An Energy-Efficient BNN Accelerator With Two-Stage Value Prediction for Sparse-Edge Gesture Recognition

Yongliang Zhang, Yitong Rong, Xuyang Duan, Zhen Yang, Qiang Li, Ziyu Guo, *Member, IEEE*, Xu Cheng, *Member, IEEE*, Xiaoyang Zeng, *Member, IEEE*, and Jun Han, *Member, IEEE*

***Abstract*— In recent years, natural, flexible, and contactless vision-based gesture recognition has received significant attention in human-computer interaction. However, employing convolutional neural networks (CNNs) for RGB or RGB-D gestures can result in excessive power consumption and poor energy efficiency, making them unsuitable for embedded systems. In this paper, we propose a lightweight sparse binarized neural network (sBNN) model for edge gesture recognition that achieves an accuracy of 89.43%-99.92% on four open-source gesture datasets with ≤20.26 million operations (MOP) and ≤15.83-Kilobytes (KB) parameters. We find high channel-level sparsity in the activation maps of sBNN when edge gestures are used as inputs. The sparse activation maps have multiple identical activation vectors called sparse activation vectors (SAV), which lead to highly repeated calculations. In order to avoid this issue, we propose a two-stage value prediction approach to skip these calculations, achieving a speedup of 1.03x-1.83x. Moreover, to reduce on-chip memory, the compression technique is applied to the sparse activation maps, providing a compression rate of 1.72x-3.45x. Finally, we implement an energy-efficient sparse BNN accelerator (SBA) on an embedded field-programmable gate array (FPGA). The experimental results show that our SBA has a latency of 26.3-46.8-μs, a power consumption of 0.807 W, and an energy efficiency of 536.22-952.70-GOPS/W at 50-MHz frequency. Our SBA offers lower latency, lower power consumption, and higher energy efficiency than previous state-of-the-art gesture recognition accelerators.**

***Index Terms*— Gesture recognition, sparse BNN, compression technique, value prediction, FPGA.**

I. Introduction

GESTURE recognition [1], [2] is crucial to human-computer interaction, offering non-contact, long-distance, and flexible advantages which have various applications in smart homes, virtual reality, augmented reality, and more. Gesture recognition methods can be classified into sensor-based and vision-based methods. While the sensor-based method [3], [4] has been extensively studied, it is either uncomfortable to wear or vulnerable to environmental effects. In contrast, the vision-based method [5], [6] uses RGB or RGB-D images for gesture recognition. Traditional machine learning algorithms are easily influenced by environmental conditions, while convolutional neural networks (CNN) are emerging as powerful tools for gesture recognition due to their superior learning capability and broad applicability. For example, using an embedded CNN model, Wang [7] achieves a recognition accuracy of 99.96% for RGB-D American Sign Language (ASL) [8]. Moreover, Lu [5] demonstrates that even four binarized gesture edges can achieve high accuracy with a CNN model. As we know, the hardware's energy efficiency negatively correlates with the number of parameters and computations. In contrast, accuracy and classification quantity positively correlate with the number of parameters and computations. CNN models typically employ numerous parameters and extensive calculations, resulting in high latency and low energy efficiency, making them unsuitable for embedded systems.

As a similar structure to CNN, binarized neural networks (BNN) [9] implement convolution operations with XNOR-Popcount instead of multiply-accumulate (MAC), achieving high energy efficiency at the cost of reduced precision. However, most BNN studies for gesture recognition [10] have focused solely on RGB images. The MAC convolution operations in the first layer of the BNN entail additional resource overhead and power consumption. To address this issue, binarized gestures [11], with insignificance texture information and important gesture edges, have been used. In this paper, we apply binarized sparse edge gestures (SEG) as input after removing the internal texture information. During forward inference, we have the insight that taking SEG as input leads to channel-level sparse activation maps (SAM), which contain many repeated constant vectors along the activation maps' channel dimension. These repeated constant vectors in activation maps, called sparse activation vectors (SAV), can be predicted to skip calculations and compressed to reduce data storage. In CNN, a significant number of input values are constant (zeros), and their corresponding partial sums do not contribute to the final result [12], [13]. However, in BNN, where input values are limited to -1 or $+1$, all partial sums are non-negligible. Fortunately, because of the sparsity brought by SEG, a large number of SAVs can be predicted to skip repeated calculations.

Manuscript received 1 May 2023; revised 26 August 2023; accepted 21 September 2023. Date of publication 9 October 2023; date of current version 12 January 2024. This work was supported by the National Natural Science Foundation of China under Grant 61934002. This article was recommended by Associate Editor X. Zhang. *(Corresponding author: Jun Han.)*

The authors are with the State Key Laboratory of Integrated Chips and Systems, Fudan University, Shanghai 201203, China (e-mail: junhan@fudan.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2023.3320175.

Digital Object Identifier 10.1109/TCSI.2023.3320175

The large number of SAV in the activation maps provides opportunities for further improvements in the energy efficiency of BNN. With many convolution results that can be predicted, we explore the value prediction approach's role in addressing repeated calculations in sBNN. Value prediction approaches are often employed in general-purpose processors to solve the computational burden resulting from a significant number of repeated calculations [14]. Given that many SAVs can be predicted in the sBNN model, we introduce a value prediction approach in the calculations of sBNN to skip repeated calculations, speed up calculations, and improve energy efficiency. Additionally, compression and decompression method are also introduced into the calculation of sBNN to reduce data storage and power consumption.

In this paper, we propose a lightweight sparse BNN (sBNN) model for binarized SEG recognition. Because of the sparsity of SEG, there are many channel-level SAVs and repeated calculations. We introduce two-stage value prediction to accelerate our sBNN model and SAM compression and decompression for competitive energy efficiency. Contributions to our study are summarized as follows:

1) An SEG recognition algorithm with a lightweight sBNN model, which has the advantages of a low number of parameters, low computations, and high accuracy for four open-source gesture datasets. It has $\leq$15.83 Kilobyte (KB) parameters, $\leq$20.26 million operations (MOP), and 89.43% - 99.92% accuracy for gesture datasets.
2) Energy-efficient approaches for sBNN, including a two-stage value prediction and a channel-level sparse activation maps compression and decompression. The energy-efficient two-stage value prediction approach can skip repeated calculations to speed up sBNN computation and reduce power consumption. The channel-level sparse activation compression and decompression can reduce the required data storage.
3) A novel real-time sparse BNN accelerator (SBA) based on the Genesys2 board, which has a latency of 26.3 - 46.8 μs, a power consumption of 0.807 W, and an energy efficiency of 536.22 - 952.70 GOPS/W at 50 MHz frequency.

The remainder of this paper is structured as follows. Section II provides a review of related work on gesture recognition. Section III describes the proposed SEG recognition algorithm. Section IV presents energy-efficient approaches for sBNN. Section V presents the hardware implementation of SBA. Finally, Section VI provides a conclusion.

II. Related Work

As mentioned in Section I, gesture recognition can be achieved through sensor-based or vision-based. The vision-based methods use various types of input data such as RGB-D images, RGB images, depth images, or extracted features from these images. Based on feature extraction or not, we categorize the vision-based methods into feature extraction and classification methods and end-to-end classification methods.

A. Sensor-Based Methods

The sensor-based methods can be further divided into two types: wearable sensors and radar sensors. Wearable gloves [15] containing multiple sensors have been studied extensively in gesture recognition. In [16], a gesture recognition glove based on charge-transfer touch sensors is developed and used to recognize gestures for the numbers 0-9 and 26 English alphabets. However, while gesture recognition based on wearable sensors can achieve high precision, it is limited to sports and provides a poor wearing experience. In the last decade, radar sensors have garnered increasing attention for gesture recognition. Zhang et al. [17] achieve high recognition rates of 96% by using frequency-modulated continuous waves to classify dynamic continuous gestures. However, radar waves may be lost during transmission or absorbed by environmental objects, leading to a poor recognition effect.

B. Vision-Based Methods

1) End-to-End Classification Methods: In the end-to-end classification methods, gesture recognition is achieved by directly processing the original input image. For example, Garcia [18] uses GoogLeNet [19] architecture to recognize RGB American Sign Language (ASL) fingerspelling with over 90% accuracy. Visual devices such as Kinect [20] and Leap Motion Controller [21], which use depth cameras, have become popular in gesture recognition. The dual-path depth-aware attention network [22] has been proposed to recognize RGB-D ASL to improve accuracy. However, the end-to-end methods are computationally-intensive and parameter-intensive, resulting in increased power consumption and delay for embedded systems as the CNN models become more complex with more detailed input images.

2) Feature Extraction and Classification Methods: In the field of gesture recognition, feature extraction is crucial. Gesture features are extracted to reduce information and improve classification accuracy. One commonly used method is gesture segmentation, which separates gestures from the background based on attributes such as color, depth, and edge. Huang et al. [23] segment the RGB hand region based on skin color and gesture contour and recognize 10 number gestures with a CNN model, achieving 98.41% accuracy. Depth can effectively handle illumination issues. The Kinect contains information about the human skeleton, and Wang et al. [7] utilize the Kinect skeleton tracking system to separate the RGB-D gesture for embedded CNN recognition, achieving 99.96% accuracy on the ASL dataset. It is true that segmented gestures achieve good recognition results, but the redundant information in the segmented gestures still results in high parameters and calculations.

To remove redundant information in segmented gestures, several studies have achieved effective recognition results by further extracting segmented gestures. For example, in the study by Lu et al. [5], the four sides of the segmented gesture are taken as input for a two-layer CNN, achieving a recognition accuracy of 90.3% for six static gestures. The combination of these static gestures results in 24 dynamic gestures. However, this method just suitable for simple gestures.

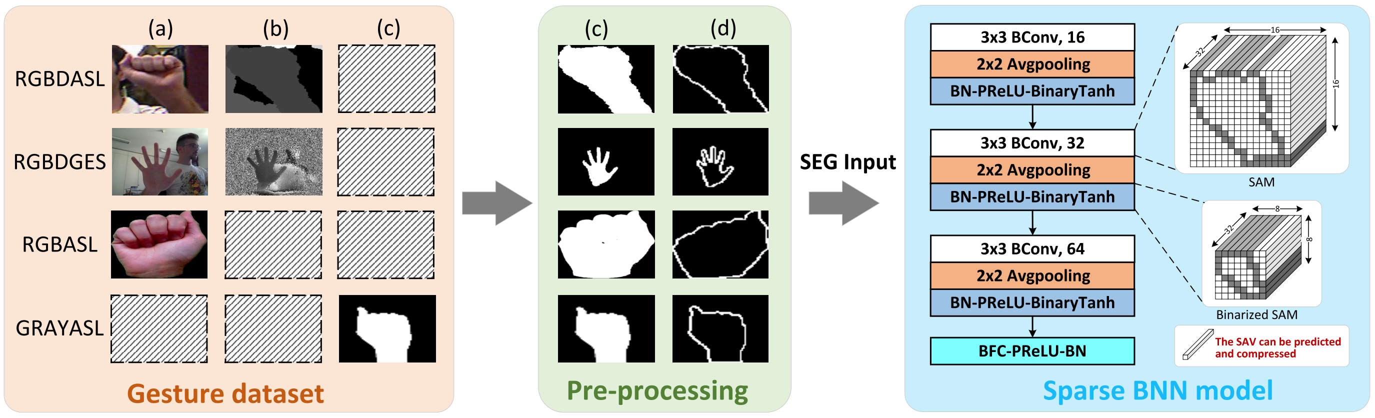

Fig. 1. The proposed sparse edge gesture recognition algorithm. After pre-processing, sparse edge gesture images can be extracted. A lightweight sparse BNN architecture is proposed for sparse edge gesture classification. (a) The RGB gesture images, (b) the depth gesture images, (c) the binarized gesture images, and (d) the sparse edge gesture images.

Abdulhussein et al. [24] apply the SEG image that is the segmented gesture's outline as input for recognition using a 4-layer CNN, which achieves an accuracy of 99.3% on recognizing 26 ASL gestures with many computations and parameters. However, most gesture recognition methods are limited to the CNN recognition models, which are computation-intensive and parameter-intensive, leading to energy inefficiency. Our goal is to develop a low-parameter, low-computing algorithm for SEG images with high accuracy and build an energy-efficient gesture recognition accelerator for embedded systems with low latency.

III. Proposed SEG Recognition Algorithm

Figure 1 depicts the proposed SEG recognition algorithm. The algorithm comprises two main blocks: pre-processing and a lightweight sBNN architecture. For SEG recognition, we employ four general-purpose open-source gesture datasets, namely, RGB-D gesture dataset (RGBDGES) [25], [26], RGBDASL [8], RGBASL [27], and GRAYASL [28]. SEG images are generated through pre-processing, which includes gesture segmentation and edge detection. To recognize SEGs, we propose a lightweight sBNN architecture. The SEG images are cropped and input to the lightweight sBNN architecture for gesture recognition. The recognition results are then converted into text and displayed. In addition, we introduce channel-level SAV in sBNN according to the sparsity of the SEG input image, and we adopt SAV padding to increase the number of SAV. The detailed functionalities of each block will be described in the following subsections.

A. Pre-Processing

Several CNN models employ RGB-D, depth, or RGB gestures to achieve high recognition performance, increasing power consumption in embedded systems. In contrast, our experiments demonstrate that gesture shape, as a distinguishable feature of gestures, can meet recognition requirements for a wide range of gesture recognition tasks. Figure 1 (d) displays SEG images from various datasets. Due to their 1-bit and sparse properties, SEG images reduce the computations of the sBNN model during forward inference. We conduct

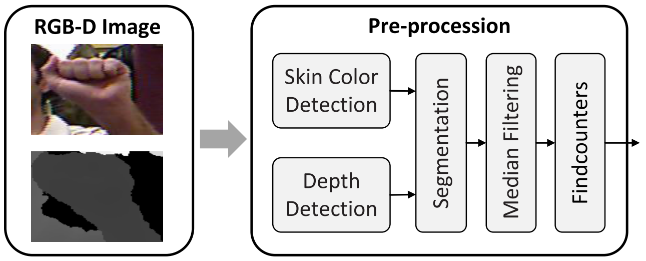

Fig. 2. The operations of pre-processing.

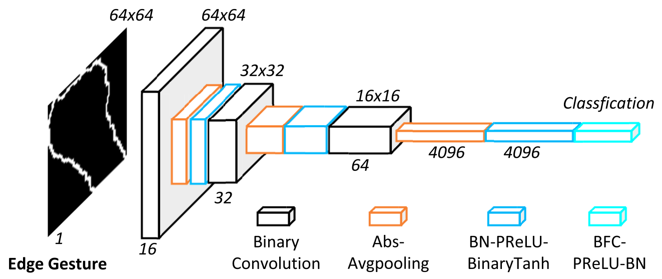

Fig. 3. Our sparse BNN architecture with a kernel size 3 × 3 for all filters for sparse edge gesture classification.

gesture segmentation to generate binarized gestures. For the RGBDGES and RGBDASL, the gesture segmentation is based on depth gestures, and the hand closest to the camera is prioritized. For RGBASL, skin color detection algorithms are used for gesture segmentation. For GRAYASL, gesture segmentation is already completed. After gesture segmentation, the edges of the binarized gestures are extracted using the edge detection method to generate SEGs.

As shown in Figure 2, we use an RGB-D image to illustrate the four steps in the pre-processing stage: skin color/depth detection, segmentation, median filtering, and finding contours. Initially, skin color/depth detection extracts the gesture area by either analyzing the skin color distribution within the color field or relying on the principle that the gesture is closest to the acquisition device. Then, all of the subsequent steps contribute to the generation of a sparse edge gesture. In addition, gesture recognition pre-processing usually occurs on the acquisition device to avoid high data storage and

TABLE I
THE DETAILED ARCHITECTURE OF THE sBNN MODEL FOR GESTURE DATASETS

Dataset		RGB-DGES	RGBD-ASL	RGB-ASL	GASL
BConv1	Input size	[1, 64, 64]			
	Kernel size	[16, 1, 3, 3]			
	Output size	[16, 64, 64]			
	Parameters	0.0175 KB			
	Operations	1.18 MOP			
Avg-Pooling1	Kernel size	[2, 2]			
	Stride	[2, 2]			
BConv2	Input size	[16, 32, 32]			
	Kernel size	[32, 16, 3, 3]			
	Output size	[32, 32, 32]			
	Parameters	0.5625 KB			
	Operations	9.437 MOP			
Avg-Pooling2	Kernel size	[2, 2]			
	Stride	[2, 2]			
BConv3	Input size	[32, 16, 16]			
	Kernel size	[64, 32, 3, 3]			
	Output size	[64, 16, 16]			
	Parameters	2.25 KB			
	Operations	9.437 MOP			
Avg-Pooling3	Kernel size	[2, 2]			
	Stride	[2, 2]			
BFC	Input size	4096			
	Kernel size	[11, 4096]	[24, 4096]	[26, 4096]	[26, 4096]
	Output size	11	24	26	26
	Parameters	5.5 KB	12 KB	13 KB	13 KB
	Operations	0.090 MOP	0.197 MOP	0.213 MOP	0.213 MOP
Accuracy		89.43 %	95.34 %	97.63 %	99.92 %
Total parameter		8.33 KB	14.83 KB	15.83 KB	15.83 KB
Total operation		20.10 MOP	20.25 MOP	20.26 MOP	20.26 MOP

Note: Input size: [C_{in}, H_{in}, W_{in}], Output size: [C_{out}, H_{out}, W_{out}], Kernel size: [C_{out}, C_{in}, H_{kernel}, W_{kernel}].

handling costs in hardware accelerators. Many acquisition devices integrate various image processing operations and provide unique processing application programming interfaces, providing more powerful and diversified pre-processing methods. In our demonstration system, we utilize a PC to emulate the pre-processing operations typically performed by the acquisition device for input gestures. The post-processing gesture is streamed in real-time to the Genesys2 board through HDMI.

B. Lightweight sBNN Architecture for SEGs Classification

Figure 3 illustrates the proposed lightweight sBNN architecture. To accommodate the binary nature of the SEG image, the first convolutional layer of sBNN employs a binary convolutional (BConv) layer instead of the MAC convolutional layer. Specifically, the 1st, 2nd, and 3rd BConv layers have 16 output channels of a kernel size 3×3 with stride 1, 32 output channels of a kernel size 3×3 with stride 1, and 64 output channels of a kernel size 3×3 with stride 1, respectively. Moreover, a binary fully connected (BFC) layer followed by a PReLU activation layer and a batch normalization (BN) layer, with a size of 4096, is also included in sBNN. Each BConv layer is followed by an absolute (abs) activation layer and an average pooling layer (avgpooling) with a kernel size of 2×2 and stride 2. In addition, the BN layer, PReLU layer, and binary HardTanh activation (BinaryTanh) layer are applied to the outputs from the avgpooling layer. The BinaryTanh layer binarizes the output activation maps, which become the next layer's input activation maps. All convolution calculations are performed with XNOR-Popcount. Table I shows that the sBNN model incorporates several BConv and avgpooling layers with specific layer sizes and dimensions.

Benefiting from weight and activation constraints of $+1$ and -1, sBNN models can replace MAC operations with XNOR-Popcount operations by binarizing both the weights and activations using the sign function, where

$$\mathrm{sign}(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ -1, & \text{if } x < 0. \end{cases} \tag{1}$$

During forward inference, the sBNN model undergoes several calculation processes, including the BConv layer, the abs activation layer, the avgpooling layer, the BN layer, the PReLU layer, and the binary HardTanh activation layer, which are executed in the following sequence

$$\begin{aligned}
\mathbf{Y}_{BConv} &= \mathrm{sign}(\mathbf{I}) \odot \mathrm{sign}(\mathbf{W}),\\
\mathbf{Y}_{abs} &= abs(\mathbf{Y}_{BConv}),\\
\mathbf{Y}_{avgpooling} &= \frac{1}{|\mathbf{R}_{ij}|} \sum_{(p,q)\epsilon\mathbf{R}_{ij}} (\mathbf{Y}_{abs})_{pq},\\
\mathbf{Y}_{BN} &= \frac{\mathbf{Y}_{avgpooling} - \mu}{\sqrt{\sigma^2}}\gamma + \beta,\\
\mathbf{Y}_{PReLU} &= \begin{cases} \mathbf{Y}_{BN}, & \text{if} \mathbf{Y}_{BN} > 0 \\ \alpha\mathbf{Y}_{BN}, & \text{if} \mathbf{Y}_{BN} \leq 0, \end{cases}\\
\mathbf{Y}_{HardTanh} &= \begin{cases} 1, & \text{if } \mathbf{Y}_{PReLU} > 1 \\ \text{-1}, & \text{if } \mathbf{Y}_{PReLU} < -1 \\ \mathbf{Y}_{PReLU}, & \text{otherwise,} \end{cases}\\
\mathbf{Y}_{binarized} &= \mathrm{sign}(\mathbf{Y}_{HardTanh}),
\end{aligned} \tag{2}$$

where $\odot$, $\mathbf{R}_{ij}$, $|\mathbf{R}_{ij}|$, i and j represent the XNOR-Popcount convolution operations, avgpooling array, the number of parameters in avgpooling array, width and height of avgpooling array, respectively. Additionally, **I**, **W**, and **Y** represent the input activation maps, the weight data, and the output activation maps. As the parameter α in the PReLU activation layer is typically greater than 0, it does not impact the sign of $\mathbf{Y}_{BN}$ and can be excluded during forward inference. Likewise, the HardTanh activation layer can also be disregarded. Therefore, $\mathbf{Y}_{binarized}$ is solely determined by the sign of $\mathbf{Y}_{BN}$.

To reduce MAC computations, the BN layer can be further optimized by converting the MAC operation into a comparison process [29]. It is essential to extract the comparison thresholds for the BN layer. The BN layer's parameters are known in the forward inference, and its calculations for each output channel can be viewed as a linear function. The comparison

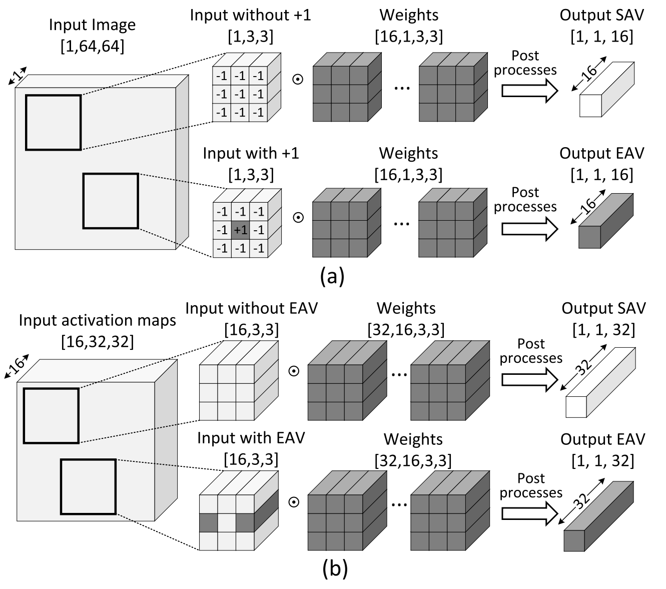

Fig. 4. The processes of generating output sparse activation vector and effective activation vector. (a) In the BConv1 layer, the output sparse activation vector and effective activation vector are generated by the 3 × 3 inputs filled with only -1 and at least one +1, respectively. (b) In the BConv2 layer, the output sparse activation vector and effective activation vector are generated by the 3 × 3 inputs filled with only input sparse activation vector and at least one input effective activation vector, respectively.

threshold for each output channel, which is the result of the linear function equal to zero, can be pre-calculated and called Th_{BN}. The PReLU activation and BN layers after BFC require MAC operations to obtain the final classification. The detailed architecture of the sBNN model for gesture datasets is presented in Table I, where C is the channel, H is the height, and W is the width. The lightweight sBNN architecture achieves high recognition accuracy with a small number of parameters and operations. For example, the sBNN model attains 95.34% accuracy with only 20.25 MOP and 14.83 KB parameters for the RGBDASL dataset.

C. Channel-Level Sparse Activation Vectors

The sparsity of the SEG image causes a significant number of pixels to be set to -1, resulting in numerous 3 × 3 blocks containing only -1 values. When the 3×3 block is convoluted with the weights in the BConv1 layer, the output of each channel remains constant during forward inference due to the known input and weight data. As shown in Figure 4 (a), the BConv1 layer has 16 output channels with a kernel size of 3 × 3, producing 16 invariant constants when a 3 × 3 input is filled with only -1 values. After the abs, avgpooling, BN-PReLU, and BinaryTanh layers, the results whose size is 1 × 1×16 are named output SAV. The high number of 3 × 3 inputs filled with only -1 values in the SEG image generates numerous SAVs in the BConv1 layer. Other output vectors that differ from SAV are called output effective activation vectors (EAV), whose corresponding input is a 3 × 3 block with at least one $+1$ value. When used as input at the next layer, the output SAV and EAV will become the input SAV and input EAV.

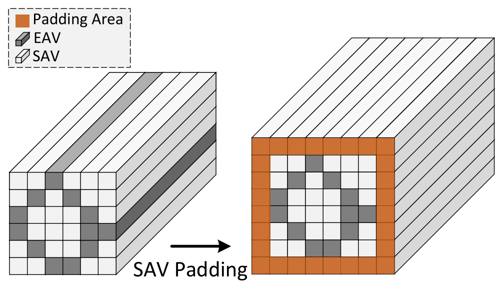

Fig. 5. Sparse activation vector padding replaces the zero padding to increase the proportion of sparse activation vector in ifmaps.

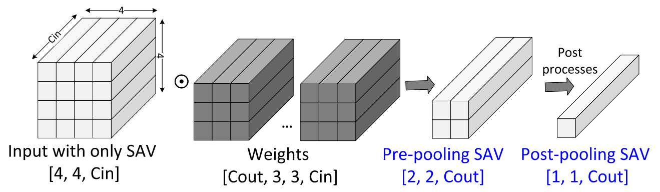

Fig. 6. Two types of predicted values are obtained during offline value prediction.

Figure 4 (b) illustrates that the BConv2 layer has 32 output channels of a kernel size 3 × 3, producing a new output SAV when a 3 × 3 input is filled with only input SAV. The size of the output SAV is 1 × 1×32. In contrast, a new output EAV is generated when the 3 × 3 input of this layer includes at least one input EAV. Since the input activation maps exist numerous SAVs in the BConv2 layer, output SAV also occupies a higher proportion in output activation maps. Similarly, for the BConv3 layer, the output SAV is generated by a 3 × 3 input filled with only input SAV. It should be noted that the SAV sizes are inconsistent across layers, with 1 × 1×16, 1 × 1×32, and 1 × 1×64 in the 1st, 2nd, and 3rd BConv layers, respectively. During forward inference, the SAV of each layer can be predicted directly.

The SAV can be predicted in advance and has a high proportion in activation maps, allowing hardware design to skip calculations and compress data. Increasing the proportion of SAV in the input activation maps will improve the data-compression effect and calculation-skipping amount. Based on the computation process and the generation principle of SAV in the entire sBNN, we find that using SAV padding can introduce a higher proportion of SAV and more repeated computations to the input activation maps of each layer. Besides, zero padding is currently the most used padding method in BNN and produces the most potent recognition effect. However, it introduces ternary calculation (-1, 0, $+1$) to the convolution computation, complicating the BNN convolution computation. SAV padding can solve this problem because all values in SAV are either -1 or $+1$.

As shown in Figure 5, we utilize the input SAV of each layer as SAV padding, which increases the proportion of SAV in each input activation image. SAV is a channel-level highly repetitive special vector which is a collection of special elements in each channel. The SAV padding of each input channel is the sparse element of that channel. In addition, SAV padding is added during training and constantly updated

TABLE II
THE PROPORTION OF SAV AT EACH BCONV LAYER AND ACCURACY FOR VARIOUS DATASETS

Dataset	Type	Edge Count	SAV rate with zero padding			Accuracy (%)	SAV rate with SAV padding			Accuracy (%)
			BConv1 (%)	BConv2 (%)	BConv3 (%)		BConv1 (%)	BConv2 (%)	BConv3 (%)	
RGBDGES	LGE	287	87.44	61.85	20.67	90.06	93.41	77.87	37.26	89.43
	SME	124	91.18	69.54	30.69		97.15	83.10	62.86	
RGBDASL	LGE	303	87.08	61.30	19.70	95.47	93.04	72.64	50.14	95.34
	SME	180	89.90	66.47	24.11		95.87	88.24	65.31	
RGBASL	LGE	237	88.59	63.55	19.44	97.63	94.56	81.33	48.16	97.04
	SME	98	91.78	76.91	36.42		97.75	93.24	73.77	
GRAYASL	LGE	361	85.74	57.96	17.90	99.98	91.71	69.54	49.07	99.92
	SME	110	91.50	77.77	39.81		97.47	89.19	76.54	

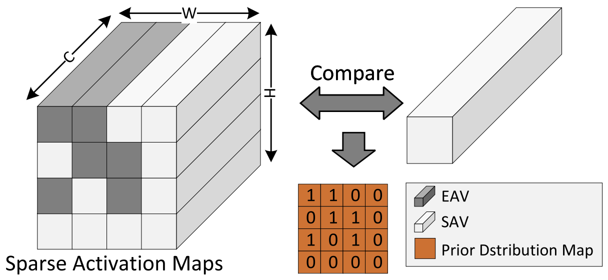

Fig. 7. Prior distribution map is generated by predicted sparse activation vector.

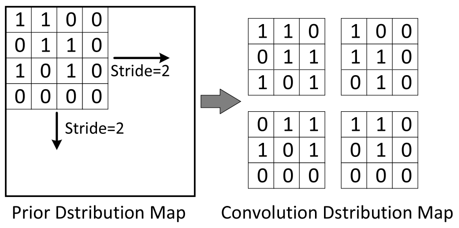

Fig. 8. A 4×4 prior distribution map block, obtained by a 4×4 convolution window slides with a stride of 2 on the prior distribution map, can be divided into four 3×3 prior distribution map blocks.

once weights are adjusted. During sBNN training, calculations are divided into forward calculation, reverse calculation, and weight updating. As the weights are constant values in the forward calculation, the SAV padding for each layer will temporarily remain the same. When the 3×3 SAV is used as an input block, the generated output SAV will be employed as SAV padding for the next layer. In addition, the SAV padding value of each channel is a random number ($+1$ or -1), which is changed when the weights are updated.

Table II illustrates the proportions of SAV in each BConv layer and accuracy for each dataset with zero padding and SAV padding. The SEG with the highest edge count, called the largest edge gesture (LGE), and the SEG with the lowest edge count, called the smallest edge gesture (SME), represent the maximum and minimum edge counts, respectively. Compared with zero padding, SAV padding brings a significantly higher proportion of SAV to the input activation map of each layer, with only a minor trade-off in accuracy. The increase in SAV proportion will bring better compression and acceleration effects to the sBNN hardware.

TABLE III
COMPARISON WITH STATE-OF-THE-ART METHODS ON THE RGBDASL DATASET

Method	Parameters	Operations	Accuracy
Random Forest [8]	-	-	75.00%
SVM [30]	-	-	91.26%
VGG19-SVM [31]	143 MB	19.63 GOP	98.44%
Multi-Stream CNN [32]	1.18 GB	-	99.60%
Resnet-18 [33]	11.68 MB	1.80 GOP	**99.91%**
eCNN [7]	0.17 MB	1.82 GOP	99.26%
Our sBNN	**14.83 KB**	**20.25 MOP**	95.34%

D. Network Evaluation

To achieve an energy-efficient gesture recognition accelerator, we make a trade-off between accuracy and parameter-computation. As shown in Table III, we compare our proposed sBNN models with various gesture classification methods on the RGBDASL dataset. Traditional machine learning algorithms such as random forest [8] and support vector machine (SVM) [30] are not effective at gesture recognition. Although the CNN models [7], [31], [32], [33] can obtain extremely high recognition accuracy, they require a large number of parameters and calculations. CNN's calculations and parameters are measured after int-8 weight/activation quantization. Compared with state-of-the-arts, our sBNN sacrifices accuracy (about 4%) to reduce more than 10x calculations and parameters, which reduces storage and calculation cycles and optimizes the energy efficiency of our gesture accelerator. Our sBNN only requires 14.83 KB parameters and 20.25 MOP calculations for the RGBDASL dataset.

IV. VALUE PREDICTION AND CHANNEL-LEVEL SAM COMPRESSION APPROACHES

In Section III, we present a detailed description of our proposed sBNN architecture and highlight the significant presence of SAV in the activation map. We have confirmed many SAVs in activation maps and high-repetition calculations in sBNN, which can be further optimized. To improve the energy efficiency of the sBNN, we focus on two main approaches: 1) accelerating sBNN by utilizing a two-stage value prediction and 2) exploiting channel-level sparsity to compress and decompress SAM.

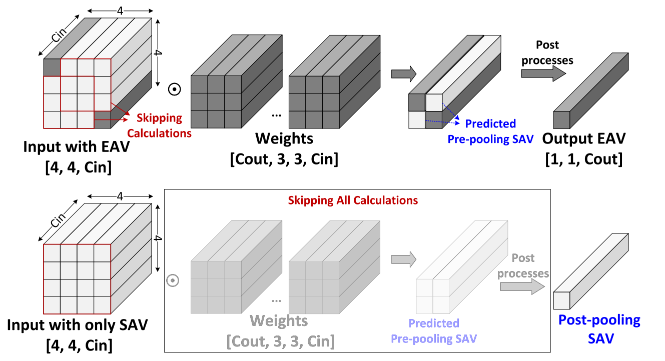

Fig. 9. Filter-level and group-level skipping modes in online value prediction method.

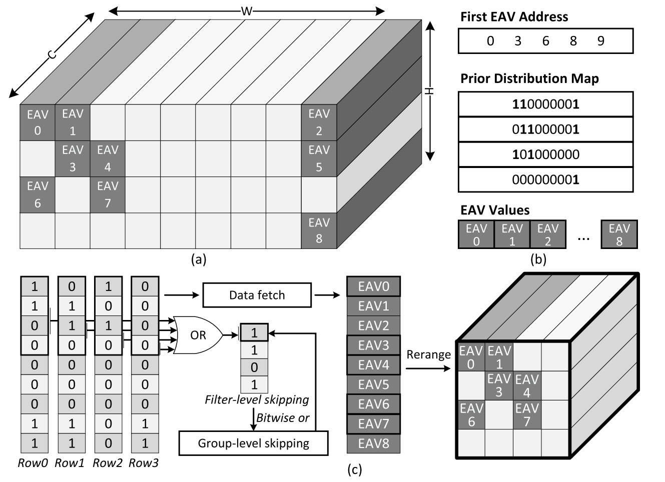

Fig. 10. Sparse activation map compression and decompression. (a) The original activation maps, (b) compressed data after compression, and (c) decompression operations.

A. Energy-Efficient Two-Stage Value Prediction

Value prediction is a technology used in micro-architecture to increase instruction-level parallelism in out-of-order processor cores [34]. It is also used in CNN to exploit the spatial correlation inherent and skip part of the MAC operations [35]. However, the value prediction technique may cause recalculations after pipeline flush when a prediction error occurs or comes at the cost of some accuracy.

To solve the problems, we propose a two-stage value prediction approach to achieve infallible prediction and absolutely no recalculation, increasing energy efficiency in sBNN. Using SEG input introduces numerous SAVs in each layer's input activation maps, which leads to repeated computations. Our two-stage value prediction method can skip these repeated computations. In the first stage, we infallibly predict the output SAV values for each layer in advance using offline value prediction. In the second stage, the hardware performs online calculations to complete the non-repeated part of the computations. In contrast, predicted SAVs, the results of repeated calculations, are buffered in hardware to omit the calculations. We obtain infallible results through the two-stage value prediction approach while skipping repeated calculations, achieving higher energy efficiency without compromising accuracy.

1) Stage 1: Offline Value Prediction: The offline value prediction can infallibly predict the SAV values of each layer based on known weights, preparing for calculation skipping in hardware. When the model is retrained, the SAV of each layer can be obtained at a negligible cost.

There are two types of SAV that are predicted in advance by the offline value prediction. The first type is pre-pooling SAVs which are the BConv results before the avgpooling layer. The second type is post-pooling SAVs, representing binarized activation vectors after avgpooling and the BinaryTanh layer. Figure 6 shows that when a 4×4 SAV block is provided as input under known weights, the offline value prediction method generates 2×2 pre-pooling SAVs for each output channel. During the subsequent avgpooling operation, these pre-pooling SAVs are accumulated, and a post-pooling SAV is obtained after the BN-PReLU and BinaryTanh layers.

2) Stage 2: Online Value Prediction: The online value prediction can skip repeated calculations based on offline predicted SAV, avoiding the negative effects of traditional value prediction techniques (e.g., recalculation process and recognition accuracy degradation). In traditional convolution processes, the convolution kernel slides over the input activation maps, requiring continuous access to the repeated SAV, which leads to repeated calculations. Online value prediction technology applies offline predicted SAV to encode online activation maps. By obtaining the distribution of SAV and EAV in advance during the calculation of the next layer, only the convolution with EAV is calculated, and the convolution result corresponding to SAV is provided by the offline predicted SAV of the next layer. The offline value prediction is divided into two processes. First, a prior distribution map (PDM) is generated based on the predicted SAV for representing the distribution of SAV and EAV in the output activation maps. It is then determined whether to perform or skip the convolution calculation based on whether includes 1 value in the 3×3 PDM block. When there exit no 1 in the 3×3 PDM block, the redundant convolution calculation will be skipped. Otherwise, valid computations continue to be performed.

In Figure 7, a PDM is generated for the entire SAM based on the comparison with predicted SAV. Each value of the PDM is 1-bit, where 0 indicates SAV data, and 1 indicates EAV data. The size of the PDM is $[H_{in}, W_{in}]$, which is the same as the input activation maps. Once the PDM is obtained, convolution calculations are performed by sliding on the PDM, as shown in Figure 8. Since there is an avgpooling layer in the model, a 4×4 sliding window size and a stride of 2 are used for convolution calculations. Each sliding window contains four 3×3 PDM blocks, where each 3×3 PDM block represents an entire convolution calculation. We classify each 3×3 PDM block into two types: the 3×3 PDM block with only 0 is called a sparse PDM block, and the others are called efficient PDM blocks.

When the four 3×3 PDM blocks enter the calculation, online value prediction will appear in two calculation skipping modes: filter-level skipping and group-level skipping, as shown in Figure 9. In the filter-level skipping mode, the four PDM blocks exit at least one efficient PDM block, and only the efficient PDM blocks will be calculated. The calculations for the sparse PDM blocks are skipped, and the results are

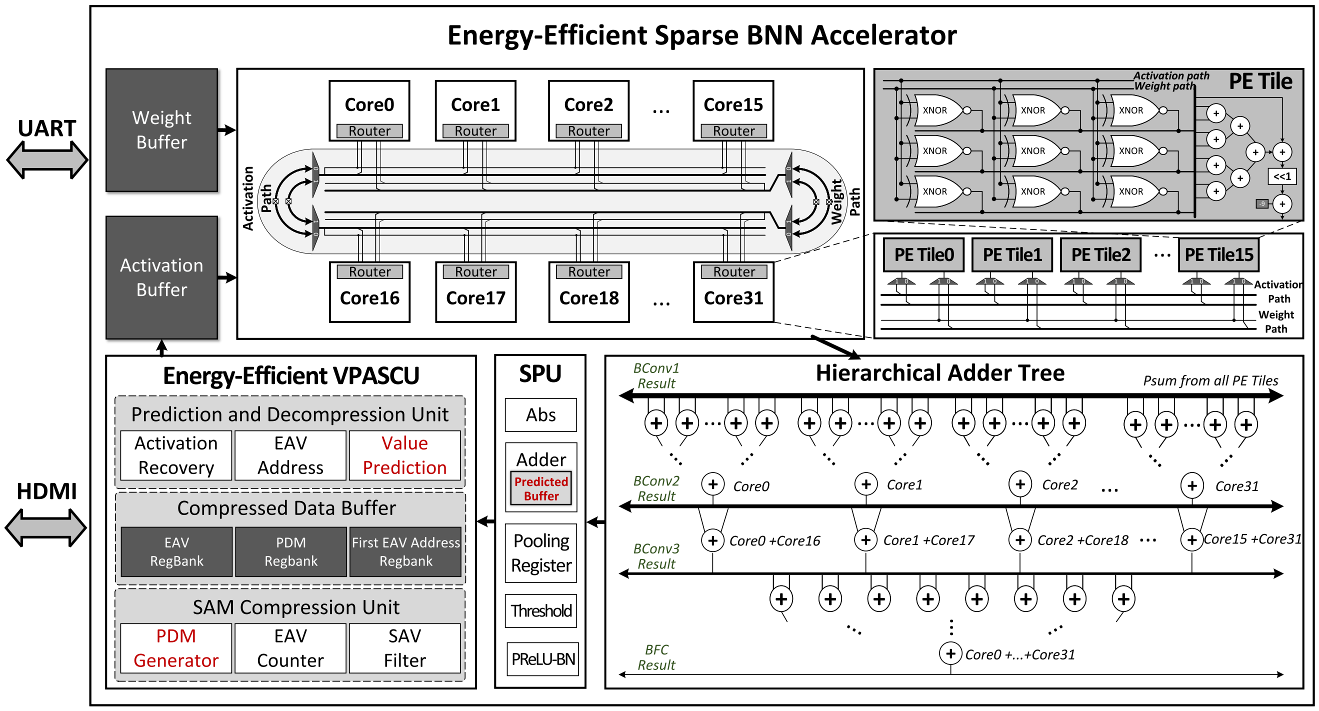

Fig. 11. The overall architecture of our sparse BNN accelerator.

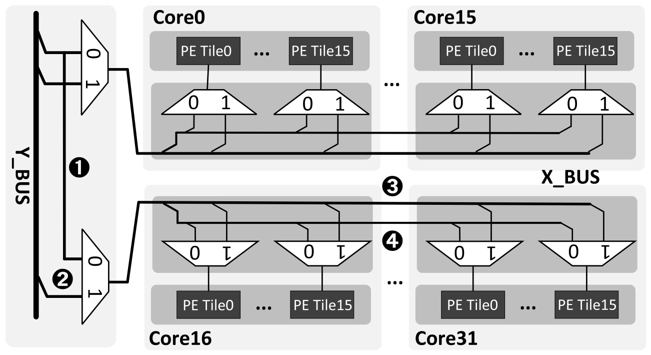

Fig. 12. The architecture of network on chip.

pre-pooling SAV obtained from offline value prediction. The results of the four PDM blocks are accumulated to generate a new EAV. In the group-level skipping mode, four PDM blocks are sparse PDM blocks, and all calculations are skipped. Then the result is the post-pooling SAV provided by offline prediction.

B. Channel-Level SAM Compression and Decompression

To enhance energy efficiency further, we apply the SAM compression and decompression approach for numerous identical SAVs, storing EAV and encoding information. By the variant of the compress sparse row format [36], we replace the column coordinates with the PDM and save the first EAV address, as shown in Figure 10 (b). Upon entering the next BConv layer, the compressed SAM is decompressed to provide input data for convolution calculations.

The compression ratio (CR) is the result of dividing uncompressed data by compressed data. For each layer, the size of output activation maps is $[H_{out}, W_{out}, C_{out}]$. The size of the first EAV address is $M \times H_{out}$ bits. M equals to ceiling($\log_2 W_{out}$), and the ceiling function gives the smallest nearest integer that is greater than or equal to the specified value. The size of PDM is $H_{out} \times W_{out}$ bits. EAV values are $N \times C_{out}$ bits, where N is the number of EAV values. Therefore, the compressed data (CD) is

$$CD = M \times H_{out} + H_{out} \times W_{out} + N \times C_{out}. \tag{3}$$

The CR is calculated as

$$\begin{aligned} CR &= \frac{Uncomressed\ Data}{Comressed\ Data}, \\ &= \frac{H_{out} \times W_{out} \times C_{out}}{CD}. \end{aligned} \tag{4}$$

Figure 10 (c) shows the decompression operations. The efficient PDM block selected by value prediction represents a non-repeated convolution calculation. The data decompression operations involve reading the EAV data based on the efficient PDM block and the first EAV address, then recombining the input data with the predicted SAV.

V. Hardware Implementation

A. Overall Architecture

In the previous sections, we introduce the proposed SEG recognition algorithm, two-stage value prediction approach, and channel-level SAM compression method. Next, we introduce the hardware implementation of the SBA. The overall architecture of our SBA consists of a 15.83KB Weight Buffer, an Activation Buffer, a Calculation Unit, and an Energy-Efficient Value Prediction and SAM Compression Unit (VPASCU), as shown in Figure 11. In addition, our

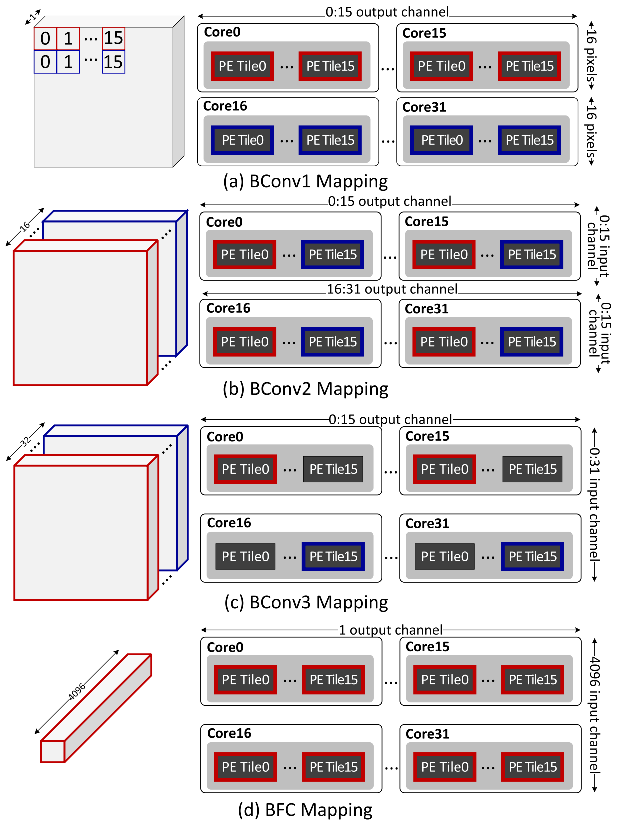

Fig. 13. Kernel-wise mapping process.

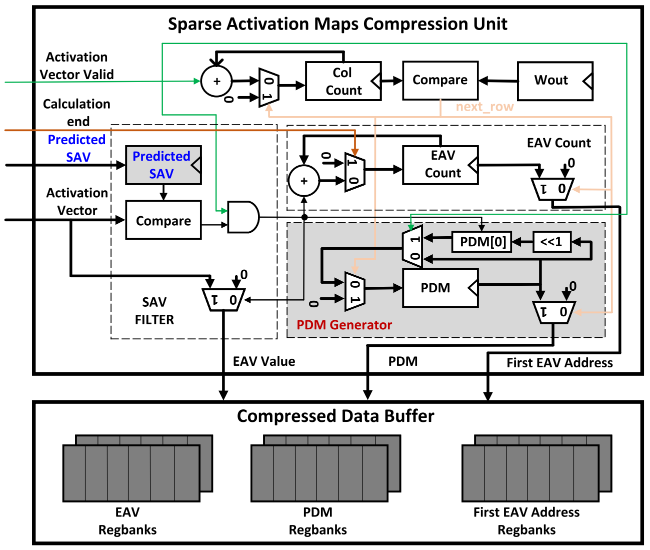

Fig. 14. The Sparse activation map compression unit.

SEG images are transmitted to the FPGA through the high-definition multimedia interface (HDMI). The weight data is loaded into the Weight Buffer, and the gesture classification is sent out through a universal asynchronous receiver/transmitter (UART).

B. Calculation Unit

The Calculation Unit is responsible for all calculation operations in sBNN, such as convolution, avgpooling, activation, comparison, and PReLU-BN (after BFC layer) calculations. It comprises a flexible Network-on-Chip (NoC), 32 Calculation Cores, a Hierarchical Adder Tree, and a Subsequent Processing Unit (SPU). Each Calculation Core contains 16 Processing Element (PE) Tiles which can perform 3×3 XNOR-Popcount operations. The Calculation Cores and Hierarchical Adder Tree only perform convolution calculations, while the SPU handles the other operations. The Calculation Unit can perform up to 4608 ($32 \times 16 \times 3 \times 3$) XNOR-Popcount operations at each cycle. The Predicted Buffer in the SPU stores the predicted SAV (both pre-pooling SAV and post-pooling SAV) obtained from offline value prediction. Each element in the output pre-pooling SAV of BConv1, BConv2, and BConv3 layers is 4-bit, 8-bit, and 9-bit respectively. The output post-pooling SAV of BConv1, BConv2, and BConv3 layers is 16-bit, 32-bit, and 64-bit respectively. The pre-pooling SAV will participate in the accumulation process of the pooling layer, and the post-pooling SAV is transmitted to VPASCU for SAV compression and decompression.

The computing characteristics (e.g., input channel, output channel, and spatial size of activation maps) of each layer in sBNN are different, which may lead to the underutilization of hardware resources. We propose flexible NoC and Hierarchical Adder Tree to efficiently implement the deployment of sBNN based on the kernel-wise mapping. The NoC is crucial in transmitting activation and weight data between Calculation Cores and their respective PE Tiles. The NoC supports unicast-multicast transmission in vertical and horizontal directions for each PE Tile, as shown in Figure 12. The four transmission modes available are Y-multicast (❶), Y-unicast (❷), X-multicast (❸), and X-unicast (❹). Based on unicast-multicast transmission ability, weight and input data can be efficiently transmitted to each PE Tile, increasing PE utilization and accelerating calculations.

The Hierarchical Adder Tree efficiently accumulates results from the PE Tiles and reuses computing resources. As shown in Figure 11, for the BConv1 layer, the output of each PE Tile is the BConv1 result, and the Hierarchical Adder Tree is not used. For the remaining layers, the final result is obtained by summing up the results of all input channels. Moreover, the input channels continue to increase as the number of layers increases, and the BConv result of the current layer may be a partial sum of the next layer. Especially in the BFC layer, the sum of results from PE Tiles used yields one BFC result. The Hierarchical Adder Tree adds up the results from most PE Tiles for the BFC layer.

Figure 13 illustrates the efficient kernel-wise mapping process for the sBNN model, achieving 100% and 88.9% PE utilization rates for the BConv and the BFC layers, respectively. The BConv1 layer requires 9 ($1 \times 3 \times 3$) XNOR-Popcount operations to produce a convolution result in each channel. The Calculation Cores within the same column perform computations for identical output channels, which allows 32 Calculation Cores to calculate 16 output channels simultaneously. In addition, each Calculation Core contains 16 PE Tiles, which can simultaneously handle 16 convolution

TABLE IV
DETAILED RESOURCE AND POWER

	LUT	FF	BRAM	DSP	Power
Calculation Unit	19396	11565	0	1	0.291W
Energy-Efficient VPASCU	86143	19256	0	0	0.372W
Weight Buffer & Activation Buffer	8774	4686	18.5	0	0.144W
Total	114313	35507	18.5	1	0.807W

calculations and generate 16 convolution results. The BConv2 layer requires 144 (16 × 3×3) XNOR-Popcount operations to produce a convolution result in each channel. 32 Calculation Cores simultaneously calculate 32 output channels and generate 32 convolution results. The BConv3 layer requires 288 (32 × 3×3) XNOR-Popcount operations to produce a convolution result in each channel. The Calculation Cores within the same column perform computations for identical output channels, which allows 32 Calculation Cores to calculate 16 output channels and generate 16 convolution results simultaneously. In the BConv3 layer, 64 output channels need four loops to complete the calculation. The BFC layer requires 4096 XNOR-Popcount operations to produce a BFC result. To achieve efficient calculations, we distribute the calculation of 4096 to 29 (4096/(16 × 3×3)) Calculation Cores.

C. Energy-Efficient VPASCU

The Energy-Efficient VPASCU is designed to enhance the energy efficiency of SBA through data storage reduction by SAM compression and decompression and repeated calculation skipping by leveraging the two-stage value prediction approach. It consists of an SAM Compression Unit, a Compressed Data Buffer, and a Prediction and Decompression Unit comprising the Value Prediction Unit, EAV Address Unit, and Activation Recovery Unit. The EAV Address Unit and Activation Recovery Unit make up the Decompressed Unit. Activation vectors generated by the SPU undergo further processing by the SAM Compression Unit.

The SAM Compression Unit plays a vital role in achieving energy efficiency in SBA by compressing the SAM with minimal resource consumption, thereby reducing the amount of stored data and power consumption of data reading and writing. It includes a SAV Filter, an EAV Counter, and a PDM Generator, as shown in Figure 14. Upon receiving an activation vector and its valid signal from the SPU, the SAV Filter discards the SAV value and saves the EAV value by comparing the activation vector with the predicted SAV. Then the EAV count is incremented by 1, and the corresponding PDM position is marked with a value of 1 by the PDM Generator. When receiving the last activation vector of a row, the SAM Compression Unit saves the EAV Counter and PDM values to the Compressed Data Buffer, where the EAV Counter is the next row's first address. Finally, the output EAV, the PDM, and the first EAV address are stored into EAV Regbanks, PDM Regbanks, and First EAV Address Regbanks respectively.

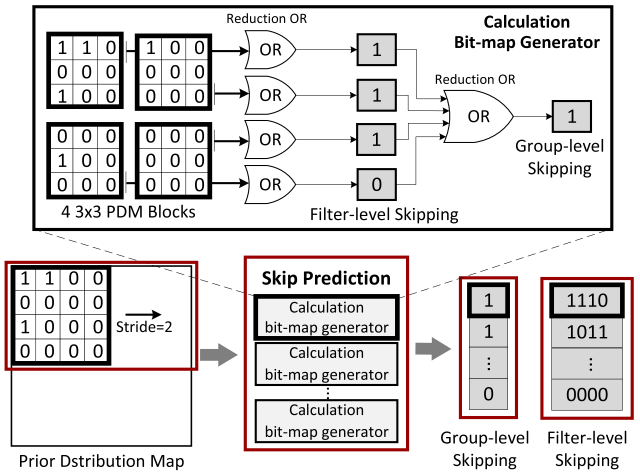

Fig. 15. Two tables for filter-level and group-level skipping are generated for value prediction.

When entering the next layer, the Value Prediction Unit efficiently accelerates convolution calculations and reduces computing power consumption by separating repeated calculations from non-repeated calculations and only performing the latter. The Value Prediction Unit performs a two-step operation: the first step generates two prediction tables for filter-level and group-level skipping. In contrast, the second step is online prediction flow which selects non-repeated calculations and skips repeated calculations based on these tables.

Figure 15 illustrates generating two prediction tables for group-level and filter-level skipping. Each Calculation Bit-map Generator first divides the 4 × 4 PDM block into four 3 × 3 PDM blocks. Then it performs Reduction OR operations on each 3 × 3 PDM block to form a filter-level skipping value. Finally, it performs Reduction OR functions on four filter-level skipping values to create the group-level skipping value. Two prediction tables are generated when computing all 4 × 4 PDM blocks in one row. Filter-level and group-level skipping prediction tables correspond to the calculations of the 3 × 3 PDM block and 4 × 4 PDM block, respectively.

The online prediction flow with two prediction tables is shown in Figure 16. If the value in the prediction table of group-level skipping is 0, all subsequent calculations can be skipped, and the results are predicted SAV. Otherwise, we reorganize the 3 × 3 PDM blocks based on the value in the filter-level skipping prediction table. The effective PDM blocks are sent to the Decompression Unit in order, and the decompressed data is sent to the Calculation Unit to generate the BConv results. Finally, the predicted pre-pooling SAV is added to the avgpooling results in the last cycle. The two prediction tables for five 4 × 4 PDM blocks, which have different skipping cases, are illustrated in Figure 17. The total calculation time for the five 4 × 4 PDM blocks is ten cycles, with ten calculation cycles saved.

The Decompression Unit is responsible for recovering the compressed data. When it receives an effective PDM block, the Address Generator Unit generates the addresses pointing to EAV and predicted SAV values based on the pointer and

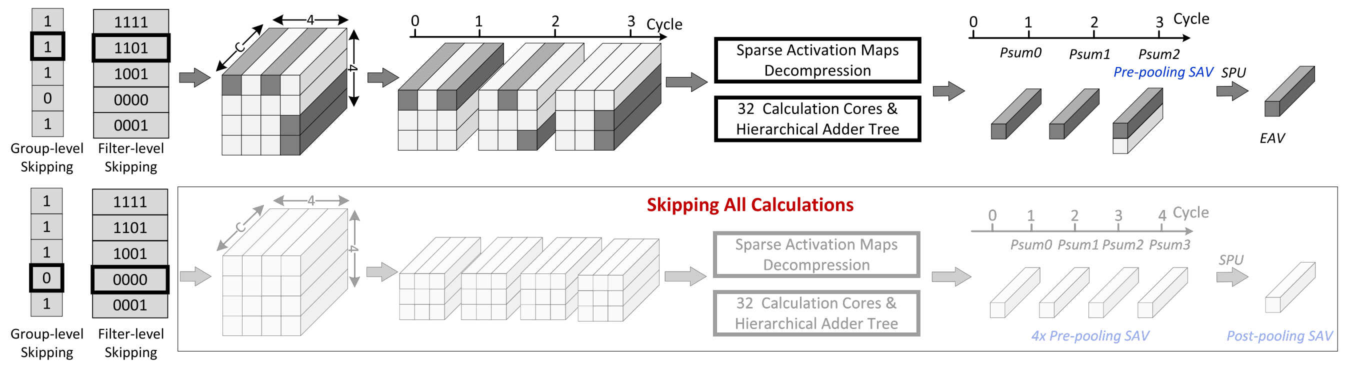

Fig. 16. Online prediction flow.

TABLE V
COMPRESSION RATE AND SPEEDUP WITH ENERGY-EFFICIENT APPROACHES

	Dataset	Type	Data amount and CR after compression					The cycles and speedup of value prediction				
			BConv1 (KB)	BConv2 (KB)	BConv3 (KB)	Total (KB)	CR (%)	BConv1 BFC	BConv2	BConv3	Total	Speedup
DBA	All Datasets	-	2	1	0.5	3.5	-	141 7×Classes	1039	1061	-	-
SBA	RGBDGES	LGE	0.648	0.837	0.5	1.985	1.76		807	1205	2230	1.03x
		SME	0.53	0.513	0.359	1.405	2.49		488	855	1561	1.48x
	RGBDASL	LGE	0.766	0.674	0.461	1.90	1.84		646	1071	2026	1.19x
		SME	0.414	0.482	0.367	1.263	2.78		435	864	1608	1.50x
	RGBASL	LGE	0.57	0.699	0.468	1.736	2.02		597	1075	1995	1.21x
		SME	0.301	0.375	0.367	1.043	3.36		344	835	1502	1.61x
	GRAYASL	LGE	0.836	0.701	0.5	2.03	1.72		663	1196	2182	1.11x
		SME	0.387	0.342	0.312	1.014	3.45		316	692	1331	1.83x

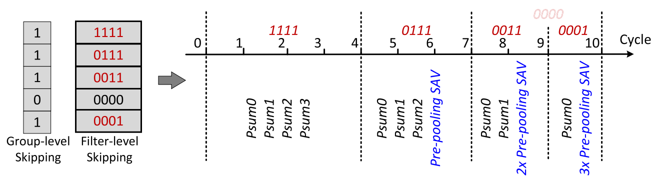

Fig. 17. The timeline of different calculations skipping cases.

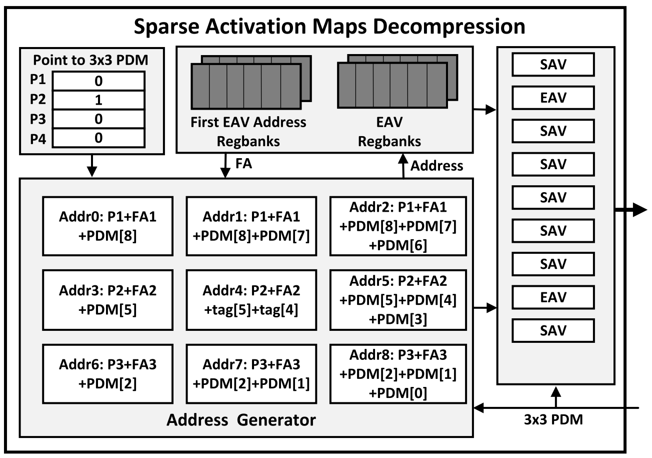
Sparse Activation Maps Decompression
Point to 3x3 PDM
P1 0
P2 1
P3 0
P4 0
First EAV Address Regbanks
EAV Regbanks
FA
Address
Addr0: P1+FA1 +PDM[8]
Addr1: P1+FA1 +PDM[8]+PDM[7]
Addr2: P1+FA1 +PDM[8]+PDM[7] +PDM[6]
Addr3: P2+FA2 +PDM[5]
Addr4: P2+FA2 +tag[5]+tag[4]
Addr5: P2+FA2 +PDM[5]+PDM[4] +PDM[3]
Addr6: P3+FA3 +PDM[2]
Addr7: P3+FA3 +PDM[2]+PDM[1]
Addr8: P3+FA3 +PDM[2]+PDM[1] +PDM[0]
Address Generator
SAV
EAV
SAV
SAV
SAV
SAV
SAV
EAV
SAV
3x3 PDM

Fig. 18. Decompression unit.

the first EAV address, as shown in Figure 18. The pointer indicates the leftmost position of the 3×3 PDM block. If the value in the block is 0, the predicted SAV is read. Otherwise,

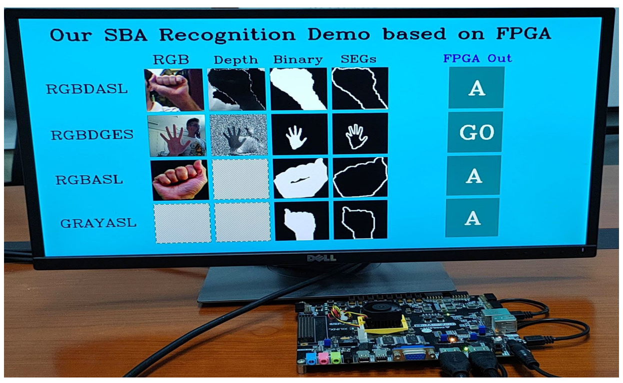

Fig. 19. The sparse BNN accelerator recognition demo with the Genesys2 board.

the EAV is read. In this way, the Activation Recovery Unit generates a $3 \times 3 \times C_{in}$ activation input. Finally, the activation input is sent to the Activation Buffer.

The bandwidths of EAV and SAV are different because SAV is just a special activation vector with a high repetition rate, and EAV is a collection of all other activation vectors. The EAV and SAV in BConv1, BConv2, and BConv3 layers are 16-bit, 32-bit, and 64-bit respectively. Redundant calculations can be skipped according to the PDM, and the remaining calculations need to be returned to the original 3×3 activation input. At this time, the corresponding EAV and SAV will be read to reassemble the original input activation block. Because the EAV is stored in register banks, an array of registers, it is

TABLE VI
COMPARISON WITH OTHER WORKS ON FPGA

Specifications	TCAS-I'2021 [37]	FPGA'2022 [38]	TCAS-I'2021 [7]	TCAS-I'2022 [39]	Our DBA	Our SBA
End-to-end	Yes		No			
FPGA	Arria10 SX660	ZCU102	ZCU102	ZCU102	Genesys2	Genesys2
Models	Resnet-18	Resnet-18	eCNN	eCNN	sBNN	sBNN
Bit-width	8-bit	95% W4A5 + 5% W8A5	8 dynamic FP	8 dynamic FP	1bit	1bit
Clock Frequency (MHz)	170	100	100	100	50	50
Latency (μs)	120000	4655	15940	14900	48.2	26.3 - 46.8
Performance (GOPS)	105.81	778.9	512	441	420.12	432.73 - 768.83
Power (W)	4.6	12.9	2.664	2.195	0.913	0.807
Energy Efficiency (GOPS/W)	19.41	60.4	192.19	200.96	460.15	536.22 - 952.70

supported to read and write 9 EAV at the same time. Therefore, increasing the data bit-width of the on-chip memory improves the data bandwidth and avoids spending too much time on data acquisition.

D. Evaluation

Table IV shows our SBA's resource utilization and power on the Genesys2 board for the RGBDASL dataset, with a clock frequency of 50 MHz. The entire SBA requires 15.83KB of memory to store all weight parameters. The PReLU-BN layer after the BFC layer can be considered multiple MAC operations, with the number of MACs equivalent to the number of gesture classifications. The power consumption of the SBA is 0.807W. The SBA recognition demo on the FPGA board is illustrated in Figure 19. We load different weight parameters to recognize the SEGs from gesture datasets. The SEGs are streamed real-time from the PC to the Genesys2 board through the HDMI. The classifications from the SBA are sent back to the PC through UART.

We have presented the percentage of SAV in each layer's SAM and described our energy-efficient optimization approaches. To demonstrate the effectiveness of our data compression and value prediction approaches, we provide the amount of data after compression and the computation cycles required for each layer. Table V shows the compression ratio achieved after compressing SAM and the speedup achieved with two-stage value prediction. We also develop a real-time dense BNN accelerator (DBA) without energy-efficient approaches for comparison. The total amount of uncompressed data in all ofmaps is 3.5KB. We select LGE and SME images from each dataset to obtain the best and worst compression ratios.

According to the data amount in Table V, the compression effect of the SBA decreases as we move from one layer to the next, with BConv1 exhibiting the most effective compression and BConv3 showing the worst. Compared to the DBA, the SBA achieves a CR of 1.72x - 3.45x. The calculation cycles and acceleration effect are shown on the right of Table V. The BConv1 calculation with a 100% PE utilization rate does not involve value prediction, and the BFC layer does not contain SAV. So there is no acceleration in the BConv1 and BFC layers. The BConv1 layer takes 141 cycles for each dataset. The cycles throughout the BFC layer are proportional to the number of classes, and each class requires seven cycles. In contrast, the BConv2 and BConv3 layers show significant acceleration effects due to the value prediction approach. For the BConv3 layer, the acceleration effect is less optimized since the value prediction approach increases cycles when the input activation maps exist a small number of SAV. We calculate the total computation cycles and obtain the speedup, revealing that our SBA achieves a 1.03x - 1.83x acceleration effect compared to the DBA. In addition, the pre-processing part only consumes 1994 μs running on the PC, which is lower than other works [7], [37], [38], [39] and can be reduced by hardware realization. The overall processing time is about 2020.3 - 2040.8μs.

In Table VI, the SBA outperforms the DBA in latency, power consumption, and energy efficiency thanks to the two-stage value prediction and activation maps compression and decompression approaches. Specifically, our SBA achieves a latency of 26.3 - 46.8 μs and a power consumption of 0.807 W, improving energy efficiency to 536.22 - 952.70 GOPS/W. Unlike end-to-end accelerators, our SBA preprocesses raw images on the PC before recognizing them on the hardware. Compared with end-to-end accelerators, non-end-to-end gesture recognition accelerators have a significant improvement in energy efficiency. By removing the background or noise from the image, pre-processing reduces calculations and parameters. During recognition, power consumption and delay are reduced, improving energy efficiency. Different CNN networks are mapped into FPGA for RGBDASL dataset in these works [7], [37], [38], [39], we compare the performance in terms of power, latency, and energy efficiency. As shown in Table VI, our SBA has the lowest latency and highest energy efficiency in the non-end-to-end accelerators.

VI. CONCLUSION

In this paper, we present a sparse BNN accelerator with value prediction for real-time edge gesture recognition on an FPGA board. We apply energy-efficient approaches for SBA, including two-stage value prediction and channel-level sparsity activation compression and decompression. The energy-efficient two-stage value prediction approach can skip repeated calculations to speed up sBNN computation and reduce power consumption. Furthermore, the channel-level sparse activation compression and decompression method can reduce the required data storage. Evaluations show that the sparse BNN accelerator can achieve state-of-the-art performance in latency and energy efficiency.

References

[1] S. Subburaj and S. Murugavalli, "Survey on sign language recognition in context of vision-based and deep learning," *Meas., Sensors*, vol. 23, Oct. 2022, Art. no. 100385.

[2] P. Goswami, S. Rao, S. Bharadwaj, and A. Nguyen, "Real-time multi-gesture recognition using 77 GHz FMCW MIMO single chip radar," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Jan. 2019, pp. 1–4.

[3] J. Pan, Y. Luo, Y. Li, C.-K. Tham, C.-H. Heng, and A. V.-Y. Thean, "A wireless multi-channel capacitive sensor system for efficient glove-based gesture recognition with AI at the edge," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 9, pp. 1624–1628, Sep. 2020.

[4] P. Haque, B. Das, and N. N. Kaspy, "Two-handed Bangla Sign Language recognition using principal component analysis (PCA) and KNN algorithm," in *Proc. Int. Conf. Electr., Comput. Commun. Eng. (ECCE)*, Feb. 2019, pp. 1–4.

[5] Y. Lu, V. L. Le, and T. T. Kim, "9.7 A 184 μW real-time hand-gesture recognition system with hybrid tiny classifiers for smart wearable devices," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 64, Feb. 2021, pp. 156–158.

[6] S. Sharma and S. Singh, "Vision-based hand gesture recognition using deep learning for the interpretation of sign language," *Expert Syst. Appl.*, vol. 182, Nov. 2021, Art. no. 115657.

[7] C.-C. Wang et al., "Real-time block-based embedded CNN for gesture classification on an FPGA," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 10, pp. 4182–4193, Oct. 2021.

[8] N. Pugeault and R. Bowden, "Spelling it out: Real-time ASL fingerspelling recognition," in *Proc. IEEE Int. Conf. Comput. Vis. Workshops (ICCV Workshops)*, Nov. 2011, pp. 1114–1119.

[9] P. Guo et al., "FBNA: A fully binarized neural network accelerator," in *Proc. 28th Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2018, pp. 51–513.

[10] M. Jaiswal, V. Sharma, A. Sharma, S. Saini, and R. Tomar, "An efficient binarized neural network for recognizing two hands Indian Sign Language gestures in real-time environment," in *Proc. IEEE 17th India Council Int. Conf. (INDICON)*, Dec. 2020, pp. 1–6.

[11] Y.-L. Zhang et al., "A 28 nm, 397 μW real-time dynamic gesture recognition chip based on RISC-V processor," *Microelectron. J.*, vol. 116, Oct. 2021, Art. no. 105219.

[12] A. Parashar et al., "SCNN: An accelerator for compressed-sparse convolutional neural networks," *SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 27–40, May 2017.

[13] S. Zhang et al., "Cambricon-X: An accelerator for sparse neural networks," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.

[14] M. H. Lipasti and J. P. Shen, "Exceeding the dataflow limit via value prediction," in *Proc. 29th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 1996, pp. 226–237.

[15] G. J. Grimes, "Digital data entry glove interface device," U.S. Patent 4 414 537, Nov. 8, 1983.

[16] K. S. Abhishek, L. C. F. Qubeley, and D. Ho, "Glove-based hand gesture recognition sign language translator using capacitive touch sensor," in *Proc. IEEE Int. Conf. Electron Devices Solid-State Circuits (EDSSC)*, Aug. 2016, pp. 334–337.

[17] Z. Zhang, Z. Tian, and M. Zhou, "Latern: Dynamic continuous hand gesture recognition using FMCW radar sensor," *IEEE Sensors J.*, vol. 18, no. 8, pp. 3278–3289, Apr. 2018.

[18] B. Garcia and S. A. Viesca, "Real-time American sign language recognition with convolutional neural networks," *Convolutional Neural Netw. Vis. Recognit.*, vol. 2, pp. 225–232, 2016. [Online]. Available: http://vision.stanford.edu/teaching/cs231n/reports/2016/pdfs/214_Report.pdf

[19] C. Szegedy et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[20] Z. Zhang, "Microsoft Kinect sensor and its effect," *IEEE Multimedia-Mag.*, vol. 19, no. 2, pp. 4–10, Feb. 2012.

[21] C. K. M. Lee, K. K. H. Ng, C.-H. Chen, H. C. W. Lau, S. Y. Chung, and T. Tsoi, "American sign language recognition and training method with recurrent neural network," *Expert Syst. Appl.*, vol. 167, Apr. 2021, Art. no. 114403.

[22] S.-H. Yang, W.-R. Chen, W.-J. Huang, and Y.-P. Chen, "DDaNet: Dual-path depth-aware attention network for fingerspelling recognition using RGB-D images," *IEEE Access*, vol. 9, pp. 7306–7322, 2021.

[23] H. Huang, Y. Chong, C. Nie, and S. Pan, "Hand gesture recognition with skin detection and deep learning method," *J. Phys., Conf. Ser.*, vol. 1213, no. 2, Jun. 2019, Art. no. 022001.

[24] A. Abdulhussein and F. Raheem, "Hand gesture recognition of static letters American Sign Language (ASL) using deep learning," *Eng. Technol. J.*, vol. 38, no. 6, pp. 926–937, Jun. 2020.

[25] A. Memo, L. Minto, and P. Zanuttigh, "Exploiting silhouette descriptors and synthetic data for hand gesture recognition," in *Proc. Smart Tools Apps Comput. Graph. (STAG)*. Eurographics Association, 2015, pp. 15–23, doi: 10.2312/stag.20151288.

[26] A. Memo and P. Zanuttigh, "Head-mounted gesture controlled interface for human–computer interaction," *Multimedia Tools Appl.*, vol. 77, no. 1, pp. 27–53, Jan. 2018.

[27] A. L. C. Barczak, N. H. Reyes, M. Abastillas, A. Piccio, and T. Susnjak, "A new 2D static hand gesture colour image dataset for ASL gestures," *Res. Lett. Inf. Math. Sci.*, vol. 15, pp. 12–20, 2011. [Online]. Available: https://core.ac.uk/download/pdf/148639234.pdf

[28] (2021). *Rolandomr*. [Online]. Available: https://www.kaggle.com/rolandomr/american-sign-language-recogntion-asl/metadata

[29] T. Geng et al., "LP-BNN: Ultra-low-Latency BNN inference with layer parallelism," in *Proc. IEEE 30th Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, vol. 2160-052X, Jul. 2019, pp. 9–16.

[30] K. O. Rodríguez and G. C. Chávez, "Finger spelling recognition from RGB-D information using kernel descriptor," in *Proc. 26th Conf. Graph., Patterns Images*, Aug. 2013, pp. 1–7.

[31] R. G. Rajan and M. J. Leo, "American Sign Language alphabets recognition using hand crafted and deep learning features," in *Proc. Int. Conf. Inventive Comput. Technol. (ICICT)*, Feb. 2020, pp. 430–434.

[32] N. Singla, M. Taneja, N. Goyal, and R. Jindal, "Feature fusion and multi-stream CNNs for ScaleAdaptive multimodal sign language recognition," in *Proc. 9th Int. Conf. Adv. Comput. Commun. Syst. (ICACCS)*, vol. 1, Mar. 2023, pp. 1266–1273.

[33] Z. Han-Wen, H. Ying, Z. Yong-Jia, and W. Cheng-Yu, "Fingerspelling identification for American sign language based on Resnet-18," *Int. J. Adv. Netw. Appl.*, vol. 13, no. 1, pp. 4816–4820, 2021.

[34] F. Gabbay and A. Mendelson, *Speculative Execution Based on Value Prediction*. Princeton, NJ, USA: Citeseer, 1996.

[35] G. Shomron and U. Weiser, "Spatial correlation and value prediction in convolutional neural networks," *IEEE Comput. Archit. Lett.*, vol. 18, no. 1, pp. 10–13, Jan. 2019.

[36] M. Pligouroudis, R. A. G. Nuno, and T. Kazmierski, "Modified compressed sparse row format for accelerated FPGA-based sparse matrix multiplication," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Oct. 2020, pp. 1–5.

[37] X. Xie, J. Lin, Z. Wang, and J. Wei, "An efficient and flexible accelerator design for sparse convolutional neural networks," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 7, pp. 2936–2949, Jul. 2021.

[38] M. Sun et al., "FILM-QNN: Efficient FPGA acceleration of deep neural networks with intra-layer, mixed-precision quantization," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb. 2022, pp. 134–145.

[39] C.-T. Chiu et al., "Chaos LiDAR based RGB-D face classification system with embedded CNN accelerator on FPGAs," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 12, pp. 4847–4859, Dec. 2022.

Yongliang Zhang received the B.S. and M.S. degrees in microelectronics from the Hefei University of Technology, Anhui, China. He is currently pursuing the Ph.D. degree with the School of Microelectronics, Fudan University, China. His current research interests include human–computer interaction chip technology, energy-efficient gesture accelerators, and computer vision.

Yitong Rong received the B.E. degree in integrated circuit design and integrated systems from Xidian University, Xi'an, China, in 2020. He is currently pursuing the Ph.D. degree in integrated circuit and system design with Fudan University, China. His current research interests include domain-specific architectures, graph deep learning, and computer vision.

Xuyang Duan received the B.S. degree from Fudan University, Shanghai, China, in 2021, where he is currently pursuing the M.S. degree with the State Key Laboratory of Integrated Chips and Systems, School of Microelectronics. His current research interests include energy-efficient VLSI design, domain-specific architectures, and computer vision.

Zhen Yang received the M.S. degree in microelectronics and solid electronics from Hunan University, Changsha, China, in 2005. He is currently pursuing the Ph.D. degree with the School of Microelectronics, Fudan University, China. His current research interests include computer architecture, domain-specific processors, and low power technology.

Qiang Li received the B.S. degree in electronics science and technology from Xidian University, Xi'an, China, in 2019. He is currently pursuing the Ph.D. degree with the School of Microelectronics, Fudan University, China. His current research interests include microarchitecture design methodology, system-level power/thermal analysis and management, and low-power hardware design for deep-learning.

Ziyu Guo (Member, IEEE) received the B.E. and Ph.D. degrees from the School of Information Science and Engineering, Southeast University, Nanjing, China, in 2011 and 2016, respectively. From 2013 to 2015, he was a Visiting Student with Columbia University, New York, NY, USA. From 2019 to 2022, he was a Post-Doctoral Researcher with the State Key Laboratory of ASIC and System, Fudan University, Shanghai, China. He is currently an Assistant Professor with the Department of Electrical Engineering, Fudan University. His research interests include millimeter-wave communication and VLSI design for digital signal processing.

Xu Cheng (Member, IEEE) received the B.S. and M.S. degrees in electronics engineering from Fudan University, China, in 1999 and 2002, respectively, and the Ph.D. degree from University College Cork, Ireland, in 2007. From 2007 to 2009, he was with Cypress Semiconductor, Ireland. In 2009, he joined Fudan University, where he is currently an Associate Professor. His current research interests include energy-efficient analog mixed-signal design and analog CAD.

Xiaoyang Zeng (Member, IEEE) received the B.Sc. degree from Xiangtan University, Xiangtan, China, in 1996, and the Ph.D. degree (Hons.) from the Changchun Institute of Optics, Fine Mechanics and Physics, Chinese Academy of Sciences, Beijing, China, in 2001. In 2001, he joined Fudan University, where he was a Post-Doctoral Researcher from March 2001 to February 2003. Since 2003, he has been with Fudan University, as a Faculty Member, where he is currently a Chair Professor and the Executing Director of the State Key Laboratory of State Key Laboratory of Integrated Chips and Systems. He has published more than 200 articles in international journals and conferences, such as IEEE ISSCC, IEEE JOURNAL OF SOLID-STATE CIRCUITS, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, IEEE VLSI Symposia, IEEE CICC, IEEE ESSCIRC, IEEE ASP-DAC, and IEEE A-SSCC. He has applied for more than 120 patents. His research interests include information security chips, base-band processing technologies for wireless communication, mixed-signal IC designs, and ultra-low power IC methodology.

Jun Han (Member, IEEE) received the B.S. degree from Xidian University, Shaanxi, China, in 2000, and the Ph.D. degree in microelectronics from Fudan University, Shanghai, China, in 2006. In July 2006, he joined Fudan University as an Assistant Professor, where he is currently a Full Professor with the State Key Laboratory of Integrated Chips and Systems. His current research interests include domain-specific processors and systems for digital signal processing, machine learning, and human–computer interaction.