. $w_{1,1}$, $w_{1,2}$, ..., $w_{1,r}$ cannot be read out in serial to join MAC operations). J. Su *et al.* proposed a multi-mode MAC PE to solve this problem based on conventional SRAM in Fig. 9 (a) [13]. Nevertheless, this solution abandoned advantages of parallel read and parallel MAC operations in BR phase. In essence, the circuit-level optimization to solve this problem is to design a memory structure that supports transposable and parallel read-out, as Fig. 9 (b) shows.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JETCAS.2021.3114396, IEEE Journal on Emerging and Selected Topics in Circuits and Systems

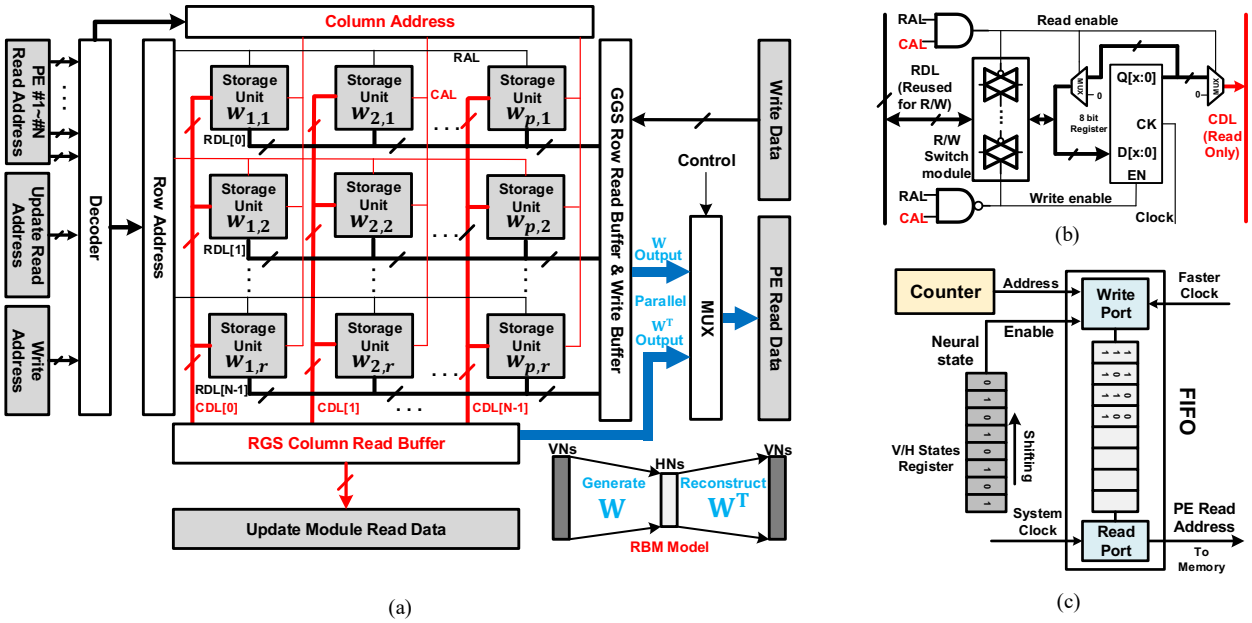


Fig. 10. (a) Proposed transposable weight memory, (b) storage unit of the memory and (c) sparse address generator for transposable memory.

In this work, a transposable memory structure based on registers is proposed. Fig. 10 (a) shows the details of the proposed memory design. In this transposable memory, the data lines and address lines are divided into row data lines (RDL), column data lines (CDL), row address lines (RAL) and column address lines (CAL). Compared with the traditional SRAM architecture, the transposable memory adds a set of row data lines and address lines to achieve the effect of transposable parallel reading. Since the FG and BR phases are not processed at the same time, the rows and columns of this memory share the same set of address decoders to drive the reading by row or column, and the output can be selected according to the current state controlled by outside signal. In addition, when writing weight data, it also supports multi-port writing by row or column based on this structure. The storage units can be implemented as 7T/8T SRAM units or be simply implemented as registers. Fig. 10 (b) presents the storage unit design of proposed transposable memory, where the x-bit local weight is stored in the register, and bus reuse for Read/Write operations is enabled by transmission gates. Note that the registers-based design is a naïve version of the proposed transposable memory for FPGA evaluation because registers-based design is feasible for FPGA devices. We will focus on 7T/8T SRAM-based ASIC design [24] in our future work to further prove the energy/area efficiency of the proposed transposable memory for the DBN processor design.

2) Sparse Address Generator: In addition to data transposed reuse, data sparsity is another important character in DBN/RBM unsupervised learning according to algorithm-level analysis. In the current academia, the key point for optimizing the data sparsity is skip reading strategy. This strategy does not only skip invalid MAC operations, but also skips the reading of invalid weights, thereby improving calculation speed and reducing the energy consumption of weight reading. However, in recent years, the implementations of skip reading are mostly based on predefined address encoding [25]-[26] or complex controller [27]. Combining with the transposable memory architecture, this work proposes a skip reading strategy based on pre-generated sparse addresses. Fig. 10 (c) presents the sparse address generator (SAG) and its working principle. Before each RBM Core starts MAC operation, SAG shifts the visible or hidden neuron state register, meanwhile the counter starts to accumulate. If the least significant bit (LSB) of register is 1, the counter value will be written into the FIFO as the address for transposable memory t. If the LSB is 0, the corresponding address will not be generated. After the pre-generated address phase is over, the PEs will serially fetch the address from the FIFO, fetch weights from the transposable memory, and join the MAC operations. Finally, only those addresses of valid weights are generated and stored in FIFO. For instance, if v_2 is 0 in FG phase, the SAG will not generate address "2", resulting in $w_{2,1}$, $w_{2,2}$, $w_{2,3}$, ..., $w_{2,r}$ will not be read out for MAC operations. This skip reading strategy reduces computing time and memory access energy consumption. It is worth mentioning that because the pre-generated address circuit has low hardware overhead, it works at a higher clock frequency ("Faster Clock" in Fig. 10 (c)), so that the additional generation process does not increase the total computation time.

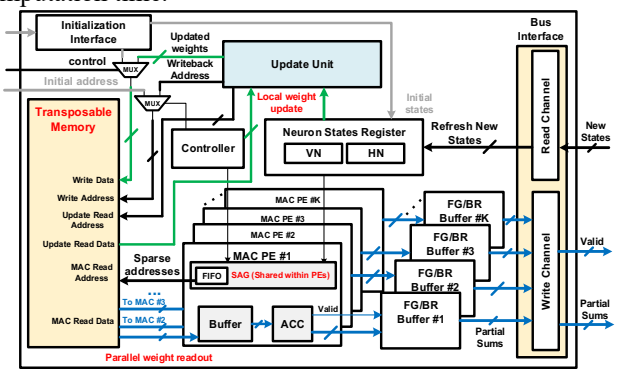


Fig. 11. Architecture and data path of the proposed RBM core with *K* parallel MAC PEs.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JETCAS.2021.3114396, IEEE Journal on Emerging and Selected Topics in Circuits and Systems

C. RBM Core Design and Asynchronous Data Flow

Fig. 11 presents the RBM core design based on the aforementioned transposable memory, parallel PEs and sparse address generator. In the initialization phase, the local weights (in the transposable memory) and neuron states (in the neuron states registers) are initialized from the outside interface. According to these optimizations, RBM cores can efficiently complete a partial sum of sub-network with local weights and neuron states. In FG and BR phase, K parallel PEs are responsible for MAC operations based on sparse addresses generated by SAG. Assuming the scale of sub-network in this core is $p \times r$, K should be equal to the larger value of p and r due to the parallel PE strategy. The partial sums are stored in buffers temporally and will be sent to horizontal or vertical bus with the valid signal set to 1, informing the bus that partial sum of this RBM core has been calculated. Note that within each RBM core, the data flow is synchronous and sequential, by computing the first FG, followed by the BG, and finally the second FG.

After the 2 FG phases and 1 BR phase, the update unit in each RBM core will start local weight learning. Fig. 12 illustrates the weight updating process as Equation (11). The multiplication of the learning rate is approximately computed by shifter operations. Since the proposed transposable memory supports multi-port parallel read/write, the update unit is designed as row-by-row processing. For instance, the first row of weights ($w'_{1,1}$, $w'_{2,1}$, ..., $w'_{p,1}$) and corresponding neuron states (v_1, v_2, ..., v_p and h_1) are fetched into the update unit in cycle 1. Then the calculation results $w_{1,1}$, $w_{2,1}$, ..., $w_{p,1}$ will be written back to the first row of transposable memory in cycle 2, meanwhile the second row of weights are processed similarly. When the r-th row of weights are updated (after r + 1 cycles), it means that this learning iteration is completed, and the learning process can be started.

The asynchronous data flow exists because different RBM cores produce the partial sums with different clock cycles, thereafter the ASE is in charge to accumulate these partial sums from the different RBM cores. To address the synchronization issue, the ASE is in charge to differentiate the data arrivals (from the different RBM cores) by using a bus arbiter and a valid-ready handshake protocol. Fig. 13 takes the first row of RBM cores and cluster 1 in AES as an example to illustrate the asynchronous data flow. Since the neuron states mapped in each RBM core are variant, the number of addresses generated by the SAG is thereby different, resulting in some RBM cores with fewer sparse addresses calculating faster and needing fewer clock cycles. This phenomenon will lead some RBM cores sending the partial sums result to the bus earlier than others, like RBM Core (1, 1) in this figure. Fig. 14 presents the design of bus arbiter and handshake protocol. The handshake protocol used in this design is a valid-ready protocol, and the partial sums will be transferred to the ASE if and only if the two signals are both high (Fig. 14 (a)). Besides, in order to avoid bus conflicts, this design presents a fixed priority. When more than one RBM cores complete partial sum calculations at the same time, partial sum of the RBM core with a higher priority is selected by the arbiter and transferred to ASE first (Fig. 14 (b)).

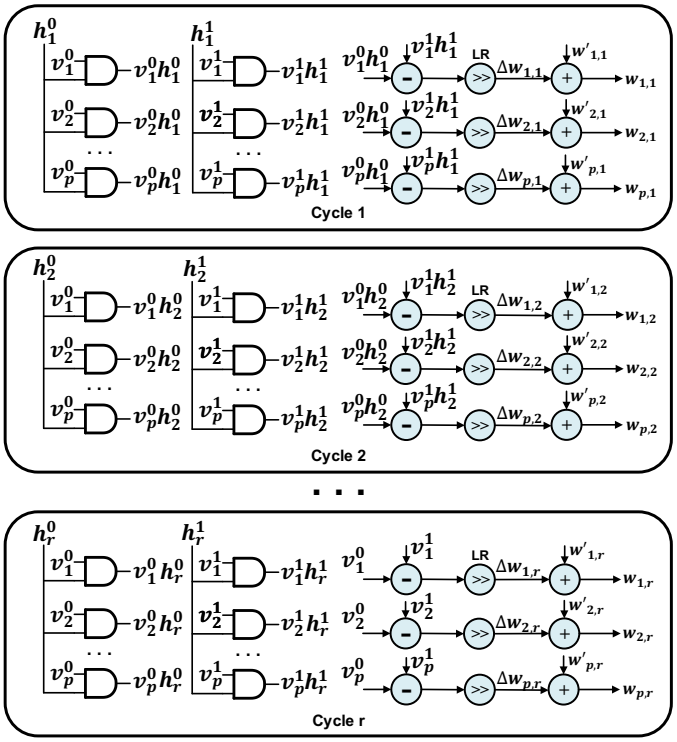


Fig. 12. Update unit design and dataflow.

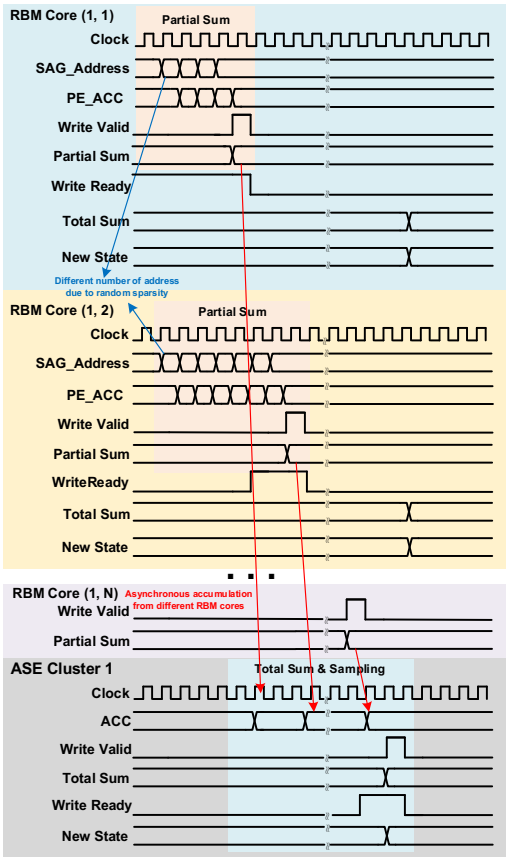
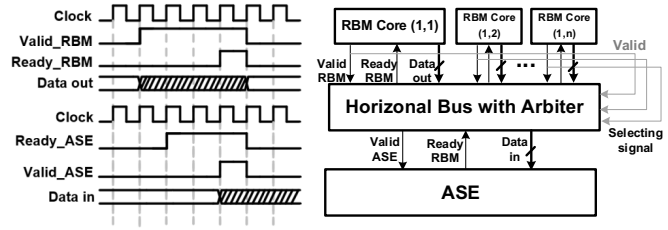


Fig. 13. Asynchronous data movement of different RBM cores in the first row in MPSE.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JETCAS.2021.3114396, IEEE Journal on Emerging and Selected Topics in Circuits and Systems

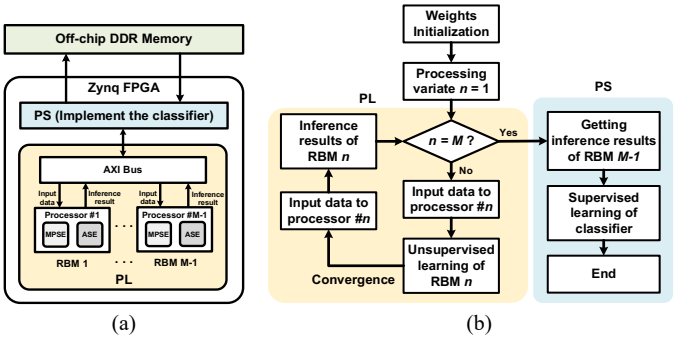


(a) (b)

Fig. 14. Asynchronous data movement of different RBM cores in the first row in MPSE.

TABLE I
EVALUATION OF CLASSIFICATION PERFORMANCE BASED ON DIFFERENT BIT WIDTH OF DBN WEIGHTS

Decimal digit of DBN weights	Classification accuracy
6-bit	9.4%
7-bit	66.8%
8-bit	91.1%
9-bit	91.1%
10-bit	91.3%
11-bit	91.6%
12-bit	91.5%



(a) (b)

Fig. 15. (a) DBN processor design on Zynq FPGA and the (b) processing flow. Assuming there are M-1 processor modules on PL of FPGA, their unsupervised learning tasks are processed in order. The input training data of RBM *n* is from the inference results of RBM *n*-1, as illustrated in Section III. After all RBMs finish unsupervised learning, the inference results of RBM M-1 will be used for supervised learning of the classifier on PS.

IV. HARDWARE EVALUATION AND DISCUSSION

A. System Implementation

To thoroughly evaluate the proposed DBN processor, this study selects a Xilinx's Zynq FPGA device (xc7z100ffg900-2) to perform evaluation of the hardware utilization, throughput and energy efficiency considering the flexibility and reconfigurability of FPGA.

1) Algorithm-hardware Co-simulation: Before hardware system design, algorithm-hardware co-simulation can help hardware designers determine the best hardware parameters based on the algorithm model, such as data bit width, number of training iterations, etc. [28]. In this study, we aim to introduce hardware errors such as quantization, non-linear function error, to find the least data bit width and save hardware utilization. An algorithm-hardware co-simulation platform of DBN is developed with Python, and MNIST dataset is used for evaluation. We select a DBN model with 784 input neurons (equal to pixel numbers of MNIST dataset), 200 and 100 hidden neurons and 10 output neurons for evaluation (784-200-100-10), which includes a 784×200 RBM, a 200×100 RBM and a 100×10 classifier. Table I shows the classification performance based on different bit width (decimal digit) of DBN weights. According to the evaluation result, the accuracies based on decimal digits over 8 are all around 91.1%. Thus, 8-bit decimal digits have been selected for weights format for our hardware design.

2) System Design on FPGA: Fig. 15 presents the DBN processor design and the processing flow. The programmable logic (PL) part in the Zynq FPGA deploys *M*-1 processor modules, and each module is responsible for the unsupervised learning of a certain RBM in the DBN. The weights are stored in the local transposable memory, and local memory only transfers with the off-chip DDR memory during the initialization phase. The input dataset and all RBM inference results are stored in off-chip DDR memory. Note that since this study is to optimize and discuss the unsupervised on-chip learning of DBN, the supervised learning of the last two layers (classifier) is not implemented on PL part of FPGA with backpropagation (BP) algorithm [29]. This supervised learning process is implemented on the processing system (PS) part of the Zynq FPGA and is completed by the embedded processor Cortex-A9 on PS.

From the algorithm-hardware co-simulation results, the proposed DBN processor is comparable to previous unsupervised DBN design on MNIST classification performance in [5] with 93.4% accuracy. In addition, the system architecture shown in Fig. 15 (a) makes full use of the local memory, greatly reducing data communication with off-chip memory. The next part will explain that in order to further illustrate high energy efficiency and acceleration performance of this design, we have selected a DBN scale and performed hardware evaluation based on the available hardware resources of the FPGA.

B. Evaluation and Discussion

1) Hardware Overhead: Considering the hardware resource limitation of Zynq-7100 FPGA device, different DBN scales have been examined and a 4-layer DBN model with a layer of 100 input neurons, two layers of 60 and 30 hidden neurons, and a layer of 10 output neurons (100-60-30-10) is finally selected to be implemented on FPGA at the system architecture as described in Fig. 15 (a). Thus, in this DBN processor, there are two processor modules for the two RBMs (100×60 and 60×30) in the selected DBN. The two RBMs are both mapped into the architecture in Fig. 6 with different M and N values. We aim to discuss hardware overhead of different mapping strategies. For instance, 100×60 RBM model can be mapped into 6×10 RBM cores (M=6, N=10) with 10×10 sub-network in each core, or 3 ×5 RBM cores (M=3, N=5) with 20×20 sub-networks, etc.

Table II shows the hardware utilization of the selected DBN implementation with different mapping. We also compare this study with other typical FPGA-based DBN/RBM processors. Compared with the other two works, because the transposable memory cannot be synthesized as Block RAM (BRAM) on FPGA, which causes a large overhead of these two resources.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JETCAS.2021.3114396, IEEE Journal on Emerging and Selected Topics in Circuits and Systems

TABLE II
HARDWARE UTILIZATION FOR DBN PROCESSORS IMPLEMENTATIONS IN XILINX'S ZYNQ FPGA DEVICE

Implementations (divided into RBMs)			Slice LUTs	Slice Registers	DSPs	BRAMs
This study	100×60 RBM	M=6, N=10	102482	101760	0	0
		M=3, N=10	82165	91256	0	0
		M=3, N=5	71584	86448	0	0
	60×30 RBM	M=N=10	31245	32807	0	0
		M=2, N=3	10587	12043	0	0
B. Ahn *et al.* [12]	784×200 RBM		84870	84871	N/A	N/A
GeCo [14]	1760×576 RBM		135359	115455	N/A	240

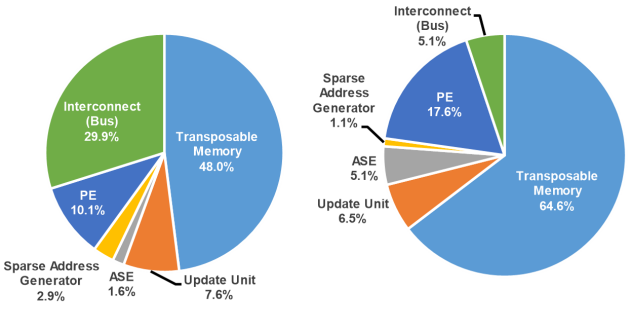


Fig. 16. Hardware overhead divided into different components for (a) LUTs and (b) registers based on 100×60 RBM and M=6, N=10 mapping. According to Table II, the total LUTs and registers consumption are 102482 and 101760, respectively.

For further discussion about hardware overhead, we divided the hardware utilization into different components based on 100×60 RBM and M=6, N=10 mapping, as Fig. 16 shows. The transposable memory occupies most of the hardware overhead, with 48.0% in LUTs and 64.6% in registers. But in previous FPGA works [11]-[14], the memory implementations are all based on BRAM. Thus, although our design has a larger overhead on LUT and register, it is still acceptable, and the comparison of other evaluations are fair enough.

In terms of this study, different mapping configurations actually result in different hardware overhead. It can be concluded from this table that with the number of RBM cores increases, hardware overhead grows up. According to Fig. 16, the most costly components in DBN processor design are transposable memory, PE and interconnect (bus) in the MPSE. Therefore, we develop a mathematical model to approximately evaluate the hardware overhead of these three components. Assume that the model is $x \times y$ and the MPSE consists of $a \times b$ RBM cores (i.e. M=a and N=b), which indicates each RBM core is responsible for a sub-network with $x/a \times y/b$ scale. Considering the transposable memory, because the total number of storage units is fixed ($x \times y$ weights in this RBM), the main factor to influence hardware overhead is Read/Write ports of memory. Thus, we use the number of R/W ports to evaluate the overhead caused by transposable memory:

$$Overhead_{memory} = a \times b \times (\frac{x}{a} + \frac{y}{b}) = xb + ya \quad (15)$$

For the overhead of PEs, we also evaluate it as the number of PEs because each PE theoretically costs the same LUTs and registers. The number of PEs in each core is the larger value of x/a and y/b according to Fig. 10. Thus, the overhead can be presented as:

$$Overhead_{PE} = a \times b \times max(\frac{x}{a}, \frac{y}{b}) \quad (16)$$

As for interconnections overhead, we divided it to connections from PEs to the horizontal or vertical bus (equal to the number of PEs), and the connections outside the RBM cores ($a + b$). So, this part can be written as:

$$Overhead_{connection} = a \times b \times max(\frac{x}{a}, \frac{y}{b}) + a + b \quad (17)$$

From Equation (15)-(17), because x and y are both fixed in the RBM model, the number of RBM cores increasing will naturally lead to growth of hardware overhead. Therefore, for more hardware-saving design, such as design area is limited for the DBN processor, less RBM cores can greatly help reduce the hardware overhead.

2) Acceleration Performance: Because the MAC operations of an RBM are processed in parallel, and RBM weights are stored and updated locally, the proposed DBN processor can achieve high acceleration performance compared with conventional CPU and other previous accelerators/processors. Limited by hardware resources, the 100-60-30-10 DBN model is chosen for acceleration. Accordingly, the MNIST training images are down sampled to 10×10 pixels. In addition, we set the system clock as 100 MHz and the "faster clock" in sparse address generator circuits as 200 MHz in simulation. For more general results, all the 60,000 down sampled MNIST images join the unsupervised learning based on our design, and the average time of one image is used for evaluation. Because the images are not same with previous works due to down sampling, we also simulate throughput by GNWPS (G neural weights per second) for a fair comparison with previous works.

Table III presents the comparison of acceleration performance based on different DBN/RBM accelerators. The 5.53 GNWPS throughput of our design achieves ×276.5 and × 2.25 speedup compared with CPU and the state-of-art FPGA-based design, while the utilization of data sparsity and skip reading strategy greatly help our work enhance throughput with only 100 MHz system clock. Besides, although the previous work [5] achieves higher performance, it is based on a higher system clock (210 MHz) and more precise technology (ASIC). In the whole processor system, only a very small circuit is working at 200 MHz (SAG) to accelerate the address pre-generation phase. As Fig. 14 displays, this circuit only costs 2.9% LUTs and 1.1% registers of hardware overhead.

Moreover, limited by FPGA hardware resources, this work just implemented and evaluated a fairly small DBN model (100-60-30-10), which also limits the throughput performance because of the highly parallel computing in the design. If the proposed design is implemented in ASIC design with enough

area, the advantage of throughput will be expanded to the extreme. Fig. 17 presents the maximum and minimum throughput (GNWPS) based on different numbers of input neurons. In this estimate, the number of hidden neurons is fixed to 100, and the RBM core configuration is 10×10 (M=N=10). It can be concluded that based on the proposed architecture, the advantages aforementioned can make the throughput have the potential to maximum 49.44 GNWPS, which outperforms the state-of-art FPGA-based design almost 20 times.

TABLE III
ACCELERATION PERFORMANCE COMPARISON OF ON-CHIP LEARNING BASED ON DIFFERENT DESIGNS

	System Clock	Time per image	Throughput (GNWPS)	Speed up
CPU (Core i7-7500)	2.7 GHz	328 μs	0.02	×1
This study (M=6, N=10 in RBM 1; M=N=10 in RBM2)	100 MHz	1.41 μs	5.53	×276.5
This study (M=3, N=5 in RBM 1; M=2, N=3 in RBM2)	100 MHz	1.83 μs	4.26	×213.0
C. Tsai *et al.* [5] (ASIC)	210 MHz	N/A	7.53	×376.5
B. Ahn [12] (FPGA)	200 MHz	N/A	1.90	×95.0
J. Su *et al.* [13] (FPGA)	200 MHz	N/A	2.46[#]	×195.5
GeCo [14] (FPGA)	50 MHz	618.13 μs[*]	0.88	×44.0

*Note that because the training datasets for on-chip learning are different in these works, it is unfair to compare processing time per image. [#]It is converted from original GMULPS parameter.

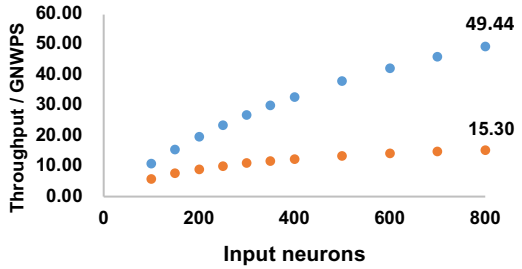


Fig. 17. For processing a single RBM, the maximum (blue dots) and minimum (origin dots) throughput (GNWPS) of the proposed DBN processor, under different numbers of input neurons. The number of hidden neurons is fixed to 100. Note that due to the data sparsity, precise throughput of unsupervised learning process cannot be obtained, and only theoretical maximum and minimum values can be given in this figure.

TABLE IV
ENERGY CONSUMPTIONS OF DIFFERENT DBN PROCESSORS/ACCELERATORS

-	RBM size	Energy per NW
This study in FPGA	100-60-30	45.0 pJ
C. Tsai *et al.* [5] in 65-nm ASIC	4096-4096	41.31 pJ (Energy by external memory access is not included)

Note that the simulation conditions are similar to Table III, with a 100 MHz system clock and a 200 MHz faster clock for the sparse address generator circuit.

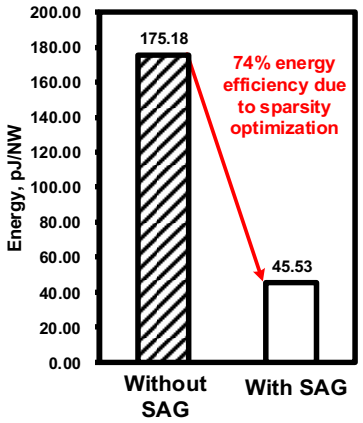


Fig. 18. Energy efficiency improvement in the proposed DBN design.

Table III also reveals that for a fixed DBN/RBM scale, the configurations with more RBM cores can help the processor achieve higher throughput, as the second and third rows of Table III illustrate. This phenomenon is easy to understand, because when the number of RBM cores decreases, the size of the sub-network mapped in each core will increase accordingly, which in turn increases the time of the address pre-generation and MAC operations. Combined with the discussion before, we can find an interesting trade-off between hardware overhead and throughput. The DBN processor with fewer RBM cores can reduce the overhead, but also reduce the working throughput. The flexibility of the FPGA can help hardware designers to measure this trade-off and design the hardware parameters of DBN processor to suit the application.

3) Energy Efficiency: Table IV presents the energy consumption simulation results on FPGA evaluation kit and comparison with other works. The implemented DBN scale is 100-60-30-10. Energy consumption per neural weight (pJ/NW) and per image is used for the energy efficiency evaluation during on-chip learning. The results in this table do not include energy consumption of off-chip DDR memory accesses, as weights are stored and updated locally in the proposed DBN processor. Besides, the energy for retrieving input data from DDR memory in the initialization phase has been neglected. Note that the total energy has been divided into on-chip learning of each RBM in the simulation process. Thanks to the data sparsity optimization of the proposed processor, the energy efficiency is still comparable with the state-of-art ASIC design, though this study is only evaluated on FPGA.

Fig. 18 presents the energy efficiency achieved by the sparsity optimization. We selected the RBM 1 with 100×60 scale, deleted the sparse address generator (SAG) circuit in all RBM cores and evaluated energy consumption per NW. Without the SAG, data flow of the processor is similar to GeCo [14], in which there is no skipping reading, regardless of whether the weights are valid in the MAC operations, they will all be read out to participate in the accumulation. From the comparison result in Fig. 16, the sparsity optimization helps the DBN processor to enhance energy efficiency by 74%.

In summary, the above evaluations of hardware implementation confirm that the proposed DBN processor is efficient in on-chip learning process with high throughput and

energy efficiency. In addition, we find a trade-off in terms of hardware overhead and throughput. This feature can guide designers to reconfigure DBN processors suitable for specific applications.

V. Conclusion and Future Work

This paper presents an energy-efficient DBN processor based on heterogeneous multi-core architecture with transposable weight memory and on-chip local learning. An algorithm-architecture-circuit co-design method is used for comprehensive analysis and hardware design. In algorithm level, weight reuse and data sparsity are found as two important characteristics for efficient acceleration. In architecture level, a heterogeneous multi-core architecture with on-chip local learning is proposed based on algorithm-level analysis. This design distributes the weighted sum operations to multiple processor cores and divides two heterogeneous architectures to complete respective different tasks in DBN processing. In order to improve the efficiency of the architecture, this study proposes a transposable memory design and a pre-generated address circuit at the circuit level, so that each core can utilize the weight reuse and data sparsity for efficient operations. The DBN processor is implemented and thoroughly evaluated on Xilinx Zynq FPGA. With the help of algorithm-architecture-circuit co-design, the proposed DBN processor design achieves ×150.5 and ×1.22 speedup compared with CPU and the state-of-art FPGA-based design, and the data sparsity optimization helps the DBN processor enhance 75.4% energy efficiency. Though the proposed processor is energy-efficient and has high throughput, it still has several bottlenecks. In detail, the transposable memory is based on registers, not an area-efficient SRAM-based design. Besides, the calculations in each core are not balanced, which causes utilization degradation of these cores. In the future, we will focus on ASIC implementation of the proposed DBN processor in a GALS architecture with a 7T/8T SRAM-based transposable memory design to solve the bottlenecks and further improve throughput and energy efficiency.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521. pp. 436–444, May 2015.

[2] C. Szegedy, A. Toshev, and D. Erhan, “Deep neural networks for object detection,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 2553–2561.

[3] G. E. Dahl, D. Yu, L. Deng and A. Acero, "Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 30-42, Jan. 2012.

[4] J. Lee, S. Kang, J. Lee, D. Shin, D. Han and H. -J. Yoo, "The Hardware and Algorithm Co-Design for Energy-Efficient DNN Processor on Edge/Mobile Devices," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 10, pp. 3458-3470, Oct. 2020.

[5] C. Tsai, W. Yu, W. H. Wong and C. Lee, "A 41.3/26.7 pJ per Neuron Weight RBM Processor Supporting On-Chip Learning/Inference for IoT Applications," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 10, pp. 2601-2612, Oct. 2017.

[6] J. Lee, J. Lee, D. Han, J. Lee, G. Park, and H.-J. Yoo, “7.7 LNPU: A 25.3TFLOPS/W sparse deep-neural-network learning processor with fine-grained mixed precision of FP8-FP16,” in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2019, pp. 142–144.

[7] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, “Efficient processing of deep neural networks: A tutorial and survey,” *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[8] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, 2006.

[9] S. Park, K. Bong, D. Shin, J. Lee, S. Choi, and H.-J. Yoo, “A 1.93 TOPS/W scalable deep learning/inference processor with tetra-parallel MIMD architecture for big-data applications,” in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2015, pp. 80–81.

[10] Y. Liu, Y. Wang, F. Lombardi and J. Han, "An Energy-Efficient Online-Learning Stochastic Computational Deep Belief Network," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 3, pp. 454-465, Sept. 2018.

[11] A. W. Savich and M. Moussa, “Resource efficient arithmetic effects on RBM neural network solution quality using MNIST,” in *Proc. Int. Conf. Reconfigurable Comput. FPGAs*, Nov./Dec. 2011, pp. 35–40.

[12] B. Ahn, “Computation of deep belief networks using special-purpose hardware architecture,” in *Proc. IEEE Int. Conf. Neural Netw.*, Beijing, China, Jul. 2014, pp. 141–148.

[13] J. Su, D. B. Thomas and P. Y. K. Cheung, "Increasing Network Size and Training Throughput of FPGA Restricted Boltzmann Machines Using Dropout," *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM*), 2016.

[14] W. Yi, J. Park and J. Kim, "Geco: Classification restricted Boltzmann machine hardware for on-chip learning," *Proc. Int. Symp. Rapid Syst. Prototyping (RSP)*, Oct. 2017, pp. 30–35.

[15] J. Wu *et al.*, "An Energy-efficient Multi-core Restricted Boltzmann Machine Processor with On-chip Bio-plausible Learning and Reconfigurable Sparsity," *2020 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, 2020.

[16] W. Byrne, "Alternating minimization and Boltzmann machine learning," *IEEE Transactions on Neural Networks*, vol. 3, no. 4, pp. 612-620, July 1992.

[17] I. Tsmots, O. Skorokhoda and V. Rabyk, "Hardware Implementation of Sigmoid Activation Functions using FPGA," in *IEEE International Conference on the Experience of Designing and Application of CAD Systems (CADSM)*, 2019, pp. 34-38.

[18] K. Chong, K. Chang, B. Gwee and J. S. Chang, "Synchronous-Logic and Globally-Asynchronous-Locally-Synchronous (GALS) Acoustic Digital Signal Processors," in *IEEE Journal of Solid-State Circuits*, vol. 47, no. 3, pp. 769-780, March 2012.

[19] G. K. Chen *et al.*, “A 4096-neuron 1M-synapse 3.8-pJ/SOP spiking neural network with on-chip STDP learning and sparse weights in 10-nm FinFET CMOS,” *IEEE Journal of Solid-State Circuit*, vol. 54, no. 4, pp. 992-1002, Apr. 2019.

[20] S. Kang *et al.*, "7.4 GANPU: A 135TFLOPS/W Multi-DNN Training Processor for GANs with Speculative Dual-Sparsity Exploitation," *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2020, pp. 140-142.

[21] Norman P. Jouppi *et al.*, “In-Datacenter Performance Analysis of a Tensor Processing Unit,” in *International Symposium on Computer Architecture (ISCA)*, 2017.

[22] H. Sharma *et al*, “Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Networks,” in *International Symposium on Computer Architecture (ISCA)*, 2018.

[23] Z. Xie, Y. Lu, Y. Fan and X. Zeng, "Data mapping scheme and implementation for high-throughput DCT/IDCT transpose memory," *2014 12th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, 2014, pp. 1-3.

[24] K. Bong, S. Choi, C. Kim, D. Han and H. Yoo, "A Low-Power Convolutional Neural Network Face Recognition Processor and a CIS Integrated With Always-on Face Detector," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 1, pp. 115-123, Jan. 2018.

[25] P. Das and H. K. Kapoor, “nZESPA: A Near-3D-Memory Zero Skipping Parallel Accelerator for CNNs,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[26] A. Ardakani, C. Condo and W. J. Gross, "Fast and Efficient Convolutional Accelerator for Edge Computing," *IEEE Transactions on Computers*, vol. 69, no. 1, pp. 138-152, 1 Jan. 2020.

[27] M. Mahmoud *et al.*, "TensorDash: Exploiting Sparsity to Accelerate Deep Neural Network Training," *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 781-795.

[28] Jiajun Wu *et al.*, "Efficient Design of Spiking Neural Network with STDP Learning Based on Fast CORDIC," *IEEE Transactions on Circuits and Systems I: Regular Papers*, doi: 10.1109/TCSI.2021.3061766.

[29] S. S. Haykin, *Neural Networks and Learning Machines*, vol. 3. Upper Saddle River, NJ, USA: Pearson, 2009.

Jiajun Wu (Member, IEEE) was born in Fujian, China in 1999. He received his B.Eng. in the School of Optical and Electronic Information, Huazhong University of Science and Technology (HUST), Wuhan, China.

His research interests include energy-efficient VLSI architecture design and ultra-low-voltage circuit design for machine learning, neuromorphic computing and robot applications.

Xuan Huang (Member, IEEE) was born in Fujian, China in 2000. He is an undergraduate student of the School of Optical and Electronic Information, Huazhong University of Science and Technology (HUST), Wuhan, China.

His research interest includes ultra-low-voltage circuit design for machine learning and neuromorphic computing and robot applications.

Le Yang (Member, IEEE) was born in Henan, China in 2000. He is an undergraduate student of the School of Optical and Electronic Information, Huazhong University of Science and Technology (HUST), Wuhan, China. His current research interest includes spiking neural network processor design.

Jipeng Wang (Member, IEEE) received the B.S. degree in integrated circuits engineering from Huazhong University of Science and Technology (HUST), Wuhan, China in 2020 and now he is currently pursuing the Ph.D degree with the School of Optical and Electronic Information.

His research interests include RF transceiver system design, hardware design for AI-based bio-sensing systems, computing in memory and AI algorithms for object detection.

Bingqiang Liu (Member, IEEE) received the B.S. degree in integrated circuits engineering from Huazhong University of Science and Technology (HUST), Wuhan, China in 2020 and now he is currently pursuing the Ph.D degree with the School of Optical and Electronic Information. He is currently the chair of IEEE CASS-EDS-SSCS HUST Student Branch Chapter.

His research interests include energy-efficient VLSI architecture design and ultra-low-voltage circuit design for artificial intelligence, neuromorphic computing, healthcare and robot applications.

Ziyuan Wen (Member, IEEE) was born in Shandong, China in 2000. He is an undergraduate student of the School of Optical and Electronic Information, Huazhong University of Science and Technology (HUST), Wuhan, China.

His research interest includes hardware accelerator design for artificial intelligence and deep learning algorithms, integrated circuits and system design for biomedical and intelligent robot applications and image and video signal processing.

Juhui Li received his B.Eng in Computer Communication, Hunan University, Changsha, China, and his M. Eng in Computer Application from the School of Computer Science, Northwestern Polytechnical University, Xi'an, China, in 1994 and 2000 respectively, and the Ph.D degree, in Electronics Engineering, from the Nanyang Technological University (NTU), Singapore in 2005.

He is currently working as a Senior Technical Manager with Nations Innovation Pte Ltd Singapore. Dr Li has many years' experience in low power mixed-signal ASIC designs. His research interests include IoT, AIoT applications, ultra-low power MCU design, near-threshold SoC design, and ultra-low power mixed-signal ASIC design.

Guoyi Yu (Member, IEEE) received his B.Eng., M. Eng. and Ph.D degrees, both in Electronics Engineering, from the Huazhong University of Science and Technology (HUST), Wuhan, China in 2000, 2003 and 2006, respectively.

Since 2007, he had been with the Department of Electronic Science and Technology, HUST, as a Lecturer. Currently, he is an Associate Professor with the School of Optical and Electronic Information, HUST. His research includes analog-mixed signal circuit design, sensor interface circuit design, and heterogeneous 3D-IC design for biomedical/healthcare, wireless sensor, wearable and robot applications. Dr. Yu is an IEEE Member and currently the Treasurer of IEEE CASS-EDS-SSCS Wuhan Joint Chapter. He is an active reviewer of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS, IEEE TRANSACTIONS ON BIOMEDICAL CIRCUITS AND SYSTEMS, IEEE ACCESS journal, and *Microelectronics Journal* (Elsevier).

Kweng-Siong Chong (Senior Member, IEEE) received his B.Eng., M.Phil. and Ph.D degrees in electrical and electronic engineering from Nanyang Technological University (NTU), Singapore, in 2001, 2002, and 2007 respectively.

He is currently a Senior Research Scientist with the Temasek Laboratories @ NTU, Singapore. He is/was principal investigator (PI), co-PI, and collaborator of several research projects, including the projects supported from National Science Foundation (Singapore), Defense Advanced Research Projects Agency (USA), Ministry of Education (Singapore), and Public Sector Research Funding (Singapore). He co-founded two start-ups. His research interests include hardware security, space-grade resilient circuits and systems, asynchronous VLSI designs, low-voltage low power VLSI circuits, and signal processing.

Dr. Chong has served as an organizing committee for several conferences, including the ASP-DAC 2014, DSP-2015, DSP-2018, SOCC2019 and ISICAS2021. He was also the Chair of IEEE Circuits and Systems (CAS) Society, Singapore Chapter, in 2017 and 2018. He has been a member of IEEE CAS Society VLSI Systems and Applications Technical Committee since 2009.

Chao Wang (Senior Member, IEEE) received his B.Eng. and Ph.D degrees, both in Electronics Engineering, from Huazhong University of Science and Technology (HUST), China in 2000, and Nanyang Technological University (NTU), Singapore in 2008, respectively.

From 2005 to 2007, he was also with the Center for Signal Processing, NTU as a Research Engineer. Since then, he worked as an IC Design Engineer with STMicroelectronics, Asia-Pacific Design Centre, Singapore, from 2008 to 2010. He participated in the development of STM Bayer/YUV CMOS image sensing and processing ICs/SoCs. From 2010 to 2017, he was a Research Scientist and a Project Leader with the Institute of Microelectronics (IME), Agency for Science, Technology and Research (A*STAR), Singapore, where he led projects on wearable medical devices for cardio-engineering applications, and MEMS Accelerometer ASICs for medical and navigation applications. Currently, he is a Professor with the HUST and also with Wuhan National Laboratory of Optoelectronics (WHLO), Wuhan, China. His major research interests include energy-efficient digital signal processor design, ultra-low-voltage circuit design, MEMS sensor ASIC design, and heterogeneous 3D-IC design, especially for biomedical/healthcare, wireless sensor, IoT, neuromorphic computing and robot applications.

Dr. Wang is an IEEE Senior Member and currently the chair of IEEE CASS-EDS-SSCS Wuhan Joint Chapter. He used to serve as a committee member, the vice chair and chair of IEEE SSCS Singapore Chapter. He has also served as TPC members for a number of international devices, circuits and systems conferences. He served as a Guest Editor for Sensors journal, Hindawi in 2016, and IEEE TRANSACTIONS ON BIOMEDICAL CIRCUITS AND SYSTEMS in 2019. He is also an Associate Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS, *IEEE Circuits and Systems Magazine*, IEEE ACCESS journal, and *Microelectronics* Journal (Elsevier).