A 510-nW Wake-Up Keyword-Spotting Chip Using Serial-FFT-Based MFCC and Binarized Depthwise Separable CNN in 28-nm CMOS

Weiwei Shan, *Member, IEEE*, Minhao Yang, *Member, IEEE*, Tao Wang, Yicheng Lu, Hao Cai, *Member, IEEE*, Lixuan Zhu, Jiaming Xu, Chengjun Wu, Longxing Shi, *Senior Member, IEEE*, and Jun Yang, *Member, IEEE*

***Abstract*— We propose a sub-μW always-ON keyword spotting (μKWS) chip for audio wake-up systems. It is mainly composed of a neural network (NN) and a feature extraction (FE) circuit. For significantly reducing the memory footprint and computational load, four techniques are used to achieve ultra-low-power consumption: 1) a serial-FFT-based Mel-frequency cepstrum coefficient circuit is designed for FE, instead of the common parallel FFT. 2) A small-sized binarized depthwise separable convolutional NN (DSCNN) is designed as the classifier. 3) A framewise incremental computation technique is devised in contrast to the conventional whole-word processing. 4) Reduced computation allows a low system clock frequency, which enables near-threshold voltage operation, and low leakage memory blocks are designed to minimize the leakage power. Implemented in 28-nm CMOS technology, this μKWS consumes 0.51 μW at a 40-kHz frequency and a 0.41-V supply, with an area of 0.23 mm^2. Using the Google speech command data set, 97.3% accuracy is reached for a one-word KWS task and 94.6% for a two-word task.**

***Index Terms*— Binary neural network (NN), data reuse, depthwise separable convolution (DSC), keyword spotting (KWS), near-threshold voltage (NTV) design, serial fast Fourier transform (FFT).**

I. Introduction

LOW-POWER speech interface is of high interest nowadays with the rapid development of Internet of Things applications, including various wearable devices. Power-efficient keyword spotting (KWS) that detects one or a few command words to wake up functional blocks for more complex tasks is particularly under intensive research at present in both software and hardware.

Manuscript received May 5, 2020; revised July 25, 2020 and September 2, 2020; accepted September 28, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61774038 and Grant 62074035 and in part by China Major S&T Project under Grant 2018ZX01031-101. This article was presented in part at the IEEE International Solid-State Circuits Conference, Feb. 2020, pp. 230–232. This article was approved by Guest Editor Jun Deguchi. *(Corresponding author: Jun Yang.)*

Weiwei Shan, Tao Wang, Yicheng Lu, Hao Cai, Lixuan Zhu, Jiaming Xu, Chengjun Wu, Longxing Shi, and Jun Yang are with the National ASIC Center, School of Electronic Science and Engineering, Southeast University, Nanjing 210096, China (e-mail: wwshan@seu.edu.cn; taowang@seu.edu.cn; seu_xjm@163.com; hao.cai@seu.edu.cn; flappylyc@163.com; wucj@seu.edu.cn; lxshi@seu.edu.cn; dragon@seu.edu.cn).

Minhao Yang is with the Department of Electrical Engineering, Neuchâtel EPFL, 10027 Neuchâtel, Switzerland (e-mail: yangmh.ic@gmail.com).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSSC.2020.3029097

Traditionally hidden Markov models (HMMs) [11]–[15] have been used in KWS with one HMM trained for each keyword and another for nonkeyword fillers. With the recent advances of successfully employing neural networks (NNs) in many vision and audio tasks and achieving state-of-the-art performance [16]–[22], NN-based audio inference has become a popular approach. Even though the computational load of NN is generally heavy, there has been substantial progress in tailoring it down for running on energy-constraint platforms. Two very effective techniques are the depthwise separable convolution (DSC) and the weight/activation quantization. DSC has extensively used in MobileNet [23] with much-reduced computation and only slight accuracy degradation. Basically, it decomposes the conventional 3-D convolution to 2-D depthwise convolution plus 1×1 pointwise convolution. The extreme examples of quantization reduce weights and/or activations from floating-point to binary, like in BinaryConnect and BinaryNet [24], [25]. In both cases, the energy-consuming multiplier is completely avoided, and the storage requirement is largely relieved. In addition, data reuse in NN has also been explored to reduce computation [26]–[28]. These software/hardware techniques in NN have provided new opportunities for ultra-low-power KWS systems.

Nevertheless, the existing NN hardware for KWS still consumes too much power with a large memory footprint. For example, the 288-μW NN for KWS in [5] has a large on-chip weight memory of 270 kB. The binarized CNN in [6] still has a 52-kB memory and consumes 141 μW. The long short-term memory (LSTM) accelerator in [7] uses a 32-kB memory and consumes 5 μW without including feature extraction (FE). And the work in [8], including FE, has a memory of 105 kB and consumes 16.11 μW. All of them would result in a KWS system power higher than 10 μW.

The burden of large on-chip memory and heavy computational load also exists in the FE part of a KWS system [6]–[8], [29], [30]. In the pursuit of extreme power efficiency, analog FE has been used to demonstrate voice activity detection [31]. But the analog approach has the disadvantage of being difficult in technology migration and more prone to variations. One of the most widely adopted FE algorithms, Mel-frequency cepstrum coefficient (MFCC), is normally implemented in digital. While the algorithm-to-hardware mapping is relatively straightforward, the complex computations that normally involve FFT lead to high-power consumption.

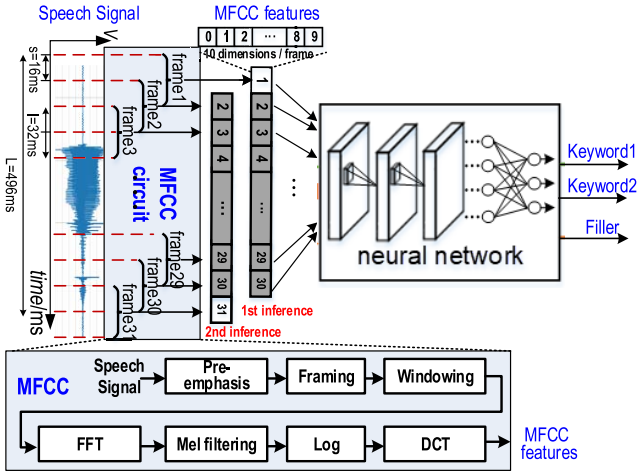


Fig. 1. Basic architecture of deep learning-based keyword spotting system.

For example, the MFCC FE in [8] uses a 1028-point DFT and consumes 7.14 μW, and the KWS chip in [6] uses a 512-point FFT with a 4-kB memory and consumes 141 μW.

In this article, aiming for low-power always-ON KWS for wake-up, we propose μKWS: a sub-μW, low-memory, low-computation KWS chip based on binarized NN to detect one or two keywords [37]. Our main contributions are as follows:

1) A serial-FFT-based MFCC circuit for FE with optimized internal data quantization, which efficiently reduces data storage and computations as compared with the parallel-FFT-based MFCC circuit.
2) A binarized DSC NN (DSCNN) for KWS, reducing memory footprint and computations as compared with a traditional CNN.
3) A framewise incremental computation technique, eliminating the repetitive computations and redundant data storage greatly as compared with the traditional whole-word processing way, without accuracy loss.
4) Near-threshold voltage (NTV) design of the whole chip with customized, low leakage latch-like memory, reducing leakage by 12× as compared with the compiler-generated SRAM with the same size.

This article is organized as follows. Section II introduces the sub-μW KWS circuit challenges and our main ideas. Section III is our serial FFT-based MFCC circuit optimizations. Section IV shows the DSCNN architecture, circuit design, and framewise incremental computation technique. Section V is our customized latch-based memory design for NTV. Section VI presents chip implementation and measurement results. Section VII is a discussion. Section VIII concludes this article.

II. Sub-μW KWS Circuit Challenges and Our Ideas

A. Basic Architecture of KWS System

Fig. 1 shows a typical KWS system consisting of a feature extractor (left) and an NN (right). First, the input speech signal of length L is framed into consecutive frames with a length of l and a shift of sh. Here, $L = 496$ ms, $l = 32$ ms, and $sh = 16$ ms are set for the keywords, such as "happy," "dog," "right," and so on, in the Google speech command data set (GSCD). Thus, a total of $T = (L - l)/sh + 1 = 30$ frames are obtained for each KWS inference. For each frame, F dimensions of speech features are extracted (here, $F = 10$, determined by the way shown in Section III-A), giving a total of $T \times F$ (30×10 here) features as the input data fed into the NN for classification. The NN calculates the probabilities for each class. In a real-world scenario, where keywords need to be recognized from a continuous audio stream, the probabilities of each output class over a period could be averaged by a posterior handling module, which improves the overall confidence of the prediction. Such a post-processing module is not included in our demo chip; our accuracy reports and power estimations do not include this part.

GSCD [35] is used to train the NN in TensorFlow [36]. This data set consists of 65 000 1-s-long audio clips of 30 words provided by thousands of different people, with each clip consisting of only one keyword. We trimmed each audio clip to 496 ms by removing the blank parts. In the training set, the ratio of fillers to keywords is 6:1, and the test set contains around 5000 audios. For continuous audio stream, we concatenate the audios from the test set to form a 1-h-long speech.

B. Sub-μW KWS Chip Design Challenges

As shown in Fig. 1 (bottom), MFCC transforms the original speech sequence into a 2-D feature matrix for the NN classifier. There are many challenges in the design of low-power MFCC. First, FFT is a complex algorithm that is not friendly to hardware implementation. We analyze the radix-2 decimation-in-frequency (DIF) algorithm [32], which is a simplified method to implement FFT, as shown in (1)

$$\begin{cases} X(2r) = \displaystyle\sum_{n=0}^{N/2-1} \left[x(n) + x\left(n + \frac{N}{2}\right)\right] W_{N/2}^{nr} \\ X(2r+1) = \displaystyle\sum_{n=0}^{N/2-1} \left[x(n) - x\left(n + \frac{N}{2}\right)\right] W_N^n W_{N/2}^{nr} \\ W_N = e^{-j\frac{2\pi}{N}} \end{cases} \quad r = 0, \ldots, \frac{N}{2} - 1 \quad (1)$$

where N is the number of points, x is the input series, X is the output series, and W_N is the twiddle factor. It divides the output sequence into odd–even groups in the frequency domain to do a divide-and-conquer computation. In this method, $\mathrm{Log_2N}$ stages are necessary for an N-point FFT, and each stage includes $N/2$ butterfly operations and N intermediate data, as shown in Fig. 2, with $N = 8$ for a simple illustration. Thus, it requires lots of butterfly units (BF units) and large intermediate storage, which consumes the most power in the whole MFCC circuit.

Second, the natural logarithm needs a mass of iterations and intermediate data storage, which decreases the throughput of the MFCC circuit while consuming a lot of power

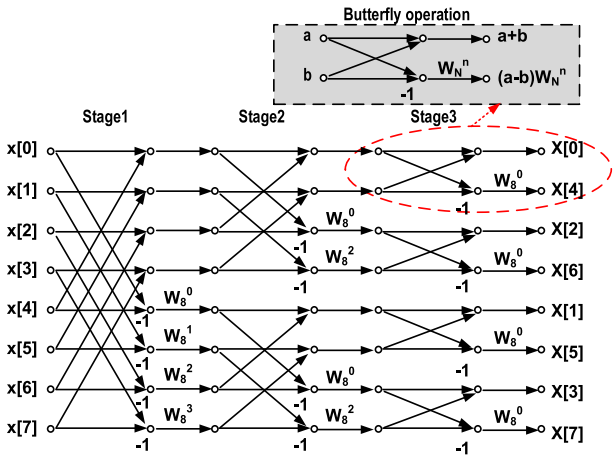


Fig. 2. Illustration of radix-2 DIF algorithm of 8 points.

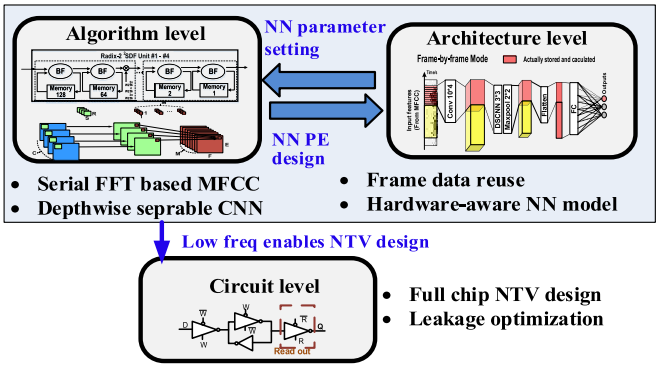


Fig. 3. Our cross-layer optimizations.

and area. Third, the expansion of data width is inevitable in the processing steps of MFCC, bringing a huge burden of data storage. Therefore, how to truncate and quantize the intermediate data is another key problem. Finally, it is very important to coordinate these processing steps efficiently for the throughput and latency of the FE circuit.

NN's power is another challenge since CNN and fully connected NN have heavy computations and large storage in KWS. Therefore, there is an urgent need to design a light-weight NN while making full use of data reuse optimizations to reduce the total computation and data storage.

Furthermore, since one keyword is tens of frames long, the NN needs to make the inferences regarding all the frames, which brings heavy computations and memory. For example, 31 frames by 13 dimensions of features were used in [5], and 11 frames by 40 dimensions were used in [6]. As shown in Fig. 1, although each NN inference uses 30 frames of input features, there is only one frame totally different between two adjacent inferences. Yin *et al.* [6] reused convolution results to eliminate part of the redundant computations but had no reduction in memory. Thus, a more efficient data reuse method would be a great help in reducing power.

C. Our Cross-Layer Optimizations

Fig. 3 shows the cross-layer optimization in our KWS chip. First, we propose optimizations at the algorithm level to reduce data storage and computations, including serial-FFT-based MFCC for FE and a lightweight DSCNN targeting 1–2 keyword detection. Second, at the architecture level, we propose a novel framewise incremental computation technique for NN to eliminate both the repeated computations and the redundant storage, which is an effective data reuse method. This technique, in turn, guides the NN parameter setting during training to obtain a power-friendly hardware architecture. By then, the number of computations is reduced dramatically to enable an ultra-low frequency of 40 kHz to finish one frame of data processing. Thus, the NTV design of the whole chip is enabled, which further decreases the total power dramatically. And we customize latch-based memory blocks to minimize leakage power.

III. Serial FFT-Based MFCC Circuit Optimizations

A. Architecture Choice of MFCC

A hardware-friendly MFCC architecture and its design parameters are optimized by low-power consideration and verified in the whole KWS training/testing system. To be clear, it is done after NN architecture is determined.

First, based on the standard MFCC model in GSCD TensorFlow example [35], we design a low-power MFCC circuit with the following optimizations: 1) set the frame length to be 2^M (32 ms with 16 ms shift), which is convenient for hardware implementation. Decrease the sampling rate from a standard 16 to 8 kHz, so that the FFT is decreased to 256-point, which saves a lot of computation resources as compared with the 512-point FFT [6] and the 1028-point DFT [8]. 2) Based on the recommended settings from a standard MFCC algorithm, we select smaller parameters to decrease the hardware overhead. For example, the number of Mel filter is set as the minimum allowable value of 20, and the output number of DCT is chosen as 11. Then, training was performed with this modified MFCC to make sure it does not compromise the KWS accuracy.

Second, for MFCC architecture design, we adopt a serial-FFT to decrease the storage and computations. We then build a C model to substitute the previous MFCC program and train the KWS system to obtain a proper setting of our serial FFT.

Third, during the HW design, we truncate and quantize the intermediate data through hardware and software co-design, which determined the final MFCC hardware architecture. Again, it is verified by feeding the quantized data into the training system. Those settings with no apparent accuracy loss are accepted, and the best one is chosen for hardware design among our extensive experiments.

In the end, we have a 256-point serial FFT-based MFCC with 11 dimensions of 8-b output and optimized internal data quantization. Its computations are much lighter than the 512-point FFT-based FE with 40 dimensions of 16-b output [6] and the 1028-point DFT-based FE [8].

B. Serial-FFT Advantages

FFT is the main part of the MFCC circuit, which consumes most of the power consumption because of its heavy computation and large storage. Optimizations of FFT help reduce

	Stoarge	Butterfly function units	Thoughput	Power* (μW)
Parallel FFT	$N*2^N$	$N*2^{N-1}$	2^N	2.281
TM FFT	$2*2^N$	2^{N-1}	$2^N/N$	-
Serial FFT	2^N	N	1	0.207

*Simulated by PTPX

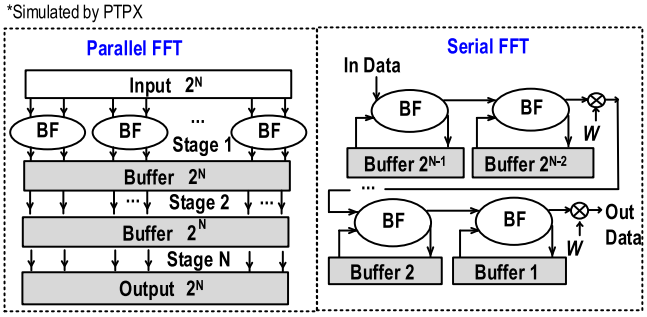


Fig. 4. Comparison of parallel, serial, and TM FFT.

the power [33], [40]–[43]. For example, a hardware-oriented radix-22 algorithm reduces both multiplier and storage overhead significantly [40]. A digital array signal processor provides both the FFT specific and the general-purpose arithmetic/logic functions [41], which supports radix-4, radix-2, and mixed radix-4/radix-2 FFT algorithms. Radix factorization achieves high energy efficiency with flexibility [42]. An energy-aware architecture with variable bit precision and FFT length was proposed in [43].

The hardware of FFT decomposes 2^N-point FFT into N stages, where 2^{N-1} butterfly operations are performed in each stage. It can be implemented in three ways: parallel, serial, and time multiplexed (TM), where parallel and serial FFT are shown in Fig. 4. Parallel FFT is the most direct and intuitive implementation, which achieves high throughput while having very high hardware costs too. For a 2^N-point FFT, its hardware implementation consists of $N{\cdot}2^{N-1}$ butterfly units and $N{\cdot}2^N$-depth memory. In our previous design, it consumes 2.281 μW by Primetime PX simulation at TT corner, 0.5 V, and 25°, which already exceeds the power goal we are targeting.

TM FFT decreases the hardware cost and throughput by reducing the data storage from $N{\cdot}2^N$ to $2{\cdot}2^N$. It time multiplexes the storage after the completion of each step. Compared with the above two methods, serial FFT only contains N butterfly units and 2^N-depth memory [33], [40], dramatically cutting the hardware resources, resulting in minimal power consumption of 0.207 μW at a price of lower throughput, which is nonetheless acceptable in voice processing.

To be specific, for a 256-point FFT, 8 $\times$ 128 butterfly units and 8 $\times$ 256-depth memory are needed in parallel implementation, while only 8 butterfly units and 256-depth memory are necessary for serial FFT, as shown in Fig. 4.

C. MFCC Circuit Optimizations

As shown in Fig. 5(a), the MFCC circuit consists of seven modules to perform pre-emphasis, framing, windowing, FFT, Mel filtering, logarithmic operation, and DCT. To reduce its power, we optimize the following parts.

First, the 256-point serial FFT circuit is composed of four radix-2^2 single-path delay feedback (R2^2SDF) units [40], each including two processing units and a twiddle unit, as shown in Fig. 5(b). The processing unit is composed of a butterfly unit and an independent memory. The butterfly unit completes the complex operation by adding and subtracting the real and the imaginary parts, respectively. The depth of memory in the first processing unit is 128 and that of the next cascaded processing unit decreases to half. The twiddle unit multiplies the output of the processing unit with the twiddle factor and transfers the results into the next stage. The input of serial FFT is in the natural order, while the output is in bit-reversed order, as shown at the bottom in Fig. 5(b). This is because the DIF algorithm divides the output sequence into odd–even groups in the frequency domain to do a divide-and-conquer computation, as shown in the FFT algorithm [49]. In the following step, we do the correct addressing to ensure correct computation.

Second, we replace the complicated natural logarithm operation with a $\log_2$ operation. Furthermore, we simplify it by using a ceiling approximation to implement its hardware as a leading-one detector. Specifically, it is implemented using a lookup table (LUT) with 32-b input and 6-b output, as shown in Table I. The input signal has 32-b data width in Table I, where "x" denotes "do not care bits." For instance, for input of 32'b0000…00001111 (d'15) as the gray row in Table I, its accurate $\log_2$ result is 3.907. Here, its hardware result is four causes the position of the leading one is four. This approximation reduces hardware cost while having no degrading effect on the final KWS accuracy according to our training experiments.

Third, to decrease the operating frequency, these 4 main blocks that last hundreds of clock cycles are operated in the pipeline, which are FFT, Mel filtering, DCT, and NN with 266, 263, 204, and 558 clock cycles, respectively, in our chip. Thus, it makes the whole chip be able to work at the same 40-kHz frequency to finish KWS in 640 clock cycles every 16 ms, as shown in Fig. 5(c). The 4 main blocks are assigned with one frame of latency, respectively; therefore, the total latency is 64 ms, which is acceptable.

Fourth, as mentioned in Section II-B, data width will be expanded after multiplication and addition. Therefore, we truncate and quantize the intermediate data through hardware and software co-design, in which we obtained the data range of each stage from the software and quantize them within the range with different data width. The final power-friendly quantization bit width is shown in Fig. 5(a).

IV. DSCNN Architecture and Framewise Incremental Computation Technique

A. Architecture and Hardware Aware NN Training

As mentioned in Section II-B, the NN algorithm is a big challenge because of the huge computations and storage of CNN and DNN. Here, we choose a lightweight DSCNN [4], [23] and apply hardware-aware training to obtain a power-friendly binarized NN architecture.

Fig. 6 shows the architecture of DSCNN and its comparison with a traditional CNN, where this DSCNN is the second layer from our NN topology (see Fig. 7). In a standard convolution,

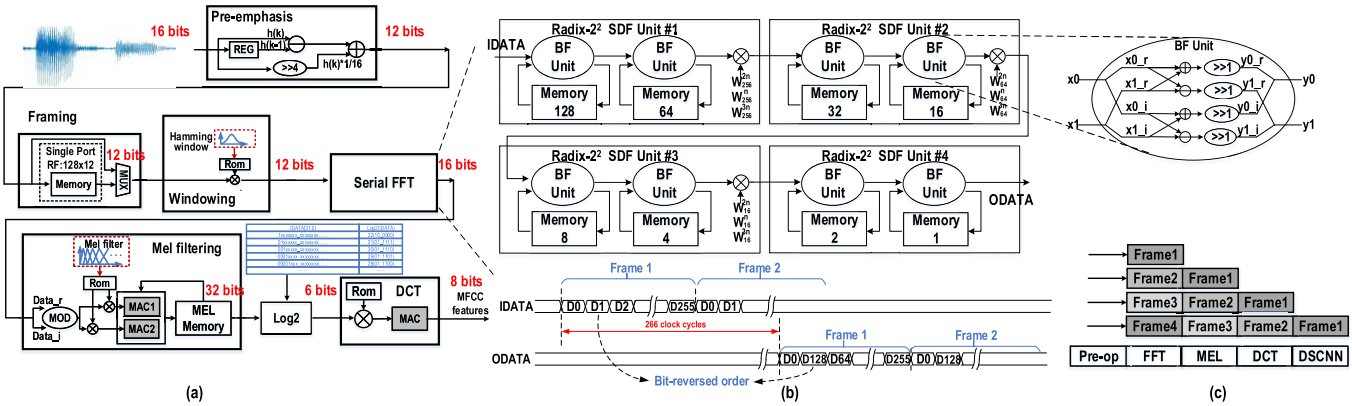


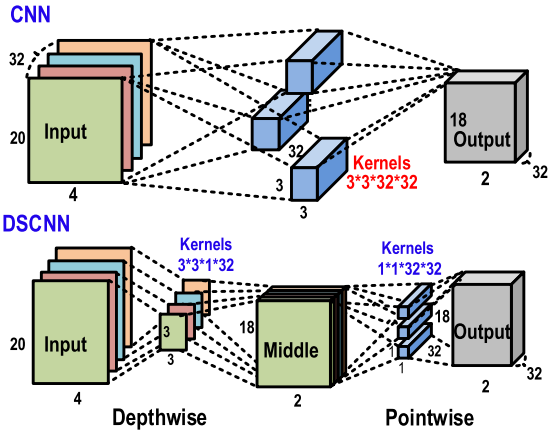
Fig. 5. (a) Overall architecture of MFCC extractor. (b) Implementation of serial FFT. (c) Four-frame pipeline scheme.

TABLE I
LUT OF OUR LOG_2 OPERATION

I [31:0]	Log2(I)
1xxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	32 (6'b10 0000)
01xx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	31 (6'b01 1111)
001x xxxx xxxx xxxx xxxx xxxx xxxx xxxx	30 (6'b01 1110)
0001 xxxx xxxx xxxx xxxx xxxx xxxx xxxx	29 (6'b01 1101)
……	……
0000 0000 0000 0000 0000 0000 0000 **1**xxx	4 (6'b00 0100)
0000 0000 0000 0000 0000 0000 0000 01xx	3 (6'b00 0011)
0000 0000 0000 0000 0000 0000 0000 001x	2 (6'b00 0010)
0000 0000 0000 0000 0000 0000 0000 0001	1 (6'b00 0001)

multiple 3-dimensional convolutional kernels slide on the input feature maps, generating multiple output feature maps by multiplication and accumulation, which is computation-intensive with a large data storage. On the other hand, DSC achieves a similar effect by using depthwise convolution and pointwise convolution. In the depthwise convolution, multiple 2-dimension convolutional kernels slide on each channel of the input feature map to filter features. Pointwise convolution, which is actually a simple 1×1 convolution, fuses the output of the depthwise layers though a line combination. Compared to a standard convolution, the depthwise convolution reduces the computations and storage both by $7\times$ regarding the specific layer shown in Fig. 6.

To choose a proper and energy-efficient NN architecture, we perform several rounds of NN training/testing by Python. First, we train a KWS demo released by Google for our task of detecting one/two keywords, with the DSCNN topology initially set based on "Hello Edge" [4]. Then, we scale the NN depth and width and binarized both the weights and activations until achieving target accuracy. The extensive NN architecture exploration of "Hello Edge" [4] offers a good reference for our design. Second, during the NN hardware design, we adopt a hardware aware NN training to tune the NN architecture slightly, to let it fit our proposed framewise incremental computation technique (Section IV-C). This requires specific modifications of the NN as follows: 1) we disallow padding in each step during the NN training to avoid inconsistent results in the adjacent convolution operations. 2) A stride of 2 is only used in the pooling layer in order to scale the size of the last



CNN vs DSCNN		
Mode	**#Mem**	**#Ops**
Traditional CNN	(9*32)*32=9216	9*32*18*2*32=331.8K
DSCNN	(9+32)*32=**1312**	(9*32+32*32)*18*2=**47.2K**
Reduced	**×7**	**×7**

Fig. 6. Algorithm of DSCNN and comparison with traditional CNN.

fully connected layer, while the stride of 1 is used in other layers, which is consistent with the input frame update stride. The reason for avoiding a stride of 2 is to save the intermediate data storage, which will be explained in detail in Section IV-C. Finally, we verify the RTL outputs of every NN layers to make them consistent with the Python output from the 10th frame data input, because the first convolution layer computes ten frames of inputs at once so that the result becomes correct until the 10th frame by our framewise incremental computation.

The whole DSCNN topology for 1–2 KWS is shown in Fig. 7, which comprises five layers (Conv + depthwise-Conv + pointwise-Conv + Pooling + FC). Each layer except the pooling is followed by a batch normalization (BN) [34] and sign activation. Input data are 10 dimensions of 8-b MFCC features, while the intermediate data and weights are both 1 b. For two-word detection, the number of total coefficients (weights) is 3456, and the number of computations is 150 496. It is composed of 102 400 8-b multiply accumulates (MACs) and 48 096 1-b MACs. The only difference between one-word and

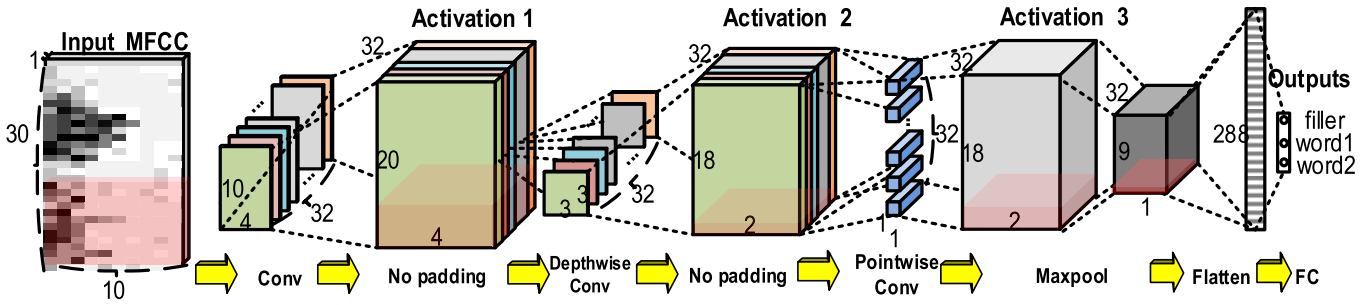


Fig. 7. Architecture of the customized binarized DSCNN for 1–2 keyword spotting.

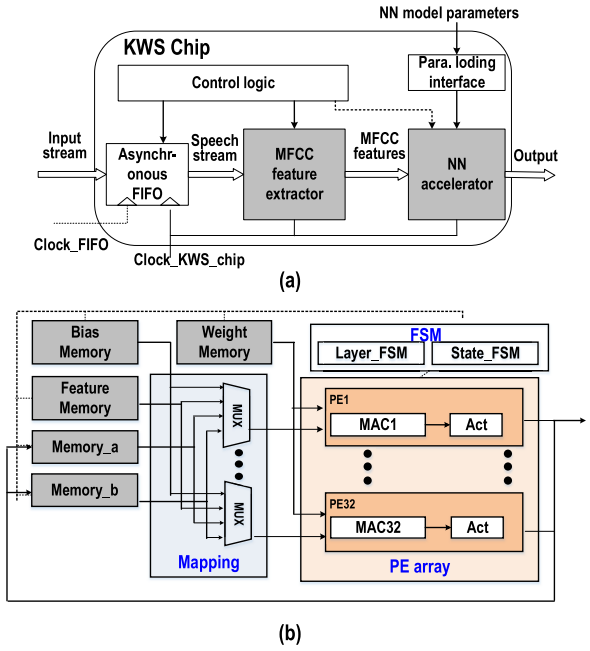


Fig. 8. KWS chip hardware architecture. (a) Whole chip architecture. (b) NN accelerator architecture.

two-word detection is the dimension of the FC layer, that is, 288×2 for one-word and 288×3 for two-word, respectively.

B. NN Hardware Design

The whole μKWS hardware architecture is shown in Fig. 8(a). It is composed of MFCC, NN, and interfaces. The NN accelerator interacts with the rest of the system as follows. Initially, the trained NN model parameters are loaded into the NN accelerator through an SPI interface. When the chip starts to work, the input audio stream (sampled by an outside ADC) enters the MFCC feature extractor through an asynchronous FIFO. Then, the generated MFCC features are fed into the NN accelerator for classification.

The hardware architecture of the NN accelerator is shown in Fig. 8(b), composed of a processing element (PE) array, a mapping module, a control module and five memory blocks of weight memory, bias memory, input feature memory, and two intermediate data memory blocks of Memory_a and Memory_b. The PE array consists of 32 PEs, each of which is responsible for the computation of one output channel, including multiplication and accumulation, BN, and activation. BN is relatively complex in multiple-bit NN, but it could be simplified to just adding an offset as (2) [38]. Here, γ, β, μ_{B}, and δ_{B} are BN parameters from floating-point Python training, and offset is the equivalent BN value after binarization. Thus, in our design, we just need an additional cycle to accumulate this offset

$$\begin{aligned}&\mathrm{Binarize}(\mathrm{BatchNorm}_{\gamma,\beta}(x))\\&= \mathrm{Binarize}\left(x-\mu_B+\frac{\sqrt{\sigma_B^2}}{\gamma}\beta\right) = \mathrm{Binarize}(x+\mathrm{offset})\end{aligned} \tag{2}$$

To execute different layers in the NN hardware, we design a finite-state machine (FSM) and a mapping module. The mapping module is responsible for selecting different data sources into the PE array in different states, which is mainly composed of some MUX logics. The control module is mainly composed of two nested FSMs. The upper FSM controls layer jumping, indicating which layer is being calculated by the processor, while the lower FSM controls the specific behaviors of the PE array, mapping module, and each storage block, such as data loading, accumulation, offset, output, and so on.

There are two levels of the memory hierarchy in our design: customized tri-latch-based memory and local registers in PEs. We mainly use input reuse (InR, an input is used by multiple kernels in standard convolution and pointwise convolution), output reuse (OutR, an output resides on local registers during the entire computational process), and output channel parallelism (every PE compute for one output channel). For the standard convolution in the first layer, the accelerator fetches 32 weights from weight memory, and 1 feature from feature memory in every cycle since one feature can be reused by 32 kernels. For the depthwise convolution, the accelerator fetches 32 weights from weight memory and 32 inputs from intermediate memory in every cycle since no InR is in this layer. For the pointwise convolution, the accelerator fetches 32 weights from weight memory in every cycle and 32 inputs from intermediate memory in every 32 cycles. Therefore, the memory bandwidth is sufficient to keep all PEs busy. Plus, the accumulation results are always stored in the local registers inside PEs.

A unified PE is designed to execute either 8-b [see Fig. 9(a)] or 1-b [see Fig. 9(b)] computations in our NN accelerator. Here, A[7:0] is the input data, W is weight, and Acc[14:0] is the accumulation register. For Mode $=$ 0 in 1-b mode,

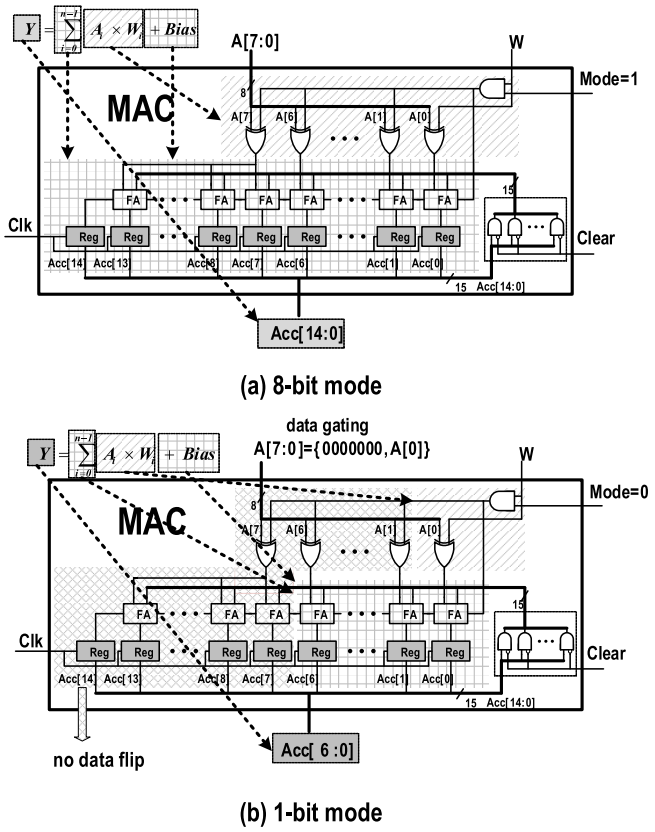


Fig. 9. Configurable PE. (a) 8-b mode or (b) 1-b mode.

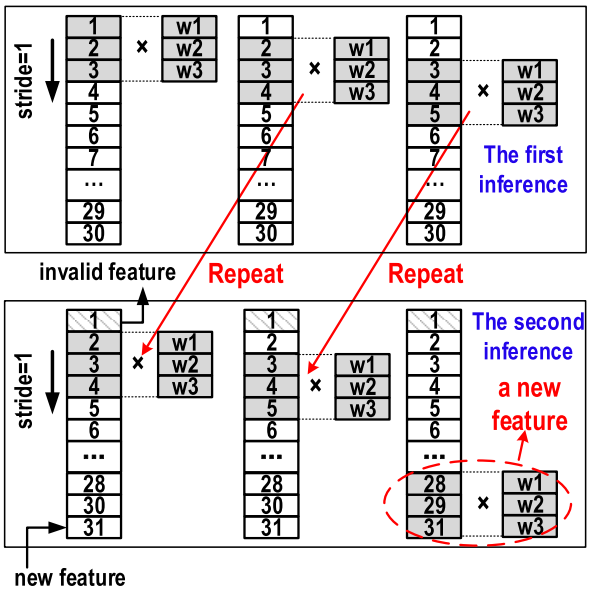


Fig. 10. Illustration of repetitive calculations in convolutional layer with stride = 1.

as shown in Fig. 9(b), the unused gates and registers are gated by setting the significant bits to zero. For Mode = 1 in 8-b mode, as in Fig. 9(a), when the weight is negative, the operation of subtracting A[7:0] is performed as adding the 2's complement of A[7:0]. It is implemented by the XOR of A[i] ($i = 1, \ldots, 6$) with W=1 (representing "−1") and adding a carry_in for the full adder.

C. Framewise Incremental Computation Technique to Eliminate Redundant Storage and Computations

The generated MFCC features are essentially a data stream sent to the NN, which processes multiframe data and updates the results for each frame. Fig. 10 shows a simplified illustration of the convolution operations in KWS; here, the weight kernel is represented by w1–w3 for simplicity with stride = 1. It can be seen that all the convolutions in the second inference are the same as those in the first inference except for the last one because only one frame of data is updated in the second inference. Thus, it is not necessary to store all the 30 frames of data for convolution; we only need to store Hk frames of data (Hk is the kernel's height and Hk is 3 in this example), which is just enough to convolve with the kernel and update a frame of data to the next layer.

Therefore, we propose a framewise incremental computation technique to eliminate redundant computations. To implement it, special data access order is scheduled, shown in Fig. 11 for a stride of 1 and kernel size of 3. In the kth inference, data corresponding to address a0, a1, and a2 are fetched orderly to convolve with the kernel, with the convolution result stored in the intermediate storage r1. In the $(k+1)$th inference, a new frame (31st) overwrites the oldest one (28th), and thus, the data in addresses of a1, a2, and a0 are fetched to convolve with the kernel in the correct order. And

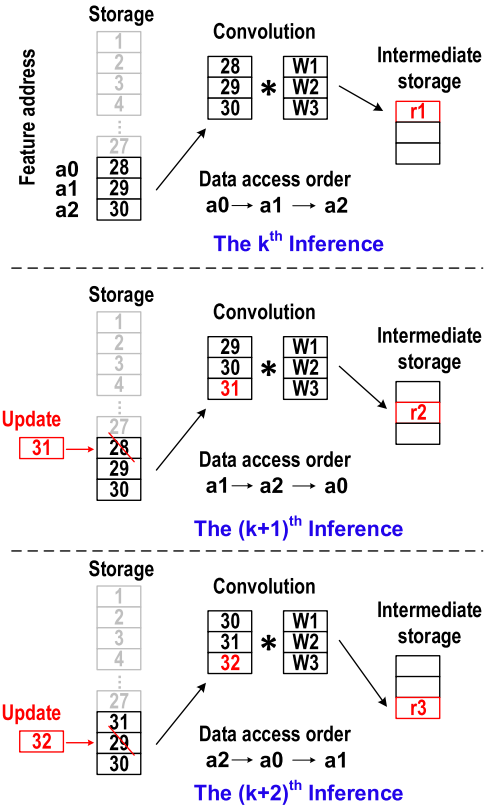


Fig. 11. Illustration of the special data access order for framewise incremental computation.

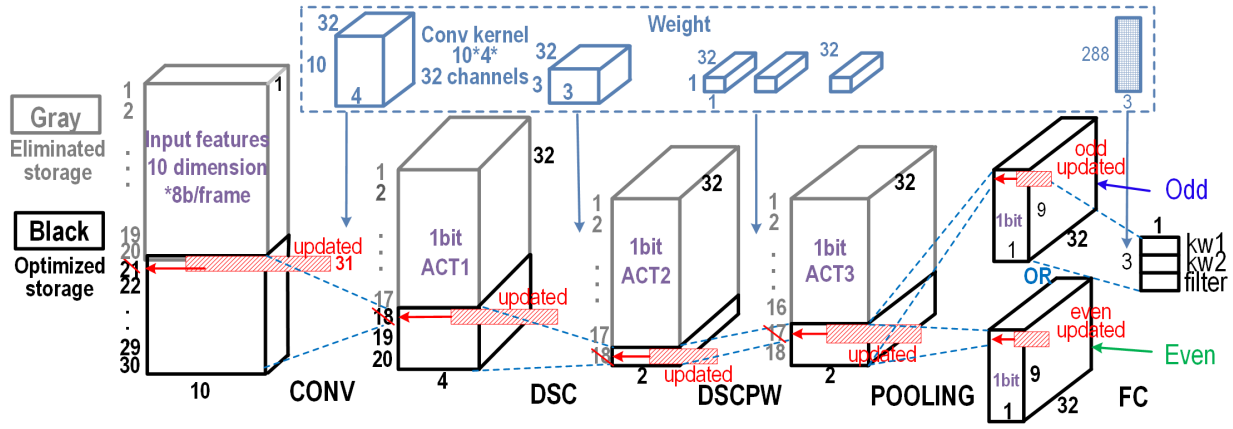


Fig. 12. Framewise incremental computation to eliminate redundant storage and computations in our DSCNN.

in the $(k+2)$th inference, data in addresses of a2, a0, and a1 are fetched in order, since the 29th frame is replaced by the new 32nd frame. It forms a circular data access pattern according to the updated frame.

On the other hand, for the case of stride $= 2$, the repetitive calculations occur every other inference. Therefore, the intermediate storage should be doubled to store the odd and even convolutions separately. This is the reason for our network topology design not using stride $= 2$ or more unless necessary. As described in Section IV-A, a stride of 2 is only used in the pooling layer in order to downsize the fully connected layer. Other processing is the same as the case of stride $= 1$.

Our framewise incremental computation significantly reduced the data storage and the related data computations. Its effect is shown as Fig. 12, where the gray boxes are the eliminated data storage for the 1st–4th layers. It can be seen that only 10, 3, 1, and 2 rows of data are stored according to the kernel sizes of 10×4, 3×3, 1×1, and 2×2 (pooling kernel), respectively. The memory and calculation are significantly decreased for each layer except for FC layers. The pooling results (which are also the input data for the FC layer) are stored separately for odd and even frames because it has a stride of 2. Then, all the data in the odd or even pooling output storage blocks are sent, in turn, to the FC layer to calculate the final KWS classification results. On the other hand, the input data size of the FC layer is not decreased, which collects all the computation results regarding the previous 29 frames and the new one. Therefore, the final FC layer is able to compute the data related to the whole word (30 frames) to get a correct classification result. The number of computations per inference after optimization is reduced to 8736 (5120 8-b MACs and 3616 1-b MACs). In total, 71.4% of the data storage and 94.3% of computations are eliminated by this method, without detection accuracy loss.

V. Near-Threshold Voltage Circuit Design

The previous design techniques enable the whole chip to work at an ultra-low frequency (40 kHz) while ensuring the frame rate. Low frequency, in turn, enables the full chip to work at NTV, which further reduces power consumption a lot.

For the NTV circuit, challenges exist in the memory design and leakage power control because leakage power becomes dominant at 40 kHz. We use the extra-high-voltage threshold (EHVT) standard cells to reduce the leakage power, but memory does not benefit from it. SRAM generated by the memory compiler is unsuitable for near-threshold voltage operation because of three reasons. First, it could not work at the low voltage of NTV. Second, its leakage power is too high, which exceeds our power budget. Third, the SRAM compiler could not generate memory with very small sizes, which are needed in our design. Our main memory blocks are shown in Fig. 13(a), with four dual-port and five single-port memory blocks with different sizes for MFCC and NN, which account for a dominant proportion of the total static power. Here, we do not count very small memory blocks, such as the bias memory, which are implemented in an easy way of using D flip-flop (DFF) here.

Since compiler-based SRAM is not applicable for NTV design, the standard cell memory (SCM) is a good way to store data at NTV [39]. We further design a customized tristate latch-like memory for low-leakage NTV applications. Fig. 13(b) shows the schematic and layout of the tristate latch as the basic memory cell, which consists of a pair of coupled inverters as a storage element and two tristate gates as input and output controllers. Based on a conventional latch cell, we use a tristate gate to replace the original inverter as the output and eliminate the internal inverters. When composing a memory block, for example, a 128×32 memory block as Fig. 13(c), it is divided into four 32-word blocks. Our advantage lies in that the output pins of many tristate latches in the same column can connect directly to constitute a read bitline like SRAM, which avoids large-fan-in multiplexers in [39]. In addition, to solve the glitch problem in decoding, we use the positive clock phase to decode and the negative clock phase to write the data. Compared with an equivalent size of 2-kB memory generated by the SRAM compiler, our customized memory reduces the static power by $12\times$, from 4.29 to 0.35 μW.

We also compare our tristate latch with a traditional latch and a traditional DFF from standard cells for the area, leakage,

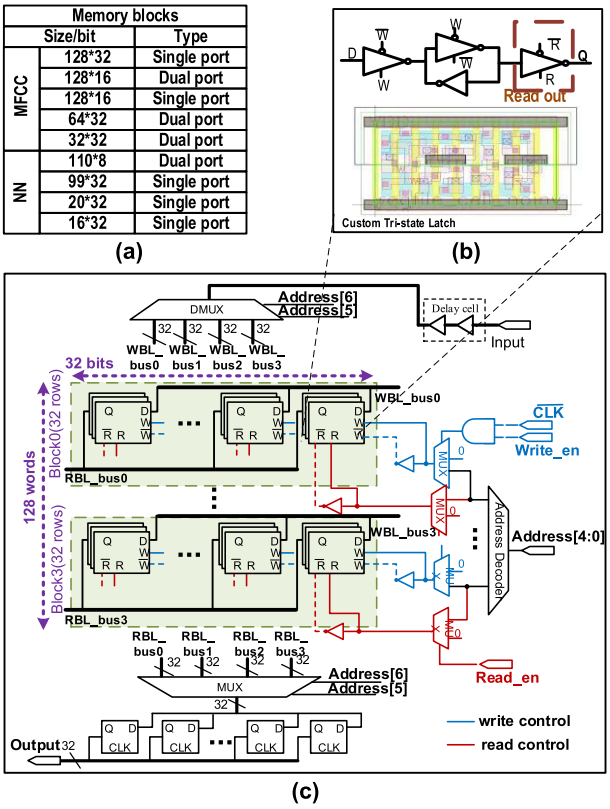


Fig. 13. Customized tristate latch-based memory design. (a) Memory blocks with different sizes in KWS chip. (b) Schematic and layout of the customized tristate latch-based memory. (c) Structure of a 128 × 32 memory block design.

and dynamic power, as shown in Fig. 14. Both the leakage and dynamic power are simulated using HSPICE with parasitic *RC* at TT corner, 0.5 V, and 25 °. Whereas DFF-based memory is more straightforward and may eliminate race conditions, the latch-based implementation is more power efficient with less area [39]. Our tristate latch has a smaller area than a standard latch, and its leakage is only about one-third of the latch's, which is helpful in our low-frequency design. Its dynamic power also shows advantages over the latch and the DFF. To be clear, these comparisons do not include the speed because it is not a concern in our ultra-low speed circuit design.

VI. Chip Implementation and Measurement Results

Fabricated in 28-nm CMOS technology, the die photograph and chip specifications of our KWS chip are shown in Fig. 15(a). The chip achieves a minimum power of 0.51 μW at 0.41 V, 40 kHz, with an on-chip 2-kB memory and a core size of 0.44 × 0.52 mm^2. Here, 30 fabricated chips are obtained for chip measurement. The testing environment is shown in Fig. 15(b), mainly composed of an oscilloscope, a power supply, and current measurement equipment, a PC, a testing PCB board with our chip on a socket. The measured KWS_output waveforms for one-keyword detection are shown in Fig. 15(c), where the Lable_start and Lable_end indicate the beginning and end of a keyword. A KWS_output pulse is generated shortly after Lable_end.

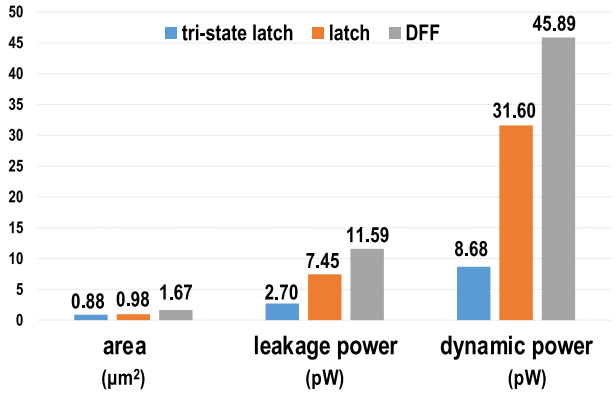
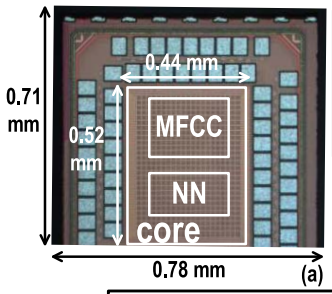


Fig. 14. Area, leakage, and dynamic power comparison of tristate latch, latch, and DFF.



Process	28 nm CMOS
Voltage	0.41V
Frequency	40kHz
Core Size	0.44×0.52 mm²
Die Size	0.78×0.71 mm²
Memory	2KB
MFCC power	0.34 μW
NN power	0.17 μW

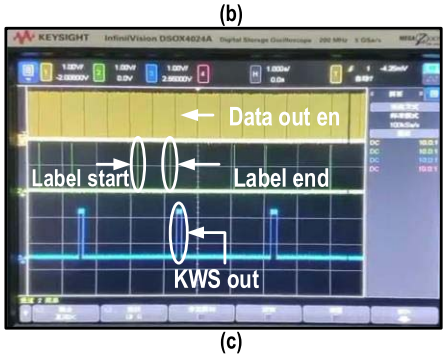


Fig. 15. (a) Die photograph and chip specifications. (b) Testing environment. (c) Measured KWS_output waveforms with Label_start and Label_end (for one-keyword detection).

A. Power Measurement

The minimum supply voltage distribution of the 30 chips is measured [see Fig. 16(a)]. The left y-axis related to the histogram represents the number of chips in each voltage range, and the right y-axis represents power consumption. It can be seen that the chips operate in a range of 0.41–0.48 V. Chips at 0.41 V achieve an average power consumption as low as 0.51 μW. To demonstrate the benefits of near-threshold

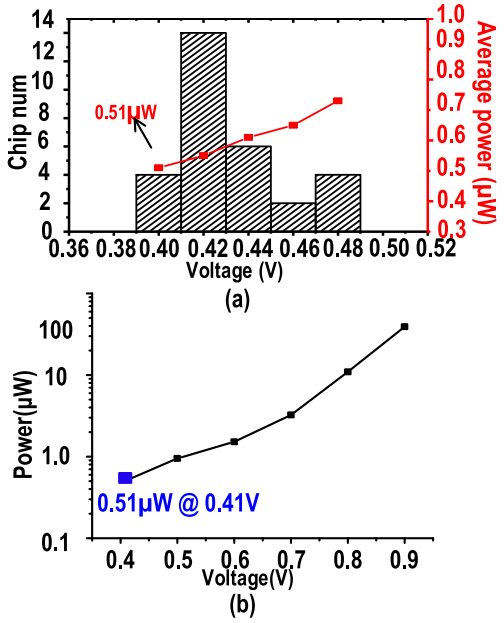


Fig. 16. (a) Functional minimum voltage distribution and corresponding power of 30 chips. (b) Power versus voltage of one typical chip.

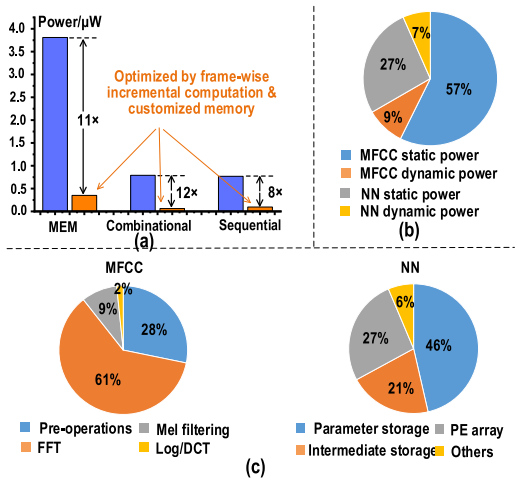


Fig. 17. (a) Power reduction by framewise incremental computation and custom memory. (b) Power break down according to static and dynamic power. (c) Power break down of MFCC (left) and NN (right) according to blocks.

design, Fig. 16(b) shows the power consumption of a typical chip with the supply voltage scaling down from normal to near-threshold voltage. The power consumption of the chip at 0.41 V is nearly 80× lower than that at 0.9 V. For a typical voltage of 0.6 V, its power consumption is 1.52 μW.

Fig. 17(a) shows power reduction by framewise incremental computation technique and customized memory. The power of memory, combinational elements, and sequential elements are reduced by 11×, 12×, and 8×, respectively, compared with the original nonoptimized implementation. Fig. 17(b) shows the proportions of the static and dynamic power of the chip. The total static power accounts for 84% (MFCC 57% plus NN 27%) of the total power. Fig. 17(c) shows the power break down according to blocks, where the FFT dominates

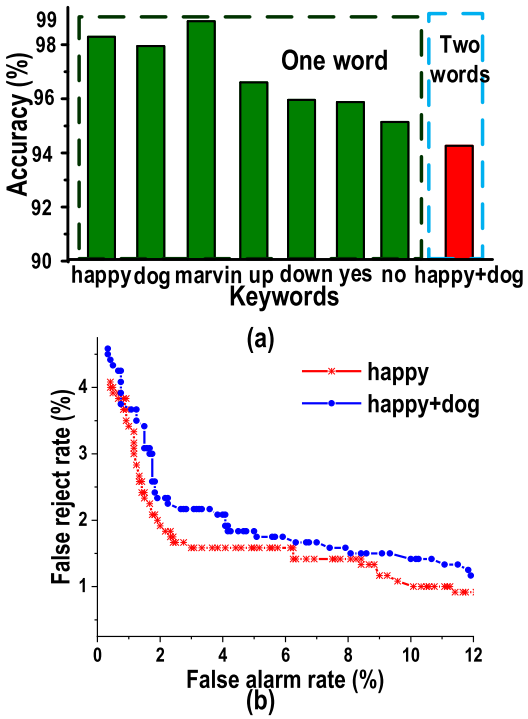


Fig. 18. (a) Accuracy and (b) ROC curves for one-word detection and two-word detection.

the power in MFCC, and the storage memory dominates the power in NN.

B. KWS Performance Evaluation

Here are some details about the training and testing of the KWS system. First, the label of each keyword from the GSCD data set appears at the end of the keyword. It is slightly different between software training/testing and HW testing because SW training has the whole keyword as input, while HW inputs the audio with only one frame of data per 16 ms. Therefore, our test audio is long audio made by concatenating the single 496-ms keyword data from the test set with filling in noise clips or zero of 1–2 s between words. Second, for the hardware testing of accuracy, if the NN fails to generate a correct signal within one frame (32 ms) near the label end signal, it is considered as a wrong detection. Since we did not implement the post-processing process in our chip, we compare the outputs of KW1, KW2, and filler, and the one with the largest output is chosen as the detection result.

For the accuracy testing of the KWS system, Fig. 18(a) shows the tested accuracy of one-/two-word detection aimed for keywords “happy,” “dog,” “marvin,” “up,” “down,” “yes,” and “no.” The accuracy of one-word detection (“happy”) is 98% and accuracy of two-word detection (“happy” and “dog”) is 94.6%. It can be seen that accuracy varies with different keywords; for example, a complex word, such as “marvin” achieves higher accuracy compared with a simple word of “no.” We tested the accuracies of all 10 words in GSCD (“Yes,” “No,” “Up,” “Down,” “Left,” “Right,” “On,” “Off,” “Stop,” and “Go”), and found that the average accuracy is 97.3%.

TABLE II
CHIP MEASUREMENT RESULTS AND COMPARISON WITH STATE-OF-THE-ART

	ISSCC 2017 [5]	VLSI 2018 [6]	VLSI 2019 [8]	ESSCIRC [7]	Our μKWS
Technology (nm)	40	28	65	65	28
NN Algorithm	DNN	CNN	LSTM	LSTM	DSCNN
Voltage (V)	0.63-0.9	0.57-0.9	0.6	0.575	0.41
Memory (kB)	270	52	65	32	2
Core Size (mm^2)	7.1	1.29	2.56	1.04	0.23
Frequency (MHz)	1.9	2.5	0.25	0.25	0.04
Latency (ms)	6.5	0.5-25	16	16	64
Frame shift (ms)	NA	NA	16	16	16
Keyword Number	10	1	10	4	1-2
Power (μW)	288	141	16.1	5[a]	0.51
Energy /frame/keyword (nJ)	NA	141	25.76	15	4.08@ 2words 8.16@1 word
Dataset	NA	TIDIGIT	GSCD	NA	GSCD
Accuracy	NA	96%	90.87%	91.2%	97.3%@1 word 94.6%@ 2words[b]

[a] This 5 μW chip did not include MFCC circuit.

[b] 97.3% is the average single-word accuracy for all the 10 keywords classes in the test set.

In addition to the accuracy, the receiver operator characteristic (ROC) [50] is a better metric to examine performance, which consists of a false reject rate and false alarm rate by tuning the threshold value for the output probability comparison in the post-processing process. The lower the FR per FA rate is, the better. The false alarm rate (also called false positive rate) is calculated as nonkeywords incorrectly classified divided by total nonkeywords. Our ROC is obtained by software simulation under a long audio sequence. The testing sequence is a 1-h-long audio sequence containing around 1000 filler words, 200 keywords, zeros, and some noise audios from the audio clips in the test set. When it recognizes a keyword, we record the moment and check the label text to find out whether there is a match within half of a word length (250 ms) near this moment. If the label is not matched, it is identified as an FA. Similarly, if it fails to generate "1" within 250 ms near the label, it is considered as a false reject. The FA rate is obtained by calculating the ratio of the number of FAs (including those triggered by noise) to the number of total filler words.

The ROC curves are shown in Fig. 18(b), where the best operating point is 1.9% FR @ 2% FA for one keyword and 2.2% FR @ 2.3% FA for two keyword. Alternatively, measuring FR at 0.5% FA rate is a typical operating point for practical applications [52]. Our FR/FA results are 4.5% FR @ 0.5% FA for one keyword and 4.6% FR @ 0.5% FA for 2 keyword. This ROC could be improved by enlarging the NN scale, such as adding more DS layers and/or increasing the channels in some layers.

Furthermore, we apply our model on LibriSpeech [47] and common voice [48] data set to evaluate its accuracy. We use a script to select the audios of the keyword parts in the data set and select some other audios as the filler words. The accuracies of the three data sets are shown in Fig. 19(a), which are all two-keyword detections of "Happy + dog." To examine the robustness to noise, we mix a certain level of noise in the original data set from GSCD and obtain the accuracies, as shown in Fig. 19(b). The horizontal axis represents the signal-to-noise ratio, and the vertical axis represents the accuracy. It can be seen that the accuracy deteriorates with the decrease of SNR.

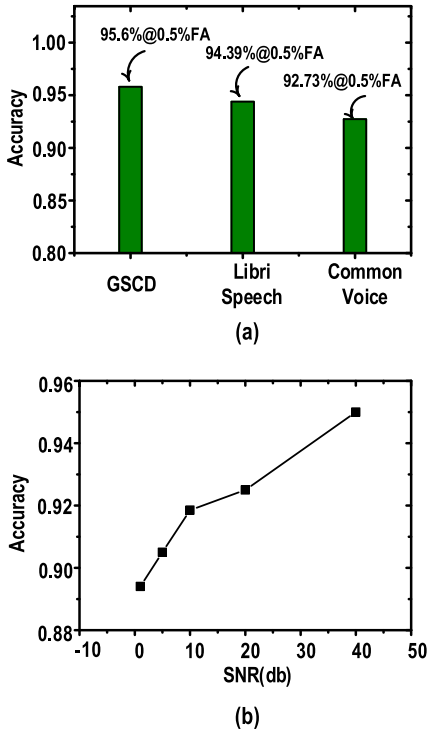


Fig. 19. (a) Accuracies of two-keyword detection of "Happy + dog" on three data sets. (b) Accuracies with 0-to-40-dB noise of "Happy + dog" on GSCD.

C. Comparison With State-of-the-Art Works

When comparing different KWS chips, the keywords number to be detected, and the aiming accuracy are two main factors that influence the power. Light tasks with fewer keywords need smaller NNs than heavy tasks with more keywords. Table II lists the comparison results with the state-of-the-art work, with different tasks of one-keyword, four-keywords, and ten-keywords. A binarized CNN for one-keyword wake-up application had 141-μW power at 2.5 MHz and 0.57 V, with 52 kB of SRAM [6]. Laika for four-keyword KWS task reduced power to 5 μW, but without including the MFCC circuit [7]. The DNN targeting for 10-keyword detection with a large on-chip weight memory of 270 kB consumed

TABLE III
TESTING ACCURACY VERSUS THE NUMBER OF KEYWORDS

Number of Keywords	1	2	3	4	>4
Accuracy	98%	94.6%	93.1%	91.7%	<90%

288 μW [5]. An LSTM accelerator for 10-keyword KWS with a storage of 105 kB reduced power to 16.11 μW for KWS with 90.8 % accuracy on the GSCD [8]. None of the above work reduced power consumption to the sub-μW region. Our μKWS also has the lowest energy/frame/keyword of 4.08 nJ@ 2 words and 8.16nJ @ 1 word, calculated by the multiplication of power and frameshift time per keyword, since one detection is generated every frameshift (usually 16 ms). Our cross-layer optimizations at the algorithm, architecture, and circuit level enable our KWS chip to operate at a minimum voltage of 0.41 V, with the smallest memory footprint and area compared with others. Aiming for 1–2 keyword wake-up tasks, we achieve the lowest power of 0.51 μW, which is 10–565 times lower than prior work.

To conduct a fair comparison with the four-keyword and ten-keyword tasks, we further train our current DSCNN model with more keywords, and the accuracies are shown in Table III. Its accuracy drops to 91.7% when detecting four-keywords, which is at the same level as 91.2% of Laika [7]. When coping with the ten-keyword task, our KWS losses its accuracy to less than 90%, which is worse than other chips [5], [8] due to its small NN size. On the other hand, we may compare MFCC's power instead, where our 0.34 μW outperforms the 7.14 μW of [8].

In addition, the latency of our KWS chip is larger than other work, which needs to be improved. Alternatively, we may combine FFT and Mel filtering as one pipeline stage, which may decrease the whole latency to 3 frames (48 ms here) with almost no extra effort.

VII. DISCUSSION

The false alarm rate is particularly important in the whole wake-up system in that the downstream system usually consumes over 1 mW. When the KWS circuit triggers to activate the entire system, it will bring much larger power consumption than the KWS circuit itself. A typical operating point for practical applications is 0.5% FA rate [52], where some software examples are: 3.5% FR at 0.5% FA [52] and 11% FR at 0.5% FA [53]. For hardware implementations of KWS, Bang *et al.* [5] and Giraldo *et al.* [7] [8] did not reveal the FA rate. Yin *et al.* [6] reported 0.01% FA with a long keyword of "Xiao'wei Xiao'wei," which is about 2×–4× longer than GSCD keywords, making the detection much easier. Plus, there is no revealing of the testing sequence for a wake-up in [6]. On the other hand, FA/hour is another useful measure of KWS output quality for real applications. However, it is highly dependent on the constituent of the testing sequences, while there are a vast of scenarios for different applications. As far as we know, there is no standard or guideline to set the testing sequence, and thus, it would not be fair to compare if the testing data details are not revealed.

Alternatively, a hierarchical cascaded system provides a good solution to minimize the energy consumption of the whole system, where a hierarchy of increasingly complex classifiers are used, each designed and trained for a specific subtask [51]. For instance, we may use a sub-μW KWS for an initial wake-up to substitute the VAD, then use a larger KWS circuit with improved FA/FR as the next stage wake-up, and finally, the automatic speech recognition system. In this case, system power and performance can be optimized.

IO power consumption should also be considered in a system. In total, there are 41 signal IOs in our demo chip, including 18 debug outputs, 16 audio signal inputs, 5 serial communication signals, and 2 global signals. When removing the unnecessary debug and output signals for practical application, we need to take the static power of 6 IOs and dynamic power of 17 IOs into account. The average dynamic power of an IO at an 8-kHz sampling rate and the static power of an IO is approximatively 16 and 8 nW, respectively; therefore, the total IO power is about 320 nW, making the total chip still be sub-μW.

Finally, some recent KWS techniques avoid the FE stage by using a deep learning network to model the raw waveform [44]–[46]. Google proposed to use convolution in time approach to model the raw waveform [44], whose features matched the performance of log-Mel filterbank energies when used with a convolutional, LSTM deep neural network (CLDNN). SincNet [45] replaced the first layer of the DNN for FE to process raw audio samples in speaker recognition. Later, it was applied to a KWS task [46], achieving 96.6% accuracy on the GSCD data set with 62K parameters. However, there has no hardware implementation of this raw waveform modeling in the KWS application yet.

VIII. CONCLUSION

An ultra-low-power KWS chip for wake-up application targeting for 1–2 keyword detection is fabricated in 28 nm with an area of 0.23 mm^2. First, a serial FFT circuit is used for FE, reducing data storage by 8× and power by 11× as compared with parallel FFT. Then, a binarized depthwise CNN is designed for classification, reducing data storage by 7× and computations by 7× as compared with the traditional CNN. Besides, the framewise incremental computation technique reduces computations by 17.4× and data storage by 3.5× as compared with traditional whole-word processing way. Lastly, we adopt a near-threshold voltage design in the whole chip with customized, low leakage memory blocks. With all these techniques, the chip achieves the lowest power of sub-μW for KWS with an operating frequency of 40 kHz, which is a good solution for always-ON KWS applications.

REFERENCES

[1] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2014, pp. 4087–4091.
[2] M. Shah, J. Wang, D. Blaauw, D. Sylvester, H.-S. Kim, and C. Chakrabarti, "A fixed-point neural network for keyword detection on resource constrained hardware," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS)*, Oct. 2015, pp. 1–6.
[3] Z. Wang, X. Li, and J. Zhou, "Small-footprint keyword spotting using deep neural network and connectionist temporal classifier," Sep. 2017, *arXiv:1709.03665*. [Online]. Available: http://arxiv.org/abs/1709.03665
[4] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," Nov. 2017, *arXiv:1711.07128*. [Online]. Available: http://arxiv.org/abs/1711.07128

[5] S. Bang *et al.*, "A 288 μW programmable deep-learning processor with 270 KB on-chip weight storage using non-uniform memory hierarchy for mobile intelligence," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 250–251.

[6] S. Yin *et al.*, "A 141 UW, 2.46 PJ/neuron binarized convolutional neural network based self-learning speech recognition processor in 28 nm CMOS," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2018, pp. 139–140.

[7] J. S. P. Giraldo and M. Verhelst, "Laika: A 5 μW programmable LSTM accelerator for always-on keyword spotting in 65nm CMOS," in *Proc. IEEE Eur. Solid State Circuits Conf. (ESSCIRC)*, Sep. 2018, pp. 166–169.

[8] J. S. P. Giraldo, S. Lauwereins, K. Badami, H. Van Hamme, and M. Verhelst, "18 μW SoC for near-microphone keyword spotting and speaker verification," in *Proc. IEEE Symp. VLSI Circuits (VLSI-Circuits)*, Jun. 2019, pp. C52–C53.

[9] T. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *Proc. Annu. Conf. Int. Speech. Commun. Assoc. (INTERSPEECH)*, Sep. 2015, pp. 1478–1482.

[10] X. Chen, S. Yin, D. Song, P. Ouyang, L. Liu, and S. Wei, "Small-footprint keyword spotting with graph convolutional network," in *Proc. IEEE Autom. Speech Recognit. Understand. Workshop (ASRU)*, Dec. 2019, pp. 539–546.

[11] J. R. Rohlicek, W. Russell, S. Roukos, and H. Gish, "Continuous hidden Markov modeling for speaker-independent word spotting," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, May 1989, pp. 627–630.

[12] R. C. Rose and D. B. Paul, "A hidden Markov model based keyword recognition system," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, Apr. 1990, pp. 129–132.

[13] J. G. Wilpon, L. G. Miller, and P. Modi, "Improvements and applications for key word recognition using hidden Markov modeling techniques," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, May 1991, pp. 309–312.

[14] M. Silaghi and H. Bourlard, "Iterative posterior-based keyword spotting without filler models," in *Proc. IEEE Autom. Speech Recognit. Underst. Workshop (ASRU)*, Dec. 1999, pp. 213–216.

[15] M. Silaghi, "Spotting subsequences matching an HMM using the average observation probability criteria with application to keyword spotting," in *Proc. Nat. Conf. Artif. Intell. (AAAI)*, Apr. 2005, pp. 1118–1123.

[16] K. Ando *et al.*, "BRein memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 TOPS at 0.6 W," *IEEE J. Solid-State Circuits*, vol. 53, no. 4, pp. 983–994, Apr. 2018.

[17] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, "An always-on 3.8 J/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 158–172, Jan. 2019.

[18] S. Choi, J. Lee, K. Lee, and H.-J. Yoo, "A 9.02 mW CNN-stereo-based real-time 3D hand-gesture recognition processor for smart mobile devices," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 220–222.

[19] K. Bong, S. Choi, C. Kim, D. Han, and H.-J. Yoo, "A low-power convolutional neural network face recognition processor and a CIS integrated with always-on face detector," *IEEE J. Solid-State Circuits*, vol. 53, no. 1, pp. 115–123, Jan. 2018.

[20] M. Price, J. Glass, and A. P. Chandrakasan, "A low-power speech recognizer and voice activity detector using deep neural networks," *IEEE J. Solid-State Circuits*, vol. 53, no. 1, pp. 66–75, Jan. 2018.

[21] D. Amodei *et al.*, "Deep speech 2: End-to-end speech recognition in English and Mandarin," Dec. 2015, *arXiv:1512.02595*. [Online]. Available: http://arxiv.org/abs/1512.02595

[22] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan, "Speech recognition using deep neural networks: A systematic review," *IEEE Access*, vol. 7, pp. 19143–19165, 2019.

[23] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," Apr. 2017, *arXiv:1704.04861*. [Online]. Available: http://arxiv.org/abs/1704.04861

[24] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," Nov. 2015, *arXiv:1511.00363*. [Online]. Available: http://arxiv.org/abs/1511.00363

[25] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," Feb. 2016, *arXiv:1602.02830*. [Online]. Available: http://arxiv.org/abs/1602.02830

[26] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[27] Y.-H. Chen, J. Emer, and V. Sze, "Using dataflow to optimize energy efficiency of deep neural network accelerators," *IEEE Micro*, vol. 37, no. 3, pp. 12–21, Jun. 2017.

[28] Z. Yuan *et al.*, "A 65 nm 24.7 J/frame 12.3 mW activation-similarity-aware convolutional neural network video processor using hybrid precision, inter-frame data reuse and mixed-bit-width difference-frame data codec," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2020, pp. 232–234.

[29] N. Dave, "Feature extraction methods LPC, PLP and MFCC in speech recognition," *Int. J. Advance Res. Eng. Technol.*, vol. 1, no. 6, pp. 1–4, 2013.

[30] L. Muda, M. Begam, and I. Elamvazuthi, "Voice recognition algorithms using Mel frequency cepstral coefficient (MFCC) and dynamic time warping (DTW) techniques," May 2010, *arXiv:1003.4083*. [Online]. Available: http://arxiv.org/abs/1003.4083

[31] M. Yang, C. Yeh, Y. Zhou, J. P. Cerqueira, A. A. Lazar, and M. Seok, "Design of an always-on deep neural network-based 1 μW voice activity detector aided with a customized software model for analog feature extraction," *IEEE J. Solid-State Circuits*, vol. 54, no. 6, pp. 1764–1777, Jun. 2019.

[32] B. R. Sekhar and K. M. M. Prabhu, "Radix-2 decimation-in-frequency algorithm for the computation of the real-valued FFT," *IEEE Trans. Signal Process.*, vol. 47, no. 4, pp. 1181–1184, Apr. 1999.

[33] J. Li, F. Liu, T. Long, and E. Mao, "Research on pipeline R22SDF FFT," in *Proc. Int. Radar Conf.*, Apr. 2009, pp. 1–5.

[34] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

[35] P. Warden. (2017). *Speech Commands: A Public Dataset for Single-Word Speech Recognition*. [Online]. Available: http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz

[36] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. OSDI*, vol. 16, Nov. 2016, pp. 265–283.

[37] W. Shan *et al.*, "A 510 nW 0.41 V low-memory low-computation keyword-spotting chip using serial FFT-based MFCC and binarized depthwise separable convolutional neural network in 28 nm CMOS," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2020, pp. 230–232.

[38] D. Miyashita, S. Kousai, T. Suzuki, and J. Deguchi, "A neuromorphic chip optimized for deep learning and CMOS technology with time-domain analog and digital mixed-signal processing," *IEEE J. Solid-State Circuits*, vol. 52, no. 10, pp. 2679–2689, Oct. 2017.

[39] A. Teman, D. Rossi, P. Meinerzhagen, L. Benini, and A. Burg, "Power, area, and performance optimization of standard cell memory arrays through controlled placement," *ACM Trans. Design Autom. Electron. Syst.*, vol. 21, no. 4, p. 59, May 2016.

[40] S. He and M. Torkelson, "A new approach to pipeline FFT processor," in *Proc. IEEE Symp. Parallel Distrib. Process*, Apr. 1996, pp. 766–770.

[41] S. Magar, S. Shen, G. Luikuo, M. Fleming, and R. Aguilar, "An application specific DSP chip set for 100 MHz data rates," in *Proc. Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, 1988, pp. 1989–1992.

[42] C.-H. Yang, T.-H. Yu, and D. Markovic, "Power and area minimization of reconfigurable FFT processors: A 3GPP-LTE example," *IEEE J. Solid-State Circuits*, vol. 47, no. 3, pp. 757–768, Mar. 2012.

[43] A. Wang and A. P. Chandrakasan, "A 180-mV Subthreshold FFT Processor Using a Minimum Energy Design Methodology," *IEEE J. Solid-State Circuits*, vol. 40, no. 1, pp. 310–319, Jan. 2005.

[44] T. N. Sainath, R. J. Weiss, A. W. Senior, K. W. Wilson, and O. Vinyals, "Learning the speech front-end with raw waveform CLDNNs," in *Proc. Interspeech*, Dresden, Germany, Sep. 2015, pp. 1–5.

[45] M. Ravanelli and Y. Bengio, "Speaker recognition from raw waveform with SincNet," in *Proc. IEEE Spoken Lang. Technol. Workshop (SLT)*, Dec. 2018, pp. 1021–1028.

[46] S. Mittermaier, L. Kürzinger, B. Waschneck, and G. Rigoll, "Small-footprint keyword spotting on raw audio data with sinc-convolutions," Nov. 2019, *arXiv:1911.02086*. [Online]. Available: http://arxiv.org/abs/1911.02086

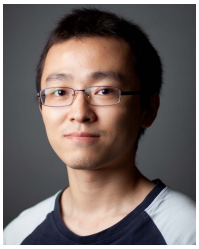
[47] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process (ICASSP)*, Aug. 2015, pp. 5206–5210. [Online]. Available: http//www.openslr.org/12/

[48] Mozilla. *Common Voice: An Open Source, Multi-Language Dataset of Voices That Anyone Can Use to Train Speech-Enabled Applications*. [Online]. Available: https://voice.mozilla.org/en/datasets
[49] G. Bi and E. V. Jones, "A pipelined FFT processor for word-sequential data," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 37, no. 12, pp. 1982–1985, Dec. 1989.
[50] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, 2006.
[51] K. Goetschalckx, B. Moons, S. Lauwereins, M. Andraud, and M. Verhelst, "Optimized hierarchical cascaded processing," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 4, pp. 884–894, Dec. 2018.
[52] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, 2015, pp. 4087–4091.
[53] R. Tang and J. Lin, "Deep residual learning for small-footprint keyword spotting," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2018, pp. 5484–5488.

Weiwei Shan (Member, IEEE) received the B.S. degree in microelectronics from Tianjin University, Tianjin, China, in 2003, and the Ph.D. degree in microelectronics from Tsinghua University, Beijing, China, in 2009.

She was a Visiting Professor with Columbia University, New York, NY, USA, from 2017 to 2019. She is currently a Professor with the National ASIC Center, Southeast University, Nanjing, China. Her research mainly focuses on variation resilient, adaptive VLSI circuits, ultra-low-power SoC design, and countermeasure techniques of security circuits. She has authored or coauthored over 50 technical articles in conferences and journals, including IEEE International Solid-State Circuits Conference (ISSCC), the JOURNAL OF SOLID-STATE CIRCUITS, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I: REGULAR PAPERS (TCASI)/IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II: EXPRESS BRIEFS (TCASII), IEEE TRANSACTIONS ON COMPUTER AIDED DESIGN OF INTEGRATED CIRCUITS & SYSTEMS (TCAD), and authorized over 25 invention patents.

Dr. Shan was a recipient of the 2014 State Scientific and Technological Progress Award, the 2017 A-SSCC Distinguished Design Award, and the 2018 GLSVLSI Best Paper Candidate.



Minhao Yang (Member, IEEE) received the Ph.D. degree in physics from ETH Zurich, Zürich, Switzerland, in 2015.

He was a Post-Doctoral Researcher with Columbia University, New York, NY, USA. He is currently a Collaborateur Scientifique with the École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland. His post-doctoral research was partly supported by the Early Postdoc Mobility Fellowship from the Swiss National Science Foundation. His research interests include ultralow-power inference sensing systems, event-driven sensors like spiking silicon retina and cochlea, and spike coding and processing.

Tao Wang received the B.S. degree in electronic engineering from Southeast University, Nanjing, China, in 2018, where he is currently pursuing the M.S. degree in electronic engineering.

His research mainly focuses on low-power and energy-efficient integrated circuit design.

Yicheng Lu received the B.S. degree in electronic engineering from Southeast University, Nanjing, China, in 2018, where he is currently pursuing the M.S. degree in electronic engineering.

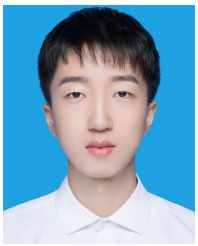
His researches mainly focus on keyword spotting circuit design and on-chip training.

Hao Cai (Member, IEEE) received the Ph.D. degree in electrical engineering from the Télécom Paristech, Université Paris-Saclay, Paris, France, in 2013.

From 2012 to 2014, he was involved in the European EUREKA Program CATRENE-RELY for high-reliability nanoscale integrated circuits and systems. He is currently a Faculty Member with the National ASIC System Engineering Center, Southeast University, Nanjing, China. His current research interests include circuit techniques for emerging technologies, ultralow-power VLSI, and reliability-aware design.

Lixuan Zhu received the B.S. degree in electronic engineering from Southeast University, Nanjing, China, in 2019, where he is currently pursuing the M.S. degree in electronic engineering.

His researches mainly focus on low-power IC design and machine learning.



Jiaming Xu received the B.S. degree in electronic engineering from Southeast University, Nanjing, China, in 2017, where he is currently pursuing the M.S. degree in electronic engineering.

His researches mainly focus on low-power integrated circuit design and information security circuit design.

Chengjun Wu received the B.S. degree in electronic engineering from Southeast University, Nanjing, China, in 2018, where he is currently pursuing the M.S. degree in electronic engineering.

His researches mainly focus on low-power integrated circuit design and adaptive voltage scaling.

Longxing Shi (Senior Member, IEEE) received the Ph.D. degree in microelectronics from Southeast University, Nanjing, China, in 1988.

He is currently a Professor and the Dean with the School of Electrical Science and Technology, Southeast University. He has published over 120 technical articles in conferences and journals. He has authorized over 200 Chinese invention patents and 12 U.S. patents. His research interests include system-on-chip design, RF, and mixed-signal integrated circuit design.

Jun Yang (Member, IEEE) received the B.S. and Ph.D. degrees in electronic engineering from Southeast University, Nanjing, China, in 1999 and 2004, respectively.

He is currently a Professor with the National ASIC System Engineering Research Center, Southeast University. He has authored and coauthored over 50 technical articles in conferences and journals, including IEEE International Solid-State Circuits Conference (ISSCC), Design Automation Conference (DAC), the JOURNAL OF SOLID-STATE CIRCUITS, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I: REGULAR PAPERS (TCASI), TCASI, and IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (TVLSI) Systems. He has authorized over 100 Chinese and U.S. invention patents. His current researches focus on SRAM design, in-memory computing, and near-threshold design.