

## RESEARCH ARTICLE

# A High-Performance Neural Network SoC for End-to-End Speaker Verification

**TSUNG-HAN TSAI** , (Senior Member, IEEE), AND MENG-JUI CHIANG

Department of Electrical Engineering, National Central University, Taoyuan 32001, Taiwan

Corresponding author: Tsung-Han Tsai (han@ee.ncu.edu.tw)

This work was supported by the Ministry of Science and Technology, Taiwan, under Grant MOST 111-2221-E-008-089-MY3.

**ABSTRACT** The use of the neural network to recognize a speaker's identity from their speech sounds has become popular in the last few years. Among these methods, the x-vector extractor, which is based on time-delay neural networks (TDNN), performs better in noise-canceling and generally achieves higher accuracy compared to previous methods such as the Gaussian mixture model (GMM) and the support vector machines (SVM). This paper presents a system-on-chip (SoC) composed of a RISC-V CPU and a neural network accelerator module for x-vector-based speaker verification (SV). To ensure real-time latency and enable the implementation of the system on edge devices, this work employs three steps for processing x-vector including size reduction, pruning, and compression. We are dedicated to optimizing the data flow with sparsity. Compared with the conventional sparse matrix compression method compressed sparse row (CSR), we propose the binary pointer compressed sparse row (BPCSR) method which significantly improves the latency and avoids the load balancing issue in each PE. We further design the neural network accelerator module that stores the compressed parameters and computes the x-vector extractor while the RISC-V CPU processes the rest of the calculations such as feature extraction and the classifier. The system was tested on the VoxCeleb dataset, containing 1251 test speakers, and achieved over 95% accuracy. Lastly, we synthesized the chip with TSMC 90 nm technology. It presents 15.5 mm<sup>2</sup> in the area and 97.88 mW for real-time identification.

**INDEX TERMS** Speaker verification (SV), speaker identification, x-vector, RISC-V, system-on-chip (SoC).


## I. INTRODUCTION

The uniqueness of each person's sound has been commonly used as biometric identification in recent years. Among biometric identification with sound, speaker recognition is one of the leading technologies. There are two major branches of speaker recognition. If the speaker claims to be of a certain identity and the voice is used to verify this claim, this is called verification or authentication. On the other hand, identification is the task of determining an unknown speaker's identity. In most cases, both speaker verification and speaker identification algorithms are similar.

A text-independent speaker verification (SV) system can identify the person without regard to the content of the speech. There are several applications in daily life. One of

the famous and successful examples is the bank's voice-activated customer service [1]. With the SV neural network automatically identifying the person, the low reliability and high repetitiveness of manual checking are no longer necessary. It can be applied to some applications, such as the voice assistants on the cellphone or television to distinguish different users for customized responses.

The recognition accuracy is always a challenge to these applications. In the past, the Gaussian mixed model (GMM) [2], [3], [4] and i-vector [5], [6], [7] were the robust methods for SV. Although the conventional iteration algorithm achieves adequate accuracy, it has been proved that they perform poor recognition accuracy in noisy or short-duration utterances. The d-vector [8] exploits the neural network to address the reduced accuracy in non-ideal environments, but it only performs comparable results to the i-vector. Finally, the x-vector [9], which is based on the time-delay neural

The associate editor coordinating the review of this manuscript and approving it for publication was Thomas Canhao Xu .

network (TDNN) [10], shows a significant improvement over the previous methods. Other methods like Recurrent neural networks (RNNs) slightly improve the recognition accuracy but contain more parameters and increase the computational complexity [11], [12], [13].

In terms of hardware design, most of the current speaker verification chips are based on traditional architecture such as GMM [14], [15] and SVM [16]. Although they have good results in power consumption, they are more difficult to apply in applications with a lower tolerance for error verification such as LOCKs. Some research [17], [18] shows that the x-vector has better accuracy than traditional mathematical models, especially in noisy environments, with a large number of enrolled speakers, or with speech of brief duration. Besides, the data flow in the x-vector is relatively simple compared with other neural network-based algorithms. Thus, we use the x-vector as the base model in this work. We also compressed the x-vector extractor because even though the x-vector extractor has outstanding performance, it comes with a much higher computational effort.

Apart from the accuracy, the power budget and the calculation latency also have strict limitations in implementing the SV system on edge devices. External memory access such as reading DRAM causes high power consumption and latency. Therefore, several recent research [19], [20] tends to store all the network parameters on the chip to avoid the above issue. Since the huge storage data may come with the increasing area, eliminating the unnecessary parameters in the network is needed. Currently, pruning can generate sparsity in the weight matrix in the neural network. It is commonly used for Long Short-Term Memory (LSTM) [21], [22], CNN, and MLP where it originally had a relatively large amount of parameters. Additionally, the storage format is considered to reduce memory size. Compressed Sparse Row (CSR) and Compressed Sparse Column (CSC) [23] are popular formats for encoding sparse matrices, designed to efficiently store and process matrices with a large number of zero values by only storing non-zero elements and their indices. However, while CSR and CSC reduce storage requirements, CSC can lead to load-balancing issues among processing elements (PEs) [24] due to its column-wise storage nature, which may result in uneven computational loads.

This paper presents an x-vector-based speaker verification SoC and implements it as an application-specific integrated circuit (ASIC) chip. To present a low power, low latency, and high accuracy SV SoC, three hierarchical challenges are (1) network compression, (2) efficient dataflow, and (3) chip architecture. At the algorithm level, we propose the storage format, BPCSR, to pre-arrange the processing elements (PEs). It can address the load-balancing issue and further improve the decoding and encoding process. At the system level, we make the design more flexible and support the general peripheral IPs such as UART or GPIO for back-end devices, the Andes N25F CPU [25] is included in this design. The CPU also supports other parts

of the work except for neural network computations, such as feature extraction and scoring. Besides, the neural network accelerator module stores all the network parameters and processes the sparse x-vector extractor in parallel and efficiently. We implemented the SV system as an ASIC instead of a Field Programmable Gate Array (FPGA) [26] because ASIC has the advantage of lower power consumption. Also, the whole SV SoC can easily integrate with other chips or systems.

This end-to-end SV SoC was implemented in the TSMC 90-nm process. Our proposed chip reduces nearly 85% of computations compared with the original x-vector, and it also maintains over 95.6% accuracy on the VoxCeleb dataset [27], [28], which is sufficient to meet most applications in life. The number of test speakers is up to 1251, which is nearly six times higher than the majority of prior works. In summary, this paper presents the following contributions:

- We provide a comprehensive design process from the software algorithm to hardware implementation for a complete SV system.
- We design an end-to-end x-vector-based SoC combined with a RISC-V CPU and neural network module for large-scale speaker verification.
- We significantly reduce the size of the x-vector neural network. The parameters were reduced to only 660K with an ignorable accuracy drop evaluated on the VoxCeleb dataset.
- We propose the sparse matrix compression method BPCSR with specific processing elements for fast decoding and PE load-balancing.

The rest of the paper is organized as follows. Section II introduces the related works and background. Section III presents the proposed speaker verification system. Section IV discusses the x-vector neural network on optimization and allocation. Section V presents the proposed SoC hardware architecture. Section VI shows the experimental results and Section VII gives the conclusions.

## II. RELATED WORKS

In this section, we will discuss related research on algorithms and hardware aspects of speaker verification. Since our chip design contains an embedded CPU and neural network computation, speech-related SoC design is discussed in subsections II-C.

### A. SPEAKER VERIFICATION ALGORITHM

In the past, GMM was used for speaker recognition in 1995 by Reynolds and Rose [2]. The speech data is first transformed into the spectrum. The GMM-based speaker model is obtained by expectation-maximization (EM) algorithm with training data. Finally, the log-likely score identifies the speakers. Based on GMM, a novel algorithm, i-vector was proposed by Dehak et.al. [4] in 2011. They improved the problem that GMM contains the speaker information along with the speech context information. Channel compensation is a technique that significantly enhances the performance of

i-vector-based text-independent speaker verification, making it a widely adopted and mainstream algorithm in the past.

Recently, the neural network methods in speaker recognition have been proven they have better performance than traditional algorithms, especially in serious background interference. In 2014, a famous neural network, d-vector [8], was proposed. The d-vector is trained to classify speakers with frame-level acoustic features. However, it only performed a similar result with the i-vector. In 2018, a robust model, x-vector, was proposed by Snyder et. al [9]. The TDNN layers in the x-vector model efficiently extract the speaker's information with time relationships. It also significantly improves the speaker verification result. Other advanced models such as RNN [11], [12], [13] help the ability for deeper extracting the speaker information. Although the recognition result has a slight improvement in more complex recognition environments, the computation cost also increases.

### B. SPEAKER VERIFICATION HARDWARE

Table 1 summarizes the hardware implementations of the speaker verification task. References [29], [30], and [31] used SVM and [32] used the linear predictive coding method to implement the SV work on FPGA. Although these methods have lower computational complexity and less memory usage, the less reliable verification results limit the application environment of this system. In [26], a mainstream neural network model, an x-vector extractor, is implemented on FPGA to improve verification accuracy. However, the SV system is implemented on a powerful FPGA, VCU118, and uses a huge part of the computation resources without optimizing the model, making the system inefficient. The overall power consumption is over 36W.

References [15], [16], and [33] are the ASIC works for speaker verification. In [16], they exploit multiple processing units in the Gaussian kernel unit to accelerate the verification process. The power consumption has significant improvement compared with other FPGA works. However, this design is not fully end-to-end, and the verification accuracy is worse than most neural network-based SV systems. In [15], an extremely low-power GMM-based end-to-end SV system was proposed. The power consumption can achieve the  $\mu$ W level, which is good for mobile devices. However, for some applications that require rigorous verification such as voice-activated door locks, GMM is not suitable since it is more unstable and more sensitive to noise. Also, their work was only tested on a clean dataset, and they trialed only 168 samples, raising concerns about its overall persuasiveness. In [33], they combined the keyword spotting and SV features by binarized weight network, which reuses 93.2% of computations. Also, they proposed a secondary threshold classification that improves the verification accuracy.

### C. SPEECH PROCESSING SoC

Based on these efficient SV methods, a system-level SoC combined with embedded CPU and dedicated hardware can

be involved to perform an end-product system. Because the pre-processing contains many non-linear calculations such as FFT and logarithm, the CPU can provide more flexibility to exploit it for speech pre-processing. In [34], a speech recognition SoC combining a dual-core ARM Cortex-A53 and a neural network accelerator module is proposed. The A53 CPU processes the pre-processing such as framing and FFT to obtain computational efficiency. Instead of ARM architecture, in [35], they used a RISC-V-based Andes N25F CPU with a simple processing unit for dysarthric voice conversion of stroke patients.

RISC-V is an instruction set architecture, which has risen in recent years. In contrast to ARM, RISC-V is an open-source instruction set architecture that does not require additional licensing fees and has good scalability. Additionally, it allows designers to modify or extend the architecture to meet different requirements. Compared to ARM, the RISC-V instruction set contains only basic instructions, i.e., the hardware architecture can get better area and power consumption.

### III. PROPOSED SPEAKER VERIFICATION SYSTEM

This section shows the proposed speaker verification system with its algorithm discussion. The neural network module plays a key role in the whole system. We design the neural network module in this section and propose some techniques to enhance the performance in the next section.

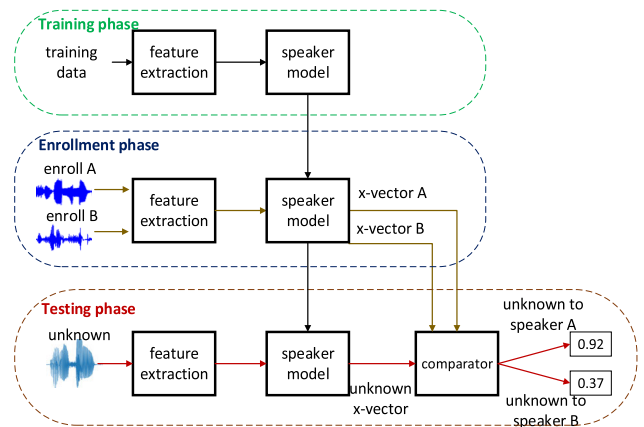


FIGURE 1. Top level speaker verification flow and example.

#### A. OVERALL SPEAKER VERIFICATION

The verification steps can mainly be divided into three phases: the training phase, the enrollment phase, and the testing phase. Firstly, the training phase is to train an SV model to extract the information from the speakers. Note that feature extraction occurs at each of the three stages to extract information from the source. Secondly, in the enrollment phase, the enrolled utterances are converted into x-vectors by processing the well-trained speaker model. Lastly, in the test phase, after the unknown utterance is converted into the x-vector, the comparator scores the unknown x-vector

TABLE 1. The summary of speaker recognition related works.

Type	Algorithm type	Algorithm	Platform	power	accuracy	Reference
Speaker verification	classifier	SVM	FPGA	High	Low	[29], [30], [31]
			ASIC	Low	Low	[16]
		GMM	ASIC	Very low	High	[15]
		LPC	FPGA	High	-	[32]
	Neural network	BWN	ASIC	Very low	High	[33]
		x-vector	FPGA	High	High	[26]
ASIC	Low		High	This work		

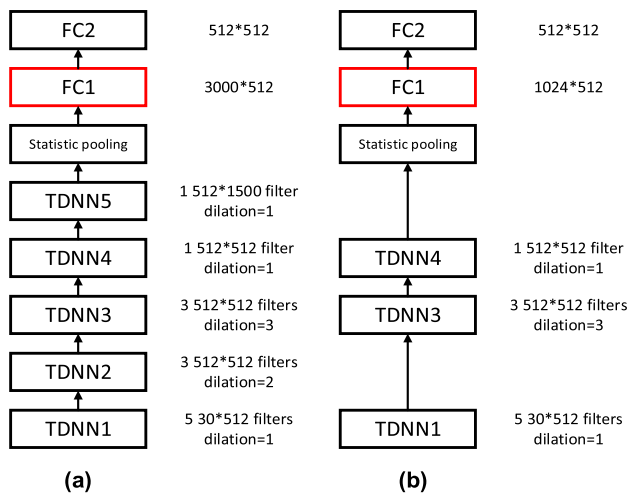


FIGURE 2. The model architectures and their configurations. (a) The original x-vector. (b) The x-vector with size reduction.

to each enrolled x-vector. As long as the score exceeds the threshold value, or a score is higher than others, the unknown utterance is considered as belonging to the corresponding enrolled speaker. As shown in Fig. 1, the SV system tries to find the source of an unknown voice, and there are two enrolled speakers, enroll A and enroll B. The trained speaker model extracts the x-vectors of the unknown voice, speaker A, and speaker B. The comparator compares the unknown x-vector to the x-vectors of enroll A and enroll B, respectively. The similarity between the unknown voice and enroll A is 0.92 (higher than enroll B’s score), i.e., the unknown voice is considered to enroll A’s voice.

The verification performance is directly affected by mainly three steps: feature extraction, speaker model, and the comparator. Mel-scale Frequency Cepstral Coefficients (MFCC) and Filter banks (FBANK) are two of the widely used feature extract methods. This step converts digital signals from time domain sequential signals to frequency domain spectrum. The speaker model typically consists of the neural network [8], [9], [10], [11], [12], [13] or mathematical model [2], [3], [4], [5], [6], [7]. This step is to compute the unique characteristics of the speaker through the speaker model. Finally, the cosine score [36] or Probabilistic Linear

Discriminant Analysis (PLDA) [37] compares each feature vector’s similarity. Typically, PLDA achieves higher accuracy than the cosine score, but it contains complex computations.

B. FEATURE EXTRACTION

The main purpose of feature extraction is to convert the incoming speech from the time domain to the frequency domain. MFCC and FBANK are the commonly used methods in the field of speech area. Since FBANK is less computationally intensive and the accuracy rate [41] is similar to MFCC or even slightly better, we used FBANK to capture the features in this work. There are five main steps: framing, windowing, fast Fourier transform (FFT), Mel filter bank, and logarithm. The spectrum output is the FBANK feature.

C. X-VECTOR

X-vector is one of the current mainstream baseline model frameworks in speaker-related applications [42]. The speaker characteristic is extracted by the first fully connected layer during the inference. It allows any length input and transforms it into a fixed-length feature vector. Also, the x-vector shows outstanding performance but lower complexity of computations [9] than other commonly used methods such as RNNs or deep CNNs. Therefore, our speaker model is based on an x-vector. It only consists of five TDNN layers, a statistic pooling layer, and two fully connected layers shown in Fig. 2 (a).

TDNN performs well in dynamic adaptation to time-domain features and contains fewer parameters. Traditional deep neural networks present input layers connected one by one with the hidden layer. Instead, TDNN processes as 1-D CNN with dilation calculations, i.e., the features of the hidden layer are related not only to the input at the current moment but also to the input at the future and past moment.

The network architecture and dilation details in this work are shown in Fig. 2. After the feature extraction, the input feature dimension is transformed into  $30 \times 512$  where 30 refers to the Mel-coefficient feature and 512 represents the number of frames. The different dilation represents the number of intervals between kernels. The following TDNN layers capture the key messages from the input feature with



**TABLE 2.** The error rate and parameters within different removal.

Remove layer	Error rate	Paras	Reduce rate
x-vector baseline	3.35%	4.2M	0%
TDNN 5	3.69%	2.6M	38%
TDNN 2, 3	4.26%	2.6M	39%
TDNN 4, 5	4.20%	2.1M	50%
TDNN 2, 5	4.23%	1.6M	62%
TDNN 2, 4, 5	4.60%	1.3M	69%
TDNN 2, 3, 5	4.86%	0.8M	81%

dilation respectively. These filters are used to capture features at different time delays when processing sequential data.

#### D. PLDA

Probability Linear Discriminant Analysis (PLDA) [37] is a scoring method based on a probabilistic model. PLDA's similarity scoring achieves good accuracy under challenging conditions and significantly addresses the problem of high recognition difficulty and reduced accuracy. The mathematical models should be well-trained. All the enrolled and test the x-vector process the transformation, and the final log-likelihood ratio can be seen as the similarity scores.

### IV. X-VECTOR NETWORK OPTIMIZATION AND ALLOCATION

For neural networks with massive parameters such as x-vector. Before reducing the overall cost of hardware computing, an efficient and compact dataflow for the compressed model is also critical such as downing or pruning. In this section, we will discuss the compression in the x-vector and its data pre-allocation.

#### A. MODEL DOWNSIZING

Undeniably, the convolutional layer contributes significantly to the computation in the neural network, and eliminating parameters (20% to 80% depending on the network) does not significantly affect accuracy. Therefore, we first remove the convolutional layers which contain a large number of parameters.

The removed TDNN layers and their corresponding information are shown in Table 2. We simulated the SV accuracy on the VoxCeleb dataset with some different TDNN layer removal. TDNN5 can be considered a critical layer because it contains the most oversized filter in the whole network. Additionally, the statistic pooling layer concatenates the mean and standard deviation of TDNN5 output as the first fully connected layer's input. Namely, TDNN5 indirectly impacts the size of the FC1 layer. Apart from TDNN5, TDNN2, and TDNN3 also significantly impact the network size because they contain three 512 by 512 filters. Finally, we remove TDNN5 and TDNN2 because TDNN2 presents less dilation, which incurs more computations than TDNN3. By doing so, the error rate maintains nearly 4.2% but eliminates over 62% of parameters. The simplified x-vector architecture is shown in Fig. 2 (b). After the

downsizing of the network, it shows a higher computational efficiency.

#### B. PRUNING

To further reduce the parameters of neural networks, we prune the network model. The goal of model pruning is to keep only the critical weights and parameters in each filter and maintain the same performance. The ratio of eliminated weights and parameters is referred to as the pruning rate.

There are two kinds of pruning: unstructured pruning and structured pruning. Unstructured pruning is considered element-wise pruning. It does not impose any constraints on parameters which are needed to be removed. Thus, the zero values are irregularly distributed in the sparse matrix. This method typically shows a higher sparsity and is easy to implement. Instead, the structured pruning removes particular portions of the continuous parameters such as whole columns or whole filters. The non-zero values will have a particular distribution in the output sparse matrix.

In this work, we process unstructured pruning with the L1 norm. L1 norm pruning is a commonly used pruning strategy, and the equation is expressed as follows:

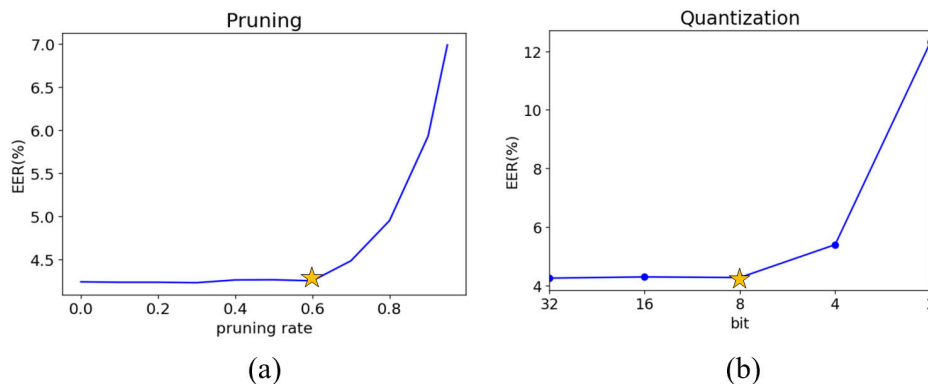
$$w'_{ij} = \begin{cases} w_{ij}, & \text{if } |w_{ij}| > \text{Threshold} \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

where  $w_{i,j}$  is the weight in filter row  $i$  and column  $j$ . The output result  $w'_{i,j}$  is based on whether the absolute input value is greater than the Threshold. Once the absolute input value is less than the threshold, the output weight will be set to zero. The reason for removing weights with small absolute values is that they are usually less influential in neural networks. Most of the layers in a neural network consist of linear operations. Weights with smaller absolute values have less impact on the output to the next layer. This method allows the pruned result to be less different from the original output.

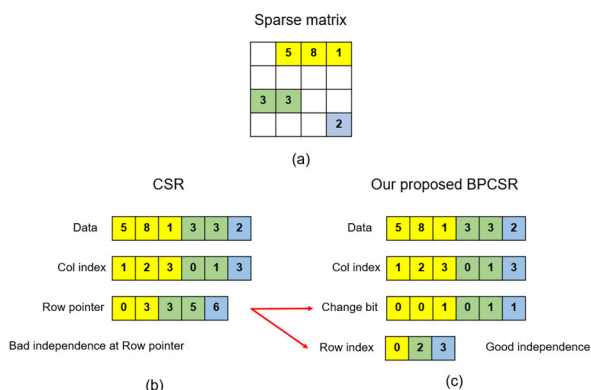
Fig 3(a) shows the error rate at different pruning rates, which represents the percentage of a filter's weight that is pruned. We follow the pruning procedure in [38]. They mainly show three pruning steps: training connectivity, prune connectivity, and training weights. The simulation has no obvious error rate increase at a pruning rate of 60% (only an increase of 0.012%). Instead of a particular threshold, we pruned 60 percent of weights in each layer including TDNN and fully-connected layers. The total number of parameters dropped from 1.6M to 640K, which shows a good balance between accuracy and quantity of parameters.

#### C. QUANTIZATION

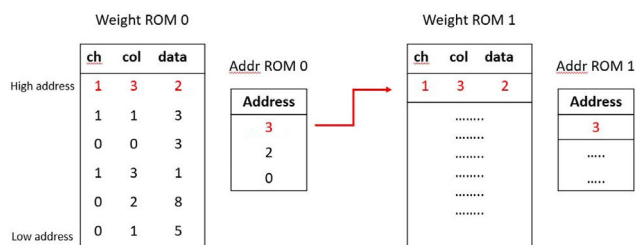
It is critical to quantize the data to a fixed point format before implementing the hardware since the floating point calculation is inefficient in area and power consumption. Usually, the temporary data requires more precision because the error caused by precision accumulates during the computation, especially in the models with large-size filters. Thus, the temporary data is set to a 24-bit (1-bit sign bit, 11-bit



**FIGURE 3. (a)The size-reduced x-vector error rate at different pruning rate. (b)The size-reduced x-vector error rate in different weight precision.**



**FIGURE 4. (a)The example of the sparse matrix. (b) The CSR storage format. (c) The proposed BPCSR storage format.**



**FIGURE 5. The BPCSR PE load-balancing illustration.**

exponent, and 12-bit fraction) fixed point. We simulated the post-training quantization error rate in different weight precisions as in Fig. 3(b). The weight in 8-bit fixed-point (1-bit sign bit, 0-bit exponent, and 7-bit fraction) and input 24-bit fixed-point data resulted in the ignorable error rate increment.

**D. LOAD-BALANCING ISSUE IN CSR/ PROPOSED BPCSR**

As the weight matrix becomes sparse after weight pruning, the matrix contains a large number of zero elements. To reduce overall storage and frequency of memory access, most of the sparse matrix is stored in special storage formats, which avoids all the zero elements occupying the storage

space. Conventional methods such as CSC/CSR can address these issues. However, CSC and CSR are difficult to achieve good computing efficiency in hardware implementation since the PE load-balancing issue occurs.

To store a sparse matrix in CSR, three necessary arrays are floating-point array data and two integer arrays (Col\_index and Row pointer). For example, a sparse matrix, Fig. 4(a), is stored in the CSR Fig. 4(b). The data array holds the non-zero elements of the matrix and is sized as the number of non-zero elements of the sparse matrix. Col\_index holds the column indexes of the elements in the data array and is sized the same as the data array. The Row\_pointer array is used to keep track of the starting index of each row’s non-zero elements in the data array. Each entry in the Row\_pointer array represents the position in the data array where the non-zero elements of the corresponding row begin. The size of the Row\_pointer array is one more than the number of rows in the matrix, with the last entry indicating the total number of non-zero elements.

The position of each element in the sparse matrix can be represented in these three arrays. However, achieving perfect performance with CSR can be challenging on most neural network accelerators. Initially, the allocation of computations from the dense matrix to PEs is usually based on the rows or columns in the matrix, e.g., the first and second rows are allocated to PE0, and the third and fourth rows are allocated to PE1. Each PE is allocated roughly equal work. When the matrix becomes sparse, the non-zero elements are not regularly distributed in each row. The larger the matrix, the greater the disparity in distribution. Thus it results in the computations in the matrix not being evenly allocated to each PE. Every PE has to wait for the PE with the most computational work to complete its work. Typically, to address the PE load-balancing issue, the PE with more work can assign some of the operations to other PEs. However, the Row pointer array in CSR is difficult to separate. The Row pointer array needs to be decoded along with the data array and col index array since it does not have the one-to-one property for non-zero elements. Namely, after a sparse matrix stored by CSR, it has been divided

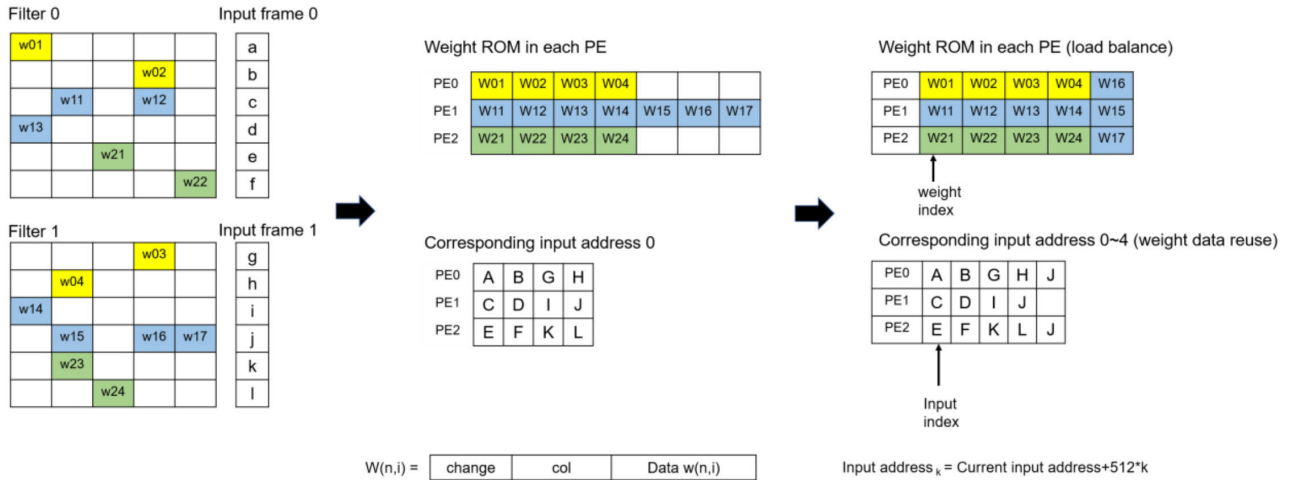


FIGURE 6. The example of PE pre-allocation in sparse matrix with BPCSR.

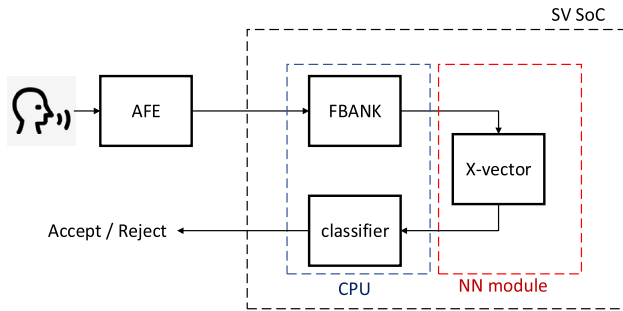


FIGURE 7. The SV procedure within our proposed SoC.

into PEs based on rows or columns. Note that redistributing the three arrays in CSR is difficult because they are not interdependent.

To address the problem mentioned above, we propose the Binary Pointer CSR (BPCSR), which is an improvement over CSR, making it more efficient for hardware implementation. In BPCSR, the data array and column index array function similarly to CSR, representing the non-zero values and their respective column positions. The row index array, however, is determined by a change-bit binary array. When the change bit is 1, the row index moves to the next address. For example, in Fig. 4(c), the BPCSR element (data, col index, change bit) = (5, 1, 0) indicates that the non-zero data 5 is at coordinate (0, 1). The row index remains 0 until the change bit becomes 1. For instance, (data, col index, change bit) = (1, 3, 1) shows that the data 1 is at coordinate (0, 3), and the row index then moves to the next row. When the next BPCSR element (data, col index, change bit) = (3, 0, 0) is processed, the row index is now 2, so data 3 is at coordinate (2, 0).

In practice, the row index array in BPCSR is converted to the corresponding input address, which can be calculated as:

$$\text{address} = \text{row index} * 512 * n + \text{frame offset} \quad (2)$$

where  $n$  is the filter number in a TDNN layer. Since our model uses 512 dimensions for each input and output layer in TDNN, each frame is stored sequentially in memory. This means that each address ROM only needs to store the address of the first frame based on the row index. For subsequent frames, the input address is obtained by adding the appropriate frame offset.

Compared to CSR, BPCSR offers a similar compression rate with just one additional binary array. The size of this array is determined by the vector of the matrix. As shown in Fig. 5, BPCSR allows for dynamic workload distribution among PEs to achieve load balancing. While some studies, such as [24], address the load-balancing issue in CSC/CSR, our approach provides a simpler and more efficient solution without complex computations. Additionally, BPCSR does not require a decoder when processing the TDNN, reducing the overall area and enabling a more compact computation process. More details are provided in Section V.

### E. PE PRE-ALLOCATION

PE load-balancing issues significantly affect overall performance, and proper PE pre-allocation can improve this problem. Fig. 6 shows the procedure of PE pre-allocation in sparse TDNN computation with our proposed BPCSR storage. The TDNN processes are similar to the 1D CNN, but different filters convolute with different frames and sum together to perform the dilation computation. Besides, TDNN is dominated by weight, i.e., each layer contains a large amount of weight.

Firstly, we initially allocate the workload by the row index of each filter to increase the data reuse. By doing so, each PE can completely calculate without reloading the same inputs until the next frame needs to be processed. Secondly, all the weights are transformed into BPCSR storage format, and the row index array in the BPCSR is processed as the corresponding address array. In Fig. 6, it is evident

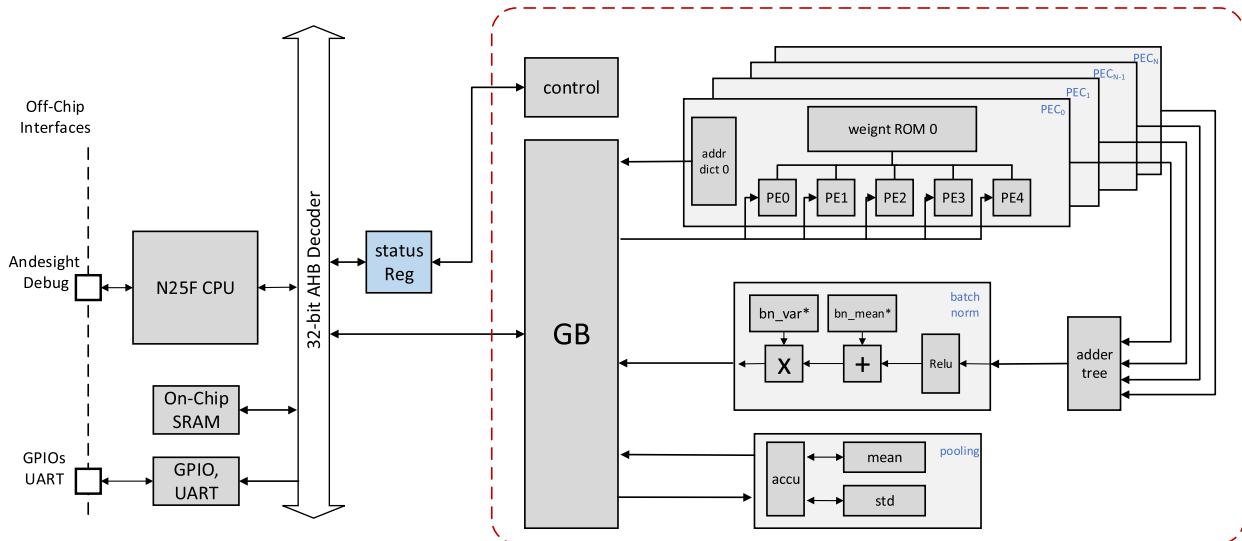


FIGURE 8. An overview of the proposed SV SoC.

#### Algorithm 1 The process workload of each PE

---

$N$  = the number of non-zero elements  
 $P$  = the number of PEs  
 concatenate filters in the column dimension  
 allocate the work-load to PEs by row  
 encode BPCSR  
 1:  $mean = N/P$   
 2: for ( $i = 0; i < N; i = i + 1$ ) {  
 3:    $w = \text{len}(\text{work}[i]) - mean$   
 4:   if ( $w \geq 0$ ) {  
 5:     append ( $temp, \text{work}[i][mean:w]$ )  
 6:     delete ( $\text{work}[i][mean:w]$ )  
 7:   }  
 8: }  
 9: for ( $i = 0; i < N; i = i + 1$ ) {  
 10:    $w = mean - \text{len}(\text{work}[i])$   
 11:   if ( $w \geq 0$ ) {  
 13:     append ( $\text{work}[i], temp[w:mean]$ )  
 14:     delete ( $temp[0:w]$ )  
 15:   }  
 16: }

---

that PE1 has more workload than the other PEs. Finally, with our proposed BPCSR format, PEs with more work can be easily assigned to other PEs. The extra work W16, W17, and their corresponding addresses are reallocated to PE0 and PE2. Thus, all the PE sets can achieve PE load-balancing, and the overall process cycle reduces from 7 to 5. In the actual operation of our proposed system, for example, if the work of the first TDNN layer is allocated to 16 PEs by storing the weights in CSR, the maximum workload in the PEs is three times the minimum one, which is 3096, and 1091 respectively. This is because the CSR format compresses the non-zero elements of the sparse matrix

together, causing the computation workload assigned to each PE to be uneven due to the distribution of these non-zero elements in the matrix. For example, some PEs may need to process more non-zero elements, while others process fewer. This results in some PEs having a heavy workload (such as 3096) and others having a lighter workload (such as 1091). To mitigate this issue, it may be necessary to further optimize the weight allocation strategy or improve the load balancing algorithm to ensure that the workload is more evenly distributed among all PEs. With the proposed BPCSR, they can achieve well balancing, namely, each PE is allocated to the same workload.

We used a simple algorithm to ensure that each PE is allocated similar workloads. The detail is represented in Algorithm 1. After the allocation according to the weight matrix row and the encoding of the BPCSR, each PE is allocated roughly uneven work. The array  $\text{work}[i]$  represents the BPCSR format workloads in  $i$ -th PE. Line 1 shows the amount of work that should be allocated to each PE. Line 2 to Line 8 aggregate the PE overwork, in which PE workloads are larger than the mean. The overwork will be put into the temp array and removed from the original allocation. Line 9 to Line 16 allocate the work in temp to PEs with less than average workload. This approach ensures that each PE is allocated an approximately mean workload and solves the load-balancing problem. Besides, this algorithm does not affect much data reuse and only fewer circumstances where it is necessary to reload the same input to different PEs

#### V. HARDWARE ARCHITECTURE DESIGN

This section describes the hardware architecture of the x-vector-based speaker verification SoC. The whole system achieves higher performance by separating neural network computation and other processes such as feature extraction and PLDA to neural network modules and CPU respectively.



Additionally, in the neural network module, all parameters are stored on-chip with BPCSR format, collaborating with the specific hardware architecture to improve power consumption, area, and latency.

### A. OVERALL ARCHITECTURE

Figure 7 shows the steps required for the hardware implementation of speaker verification. It consists of four main steps, analog front-end (AFE), FBANK feature extraction, x-vector extractor calculation, and PLDA scoring. The function of the AFE is to record speech and convert the speech from analog signals to digital signals. Since the CPU supports multiple types of peripheral IO, AFE can be replaced by the other existing recording devices. Thus, the AFE is not integrated into this work. Since the FBANK feature extraction includes a huge part of non-linear operations such as FFT and logarithm, processing FBANK is often more efficient when performed by the CPU. We designed a neural network accelerator module because the x-vector extractor consists of a large amount of multiplication, and the CPU still hard to affording the x-vector's workload even through the pruning and compression in the weight matrix. Finally, the CPU processes the PLDA scoring since it contains relatively complex non-linear computations.

The entire SoC architecture is represented in Fig. 8. The left part of the AHB bus decoder illustrates the CPU system, and the other side is the neural network accelerator module. After loading the necessary instructions and data, the CPU starts to process the program. When the CPU requires data, it sends the address to the bus decoder to decode the corresponding location based on the defined memory map to read the data. Firstly, the CPU processes the FBANK feature extraction. When the CPU completes FBANK and the result of each frame has been saved to the global buffer (GB), the CPU enables the status register to wake up the neural network module. Meanwhile, the CPU is set to wait for interrupt (WFI) mode, i.e., the system temporarily shuts down the CPU and cuts off the core clock until the interrupt occurs. Since the CPU completes the feature extraction, the authority of GB access is handed over from the CPU to the neural network module by the status register. During the computing in the neural network module, the module accesses the data from GB, and the processed data overwrites the original data. When the neural network module computations are finished, the module disables the status register to generate the interrupt. Since the CPU wakes up after an interrupt, the authority of GB access is returned to the CPU. The CPU copies the x-vector extractor from GB to other RAM. Lastly, the CPU computes the PLDA score with previously calculated x-vectors. Then PE's weight is stored in ROM to achieve area savings. The output of the SV can be represented in multiple ways by peripheral IO. For example, we can exploit the General Purpose input/output (GPIO) to directly activate some of the back-end devices or print the result in text form to the target terminal by Universal

Asynchronous receiver/transmitter (UART). Therefore, the SV system shows better flexibility.

### B. BN25F CPU

We integrate Andes N25F CPU in this work as an SoC. It is capable of delivering high per-MHz performance and operating at high frequencies while small in gate count. N25F also supports single- and double-precision floating-point and bit-manipulation instructions. The necessary modules such as the data bus, instruction local memory (ILM), data local memory (DLM), and peripheral IOs are integrated with the N25F CPU as the AE350 platform. Andesight is an Integrated Development Environment (IDE) for the AE350 platform. It compiles the C or C++ language, which is written by users. After the compiling, depending on the predefined memory mapping, the compiler processes the linker script to map each section of data into the chip. The control and status registers statement and the necessary data are transferred from the Andesight software when the CPU boots. After the CPU boots and all the data is transferred by JTAG, the CPU begins to process the program. CPU fetches the instructions from instruction local memory (ILM). The data local memory (DLM) stores the input data and the temporary calculation data.

### C. NEURAL NETWORK ACCELERATOR MODULE

Most neural networks include about 90% of computations on convolutional layers. It is evident that the use of highly parallel PE arrays efficiently speeds up highly repeatable operations. However, unlike conventional convolutional computations, TDNN has the same bottleneck in hardware design as 1D CNN or LSTM. The memory bandwidth problem is exacerbated by the high dimension filter and the low reusable weight in each frame. Namely, if the PE array does not receive the appropriate weights simultaneously, many stalls will occur.

To solve the problem of high memory bandwidth requirements and low data reuse, we refer to the hardware architecture in [23], where a hierarchical memory mesh computing architecture was proposed. Unlike the traditional approach, a cluster contains only one global buffer (GB). In [23], each PE cluster is allocated multiple GBs to store the weights, input activations (*iacts*), and partial sum (*psum*). In this work, all the network parameters are separated into the different weight ROMs connected to PEs. Additionally, each PE has its input buffer and relatively large output buffer. By doing so, the bandwidth has significantly improved.

After processing FBANK feature extraction, a short-duration utterance is converted into approximately 30 feature dimensions by 150 frames spectrum. The temporary data between TDNN layers is 512 dimensions by 150 frames due to the multiple 512 by 512 filters. We designed a highly parallel neural network module for this large number of computations. The architecture detail and its data flow are shown in Fig. 9. We divided the PE array into 16 PE clusters to increase the computational efficiency in larger filters.

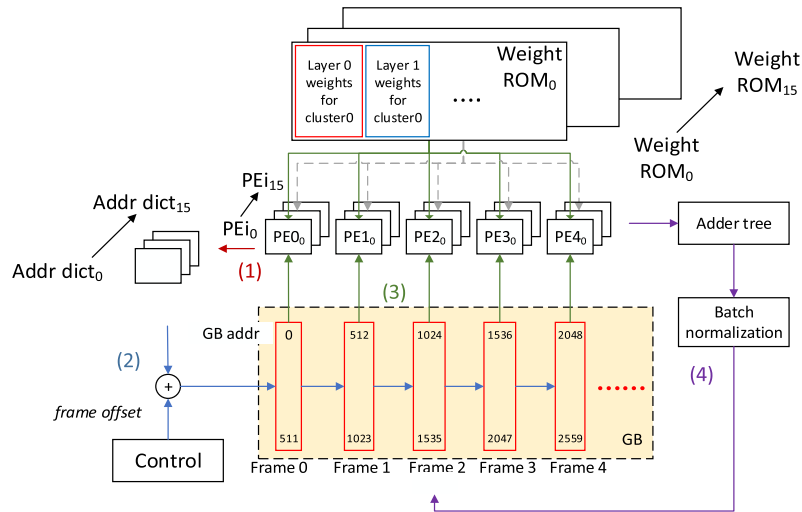


FIGURE 9. The proposed neural network module architecture and dataflow.

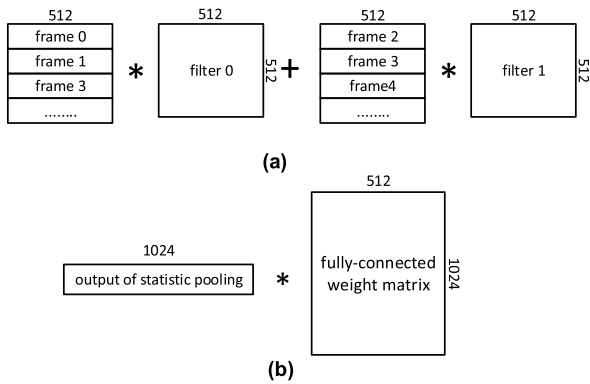


FIGURE 10. (a) The TDNN layers and (b) fully-connected layer in xvector represented as vector-matrix multiplication.

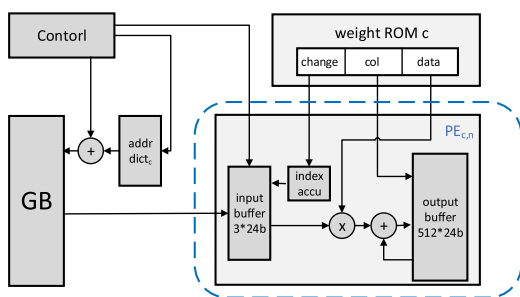


FIGURE 11. The proposed PE architecture.

Namely, each frame is processed in parallel by 16 PEs. On the other hand, to increase the data reusability further, each PE cluster has 5 PEs for the parallel calculation with the same filters but different frames. The address dictionary (*addr dict*) records the address required by PE0 for the first frame calculation. When the input buffer in the PE0 needs data, the GB address for input data is the sum of the current *addr* and *dict* value and the frame offset. At the same time, the address required for the other PEs in a PE cluster can be calculated

from the value in the *addr* and *dict* and the frame offset, as the frames are placed sequentially in the GB.

In the beginning, the control unit takes the current *addr* and *dict* value plus the frame offset as the GB address. Due to the control unit recording the layer and frame of the current calculation, the frame offset can be computed. At the same time, the control unit outputs a control signal to each PE to ensure that the PE receives the correct value. In the computation process, the weight ROM address will be added by one when each computation is finished to allow the PE to update the weight. The result of the computation is accumulated in the output buffer. When the input buffer needs to be updated, the weight ROM and PE computation will be suspended until the PE's input buffer has been updated. When the whole frame process is completed, the adder tree will sum the PE output buffer of 16 PE clusters according to each frame. After the batch normalization, data will be stored back to GB according to the position of the frame.

When the computation in the last frame is finished, the control unit incurs a local reset to the neural network module. Meanwhile, the system fetches the input of the next layer. After processing the TDNN4 layer, the system accumulates each frame and computes the average and standard deviation for statistical pooling. Finally, after the FC1 layer, we can get the x-vector. Most of the work is hard to process the convolutional and fully connected layers in the same hardware architecture. Even though they successfully share the computational resource, the system may need the additional circuit or perform a more complex data path. Fortunately, as shown in Fig. 10, the process of the fully-connected layer is similar to the TDNN layer. Namely, the FC1 layer can share the same hardware architecture as the TDNN layer in this work.

#### D. PROCESSING ELEMENT

Processing elements (PEs) are used to process multiplication and addition in the neural network accelerator. They are

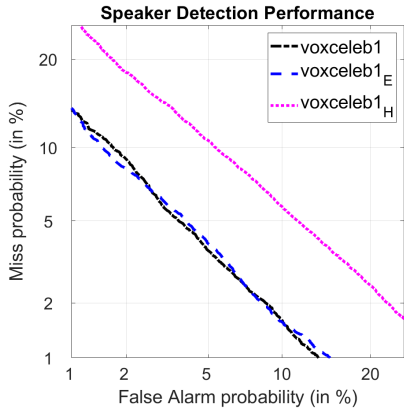


FIGURE 12. Detection Error Trade-off curves.

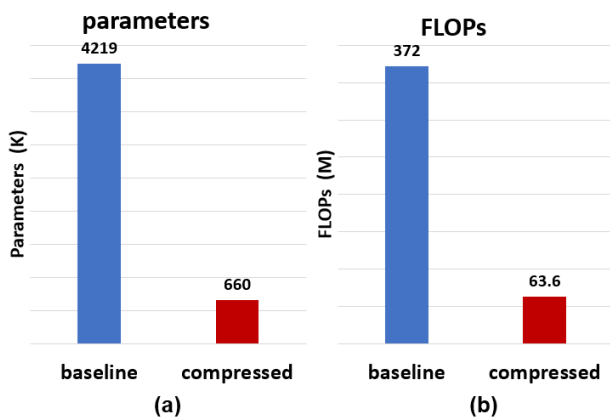


FIGURE 13. (a) The number of parameters and the (b) FLOPs between the original x-vector and the compressed x-vector in this work.

usually arranged as arrays to acquire high parallelism. Each PE connects to the global buffer (GB) to receive the input data and then transfers the output data back to the GB when the computations are finished.

However, the memory bandwidth can be seen as the bottleneck in TDNN due to the low data reusability. To prevent PEs access GB frequently, it is needed to include a larger output buffer in the PEs. The PE architecture and its calculation flow in this work are shown in Fig. 11. Each PE comprises a  $3 \times 24b$  input buffer, an index accumulator, a multiply-accumulate unit (MAC), and a  $512 \times 24b$  output buffer. The temporary data dimension in each TDNN layer is 512. Therefore, a  $512 \times 24b$  output buffer ensures sufficient space to store the entire temporary data of a frame. The adder tree sums every temporary data in each PE cluster and operates the batch normalization before writing back the output data to GB. Namely, 16 PEs are used to process a frame only on writing back the data to GB once, which significantly reduces the GB access.

Additionally, our proposed PE combines the decoding of BPCSR with standard MAC computation. After encoding the BPCSR storage format, the change, col\_index, and data array are stored into weight ROMc, and the corresponding input

address array is stored into addr\_dict where c represents the PE cluster. Furthermore, BPCSR does not need to be restored to the original matrix during the computation. Initially, the control unit sends the index to the address dictionary to fetch the necessary input address and sums it with the frame offset as the GB address. The GB writes the input data to the input buffer in the PE. Meanwhile, the input buffer receives the control signals from the control unit to ensure that only the specific PE can receive the input data. When all PEs successfully fetch three input data, the MAC unit will begin the computation. The change, col, and data array connect to the index accumulator of the input buffer, MAC unit, and the output buffer address, respectively, as shown in Fig. 11. During the computation, if the change bit equals 1, the index accumulator will be added by 1 to update the current input. Once the index accumulator is larger than two, i.e., all the input in the input buffer has been processed, PE will be stalled until the input buffer is updated and reset the index accumulator.

Since each PE independently works, one PE re-fetch the input data will not affect the other PEs' computation. Then the current input data is multiplied by the data. The col points out the corresponding data from the output buffer, then accumulates with the multiplied data and is stored back to the output buffer with the same address, col. The weight ROM address is added by one every two cycles. One is for reading out the data from the output buffer. The other is for writing back the accumulated data. PEs continue to process the computations until they finish an entire frame. When the next frame needs to be processed, PE will reset all the states and data in the buffers.

In summary, the weights can be processed directly in the BPCSR format without an extra decoding circuit. The processing can skip all the zero values without any extra spending cycles. Also, a larger output buffer in the PE addresses GB's congestion problem. Thus, the overall latency and the power efficiency have significantly improved.

## VI. EXPERIMENTAL RESULTS

### A. OVERALL ARCHITECTURE

The neural network in the training phase is trained by Python with Pytorch framework, while Python and Kaldi v5.5 emulate in the test phase. Kaldi is an open-source speech-related framework that processes the FBANK feature extraction and PLDA score in this work. Hereby we translated the FBANK and PLDA to C language and compiled them on the Andesight IDE to run on the N25F CPU.

As with most speaker verification in software research, we use the VoxCeleb2 [27] dataset for training and testing on the VoxCeleb1 [28] dataset. The VoxCeleb2 contains over 1 million utterances for 6112 different speakers. It records nearly 3000 hours of speech from YouTube videos. The VoxCeleb1 contains over 150,000 for 1251 speakers, and it also provides different trials for evaluating the performance of SV. We also tested on the speaker in the Wild (SITW)

**TABLE 3.** The error rate in different trials.

	EER (x)	DCF 0.01 (x)	DCF 0.001 (x)	EER (c)	DCF 0.01 (c)	DCF 0.001 (c)
SITW	4.757%	0.451	0.642	5.74%	0.514	0.719
VoxCeleb1	3.351%	0.368	0.589	4.268%	0.469	0.638
VoxCeleb1-H	3.404%	0.372	0.597	4.358%	0.450	0.699
VoxCeleb1-E	5.883%	0.534	0.769	7.299%	0.601	0.826
TIMIT	2.891%	0.372	0.627	3.74%	0.412	0.678

dataset. SITW is a more realistic and challenging dataset, which closely mirrors real-world scenarios.

After downsizing the model, the subsequent step involves training the network and pruning the training results. Subsequently, the pruned parameters are retrained, followed by quantizing the output to achieve the desired result.

In the testing phase, we evaluate all the data in the dataset, with each audio having a length of approximately 8 seconds. After comparing the two sets of audio, we will calculate the equal error rate (EER) from the results, and the accuracy will be derived using 1-EER. The quantity of data utilized by each tester may vary depending on the information available in the dataset.

**TABLE 4.** Accuracy and power consumption.

Bit precision of parameters	Accuracy on VoxCeleb1	Whole chip power consumption
8-bit	95.73%	97.88mW
4-bit	94.60%	72.42mW
2-bit	87.69%	60.17mW

### B. ACCURACY AND FLOPs

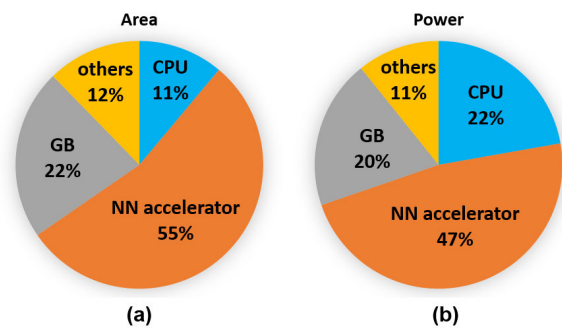
We tested the compressed x-vector extractor on three different trials, VoxCeleb1, VoxCeleb1-H, and VoxCeleb1-E. The original VoxCeleb1 test list with only 40 speakers. The challenging version of the VoxCeleb1-H list is where the test pairs are drawn from identities with the same gender and nationality. Table 3 shows the evaluation result in equal error rate (EER) and minimum detection cost function (DCF), where x represents the x-vector baseline and c represents the compressed x-vector in this work. EER shows the error rate when the false acceptance rate (FAR) equals the false rejection rate (FRR). The DCF can be expressed as follows:

$$\min DCF = C_{fa} * FAR * (1 - p_{target}) + C_{fr} * FRR * p_{target} \quad (3)$$

where  $C_{fa}$  is the risk factor for false accepted samples,  $C_{fr}$  is the risk factor for false rejecting a sample, and we set both

$C_{fa}$  and  $C_{fr}$  values to 1.  $p_{target}$  is the prior probability of the false occurring.  $p_{target}$  is usually set to 0.01 or 0.001.

The result shows that the x-vector extractor can achieve high accuracy in SV. The EER is between 3.35% to 5.88% depending on the different test lists. Figure 12 shows the detection error trade-off curves. However, as in Fig. 13, the number of parameters and the FLOPs in the x-vector is extremely high where the parameters are over 4.2M and the FLOPs are about 372M. The compressed x-vector in this work eliminated over 84% of parameters and nearly 83% of FLOPs, which are 660K in parameters and 63.6M FLOPs. Besides, the EER only increases by at most 1.42%. It shows a great trade-off between computational load and accuracy. Additionally, Table 4 shows the accuracy of VoxCeleb1 and the chip power consumption at different bit precisions of the parameters.

**FIGURE 14.** (a) The overall area breakdown. (b) The power breakdown.

### C. ASIC DESIGN AND AREA ANALYSIS

To reduce the area, we stored all the weights in ROM. A 64kB ROM shows only 27% area compared with SRAM in the same size, but ROM slightly increases the power consumption. The total memory usages are 1.28MB ROM for storing the neural network parameters and 568KB SRAM for CPU, GB, and PEs. The area breakdown and the power breakdown are shown in Fig. 14 (a) and Fig. 14 (b). Due to the large size of ROM and 80 PEs in the NN accelerator, it shows nearly 54% area and 47% power consumption in the whole chip. The CPU does not account for a significant proportion of the total area and power consumption since it does not include the Single Instruction Multiple Data (SIMD) unit.



TABLE 5. Comparison with State-of-the-art works.

	TVLSI'14 [16]	JSSCC'20 [15]	DATE'22 [33]	ASICON'21 [43]	ICTA'22 [44]	JSSCC'24[45]	This work
<b>Technology</b>	90nm	40nm	22nm	22nm	22nm	28nm	90nm
<b>Area</b>	19.53mm <sup>2</sup>	2.56mm <sup>2</sup>	0.28mm <sup>2</sup>	0.28 mm <sup>2</sup>	-	9.58 mm <sup>2</sup>	15.5mm <sup>2</sup>
<b>Algorithm</b>	SVM	GMM	BWN	BWN	TWN	BRNN	x-vector
<b>Dataset</b>	NIST SRE	TIMIT	VoxCeleb2	TIMIT	TIMIT	TIMIT	VoxCeleb
<b>Test speakers</b>	200	168	118	630	630	630	1251
<b>Programmable</b>	No	No	No	No	No	Yes	Yes
<b>Accuracy</b>	92.49%	99.5% *85%	98.6% (secondary) 78.5% (single)	99.6%	93.1%	84.8%	95.6%
<b>Supply voltage</b>	0.9V	0.6V-1.2V	0.39V & 0.6V	0.39V & 0.6V	0.6V	0.6V-1.1V	0.9V
<b>Latency per frame</b>	8ms (per feature vector)	1s	-	-	-	0.12ms	0.6s (CPU) 0.4s (SV accelerator)
<b>power</b>	8.12mW@100MHz	10.6uW@250k Hz	4.53uW@250k Hz	4.53uW@250M Hz	16.3uW@250M Hz	1.3mW@1.25M Hz	97.88mW@40 MHz
<b>Norm. power</b>	8.12mW	2.09uW	270mW	270mW	973.9mW	0.12mW	97.88mW

\* The accuracy evaluated on the VoxCeleb dataset by [28].

\* Normalizes to 90nm CMOS technology

#### D. COMPARISONS WITH EXISTING WORKS

Due to a few works implementing the whole speaker verification system on ASIC, our work is compared with five ASIC works in this paper. Table 5 shows the comparison between this work and the prior art. Most of the prior ASIC works exploited conventional algorithms such as GMM and SVM to achieve high energy efficiency. However, these works are difficult to achieve high accuracy in environments containing lots of speakers or noise.

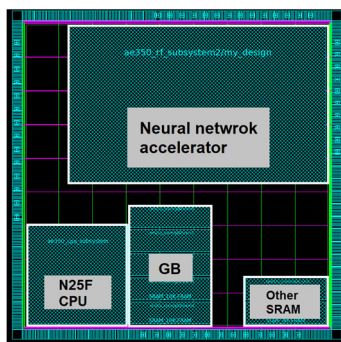


FIGURE 15. Layout of proposed speaker verification SoC.

In [16], the verification accuracy is only 92.49% tested on a clean dataset, NIST SRE with 200 speakers. In contrast, our work shows a 2.71% accuracy improvement evaluated on a more challenging dataset, VoxCeleb, which has 1251 speakers and more than 579K test samples. Although [15] shows good verification result which is higher than 99%, they only test 168 clean samples in the TIMIT dataset. Besides, the GMM method performs with only 85% accuracy on the VoxCeleb dataset refer to [28]. In [33], their work only tested on the VoxCeleb2 test set instead of the whole VoxCeleb

dataset. Although the secondary threshold can achieve good accuracy, the verification result may become unstable because of the dependency on the previous verifications. In [43] and [44], the validation accuracies on the clean dataset TIMIT are 99.6 and 93.1% respectively. Compared to [43], our accuracy is 4% lower, but this is because our work is evaluated on a more challenging dataset, VoxCeleb, which tests 1.98 times more speakers than [43], and for power comparison because of the difference in the process, therefore, the comparison was done after normalization to 90nm, and after normalization, the power consumption of [43] is 2.75 times higher than ours. Compared to [44] we tested a higher number of speakers as well as higher accuracy, and the power consumption of [44] is 9.94 times higher than ours.

Since our work contains a CPU, users can modify the program according to the different applications which are seldom addressed in the existing works. Thus, this work is more flexible and can be used in more scenarios. The Andes N25F CPU shows high energy and area efficiency. However, the RISC-V instruction set architecture supports fewer instructions, and the latency is relatively high, which takes nearly 600ms in FBANK and PLDA at 200MHz operational frequency. The proposed NN accelerator processes the sparse data with a compact and efficient data flow. Therefore, it takes 0.4s at 40MHz operational frequency (maximum can achieve 225MHz) to finish the compressed x-vector extractor computations.

#### VII. CONCLUSION

In this paper, a large and flexible x-vector-based speaker verification SoC is proposed. We consider the efficient realization of the neural network with its algorithm and architecture design. We propose the BPCSR sparse matrix

storage format which can easily pre-allocate the PEs in the NN accelerator module. With the compression in the x-vector extractor, over 84% of weights are removed and the accuracy still maintains 95.6% testing on the VoxCeleb1 dataset including 1251 speakers. With the proposed PE architecture, the PE array can achieve good PE load-balancing. Besides, the data stored in BPCSR format can be directly computed in PE without extra decoders, making the entire data flow more compact and efficient. Our system is designed on an SoC which mainly consists of the Andes N25F and the neural network accelerator. The accelerator is designed by ASIC and the chip is synthesized in TSMC 90 nm technology and performs 15.5 mm<sup>2</sup> area and 97.88 mW power consumption. Through cooperation between the CPU and NN accelerator, the overall SV system achieves real-time latency and good power efficiency.

## ACKNOWLEDGMENT

Andes N25F CPU and technical problem support are provided by Andes Technology

## REFERENCES

- [1] N. Egi, T. Hayashi, and A. Takahashi, "The proposal of quantification method of speaker identification accuracy for speech communication service," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2011, pp. 2424–2427.
- [2] D. A. Reynolds and R. C. Rose, "Robust text-independent speaker identification using Gaussian mixture speaker models," *IEEE Trans. Speech Audio Process.*, vol. 3, no. 1, pp. 72–83, Jan. 1995.
- [3] P. Kenny, G. Boulianne, P. Ouellet, and P. Dumouchel, "Speaker and session variability in GMM-based speaker verification," *IEEE Trans. Audio, Speech Lang., Process.*, vol. 15, no. 4, pp. 1448–1460, May 2007.
- [4] N. Dehak, P. J. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-end factor analysis for speaker verification," *IEEE Trans. Audio, Speech, Lang., Process.*, vol. 19, no. 4, pp. 788–798, May 2011.
- [5] M. Li, A. Tsiartas, M. Van Segbroeck, and S. S. Narayanan, "Speaker verification using simplified and supervised i-vector modeling," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2013, pp. 7199–7203.
- [6] S. Cumani, O. Plchot, and P. Laface, "On the use of i-vector posterior distributions in probabilistic linear discriminant analysis," *IEEE/ACM Trans. Audio, Speech, Lang., Process.*, vol. 22, no. 4, pp. 846–857, Apr. 2014.
- [7] C. J. S. de Souza, D. C. G. González and L. L. Ling, "VVGP features for speaker verification using i-vector framework," in *Proc. Int. Workshop Telecommun. (IWT)*, 2015, pp. 1–4.
- [8] E. Variani, X. Lei, E. McDermott, I. L. Moreno, and J. Gonzalez-Dominguez, "Deep neural networks for small footprint text-dependent speaker verification," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Jun. 2014, pp. 4052–4056.
- [9] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur, "X-vectors: Robust DNN embeddings for speaker recognition," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2018, pp. 5329–5333.
- [10] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 37, no. 3, pp. 328–339, Mar. 1989.
- [11] F. A. Rezaur rahman Chowdhury, Q. Wang, I. L. Moreno, and L. Wan, "Attention-based models for text-dependent speaker verification," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2018, pp. 5359–5363.
- [12] C. -P. Chen, S. -Y. Zhang, C. -T. Yeh, J. -C. Wang, T. Wang, and C. -L. Huang, "Speaker characterization using TDNN-LSTM based speaker embedding," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, 2019, pp. 6211–6215.
- [13] F. Zhao, H. Li, and X. Zhang, "A robust text-independent speaker verification method based on speech separation and deep speaker," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2019, pp. 6101–6105.
- [14] J. S. P. Giraldo, S. Lauwereins, K. Badami, H. Van Hamme, and M. Verhelst, "18μW SoC for near-microphone keyword spotting and speaker verification," in *Proc. Symp. VLSI Circuits*, 2019, pp. C52–C53.
- [15] J. S. P. Giraldo, S. Lauwereins, K. Badami, and M. Verhelst, "Vocell: A 65-nm speech-triggered wake-up SoC for 10-μW keyword spotting and speaker verification," *IEEE J. Solid-State Circuits*, vol. 55, no. 4, pp. 868–878, Apr. 2020.
- [16] J.-C. Wang, L.-X. Lian, Y.-Y. Lin, and J.-H. Zhao, "VLSI design for SVM-based speaker verification system," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 7, pp. 1355–1359, Jul. 2015.
- [17] D. Snyder, D. Garcia-Romero, D. Povey, and S. Khudanpur, "Deep neural network embeddings for text-independent speaker verification," in *Proc. Interspeech*, 2017, pp. 999–1003, doi: 10.21437/Interspeech.2017-620.
- [18] X. Zhang, X. Zou, M. Sun, T. F. Zheng, C. Jia, and Y. Wang, "Noise robust speaker recognition based on adaptive frame weighting in GMM for i-Vector extraction," *IEEE Access*, vol. 7, pp. 27874–27882, 2019.
- [19] D. Kadetotad, V. Berisha, C. Chakrabarti, and J.-S. Seo, "A 8.93-TOPS/W LSTM recurrent neural network accelerator featuring hierarchical coarse-grain sparsity with all parameters stored on-chip," *IEEE Solid-State Circuits Lett.*, vol. 2, no. 9, pp. 119–122, Sep. 2019.
- [20] K. -Y. Fan, J. -H. Chen, C. -N. Liu, and J. -D. Huang, "Performance optimization for MLP accelerators using ILP-based on-chip weight allocation strategy," in *Proc. Int. Symp. VLSI Design, Autom. Test (VLSI-DAT)*, 2022, pp. 1–4.
- [21] S. Wang, P. Lin, R. Hu, H. Wang, J. He, Q. Huang, and S. Chang, "Acceleration of LSTM with structured pruning method on FPGA," *IEEE Access*, vol. 7, pp. 62930–62937, 2019.
- [22] X. Dai, H. Yin, and N. K. Jha, "Grow and prune compact, fast, and accurate LSTMs," *IEEE Trans. Comput.*, vol. 69, no. 3, pp. 441–452, Mar. 2020.
- [23] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 292–308, Jun. 2019.
- [24] J. Park, W. Yi, D. Ahn, J. Kung, and J.-J. Kim, "Balancing computation loads and optimizing input vector loading in LSTM accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 9, pp. 1889–1901, Sep. 2020.
- [25] [Online]. Available: <http://www.andestech.com/en/products-solutions/andescore-processors/RISC-V-n25f/>
- [26] M. Jiao, Y. Li, P. Dang, W. Cao, and L. Wang, "A high performance FPGA-based accelerator design for end-to-end speaker recognition system," in *Proc. Int. Conf. Field-Programmable Technol. (ICFPT)*, Dec. 2019, pp. 215–223.
- [27] J. S. Chung, A. Nagrani, and A. Zisserman, "VoxCeleb2: Deep speaker recognition," in *Proc. INTERSPEECH*, vol. 1, Sep. 2018, pp. 1086–1090, doi: 10.21437/Interspeech.2018-1929.
- [28] A. Nagrani, J. S. Chung, and A. Zisserman, "VoxCeleb: A large-scale speaker identification dataset," in *Proc. Interspeech*, Aug. 2017, pp. 2616–2620, doi: 10.21437/Interspeech.2017-950.
- [29] R. Ramos-Lara, M. Lopez-Garcia, E. Canto-Navarro, and L. Puente-Rodriguez, "SVM speaker verification system based on a low-cost FPGA," in *Proc. Int. Conf. Field Program. Log. Appl.*, Aug. 2009, pp. 582–586.
- [30] R. Ramos-Lara, M. López-García, and E. Cantó-Navarro, "Real-time speaker verification system implemented on reconfigurable hardware," *J. Signal Process. Syst.*, vol. 71, pp. 89–103, Jun. 2013.
- [31] E. Cantó-Navarro, M. López-García, R. Ramos-Lara, and R. Sánchez-Reillo, "Flexible biometric online speaker-verification system implemented on FPGA using vector floating-point units," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 11, pp. 2497–2507, Nov. 2015.
- [32] A. S. Bora, R. Reddy, S. Satpathy, H. Balachander, V. Vijendra, G. Trivedi, and R. Sinha, "Power efficient speaker verification using linear predictive coding on FPGA," in *Proc. Int. CET Conf. Control, Commun., Comput. (IC)*, Jul. 2018, pp. 260–265.
- [33] B. Liu, H. Cai, X. Zhang, H. Wu, A. Xue, Z. Zhang, Z. Wang, and J. Yang, "A target-separable BWN inspired speech recognition processor with low-power precision-adaptive approximate computing," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2022, pp. 196–201.

- [34] T. Tambe, E.-Y. Yang, G. G. Ko, Y. Chai, C. Hooper, M. Donato, P. N. Whatmough, A. M. Rush, D. Brooks, and G.-Y. Wei, "9.8 a 25 mm<sup>2</sup> SoC for IoT devices with 18ms noise-robust speech-to-text latency via Bayesian speech denoising and attention-based sequence-to-sequence DNN speech recognition in 16 nm FinFET," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 64, Feb. 2021, pp. 158–160.
- [35] T.-J. Lin, C.-Z. Liao, Y.-J. Hu, W.-C. Hsu, Z.-X. Wu, and S.-Y. Wang, "A 40 nm CMOS SoC for real-time dysarthric voice conversion of stroke patients," in *Proc. 27th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, 2022, pp. 7–8.
- [36] M. D. Balasingam and C. S. Kumar, "Refining cosine distance features for robust speaker verification," in *Proc. Int. Conf. Commun. Signal Process. (ICCCSP)*, Apr. 2018, pp. 0152–0155.
- [37] S. J. D. Prince and J. H. Elder, "Probabilistic linear discriminant analysis for inferences about identity," in *Proc. IEEE 11th Int. Conf. Comput. Vis.*, Jul. 2007, pp. 1–8, doi: [10.1109/ICCV.2007.4409052](https://doi.org/10.1109/ICCV.2007.4409052).
- [38] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.
- [39] Y. Zhang, "Hello edge: Keyword spotting on microcontrollers," 2017, *arXiv:1711.07128*.
- [40] R. Serrano, M. Sarmiento, C. Duran, K.-D. Nguyen, T.-T. Hoang, K. Ishibashi, and C.-K. Pham, "A low-power low-area SoC based in RISC-V processor for IoT applications," in *Proc. 18th Int. SoC Design Conf. (ISOCC)*, Oct. 2021, pp. 375–376.
- [41] F. Müller and A. Mertins, "Feature extraction with a multiscale modulation analysis for robust automatic speech recognition," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2013, pp. 7427–7431.
- [42] L. He, R. Li, and M. Niu, "A study on graph embedding for speaker recognition," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Seoul, South Korea, Apr. 2024, pp. 10741–10745, doi: [10.1109/icassp48485.2024.10448308](https://doi.org/10.1109/icassp48485.2024.10448308).
- [43] H. Wu, "An always-on ultra-low power speaker verification accelerator based on binary weighted neural network with system co-optimization," in *Proc. IEEE 14th Int. Conf. ASIC (ASICON)*, 2021, pp. 1–4.
- [44] X. Zhang, "A TWN inspired speaker verification processor with hardware-friendly weight quantization," in *Proc. IEEE Int. Conf. Integr. Circuits, Technol. Appl. (ICTA)*, Jul. 2022, pp. 160–161.
- [45] Y.-H. Tsai, Y.-C. Lin, W.-C. Chen, L.-Y. Lin, N.-S. Chang, C.-P. Lin, S.-H. Chen, C.-S. Chen, and C.-H. Yang, "A 28-nm 1.3-mW speech-to-text accelerator for edge AI devices," *IEEE J. Solid-State Circuits*, vol. 59, no. 11, pp. 3816–3826, Nov. 2024, doi: [10.1109/jssc.2024.3389965](https://doi.org/10.1109/jssc.2024.3389965).



**TSUNG-HAN TSAI** (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from National Taiwan University, Taipei, Taiwan, in 1990, 1994, and 1998 respectively.

From 1999 to 2000, he was an Associate Professor of electronic engineering with Fu Jen University. He joined National Central University, in 2000. Since 2008, he has been a Full Professor with the Department of Electrical Engineering, National Central University. He is currently the Director of the Intelligent Chip and System Center, National Central University, and also the Principal Investigator of the National Program for Intelligent Electronics. His research interests include VLSI signal processing, video/audio coding algorithms, DSP architecture design, wireless communication, and system-on-chip design. He has been awarded more than 40 patents and 230 refereed papers published in international journals and conferences. He received the Industrial Cooperation Award in 2003 from the Ministry of Education, Taiwan, the Best Paper Award from the IEEE International Conference on Innovations in Bio-Inspired Computing and Applications (IBICA) in 2011, and the IEEE International Conference on Innovation, Communication and Engineering (ICICE) in 2015. His research team has won many international IC related student design contest awards, including ISOCC in 2015, TI DSP Asia Design Contest in 2008, and ISSCC in 2011. He was the General Co-Chair of the IEEE International Conference on Internet of Things 2014. He serves as a technical program committee member or the session chair for several international conferences. He served as the Guest Editor for a Special Issue on *Journal of VLSI Signal Processing Systems*.



**MENG-JUI CHIANG** received the B.S. degree in electrical engineering from National Central University, Taiwan, in 2020, where he is currently pursuing the M.S. degree in electrical engineering. His research interest includes deep learning for computer vision.

• • •