electronics

MDPI

Article

A Configurable Accelerator for Keyword Spotting Based on Small-Footprint Temporal Efficient Neural Network

Keyan He, Dihu Chen and Tao Su *

School of Electronics and Information Technology, Sun Yat-sen University, Guangzhou 510006, China
* Correspondence: sutao@mail.sysu.edu.cn

Abstract: Keyword spotting (KWS) plays a crucial role in human–machine interactions involving smart devices. In recent years, temporal convolutional networks (TCNs) have performed outstandingly with less computational complexity, in comparison with classical convolutional neural network (CNN) methods. However, it remains challenging to achieve a trade-off between a small-footprint model and high accuracy for the edge deployment of the KWS system. In this article, we propose a small-footprint model based on a modified temporal efficient neural network (TENet) and a simplified mel-frequency cepstrum coefficient (MFCC) algorithm. With the batch-norm folding and int8 quantization of the network, our model achieves the accuracy of 95.36% on Google Speech Command Dataset (GSCD) with only 18 K parameters and 461 K multiplications. Furthermore, following a hardware/model co-design approach, we propose an optimized dataflow and a configurable hardware architecture for TENet inference. The proposed accelerator implemented on Xilinx zynq 7z020 achieves an energy efficiency of 25.6 GOPS/W and reduces the runtime by 3.1× compared with state-of-the-art work.

Keywords: keyword spotting; temporal convolutional neural network; configurable accelerator; mel-frequency cepstrum coefficient

Citation: He, K.; Chen, D.; Su, T. A Configurable Accelerator for Keyword Spotting Based on Small-Footprint Temporal Efficient Neural Network. *Electronics* **2022**, *11*, 2571. https://doi.org/10.3390/electronics11162571

Academic Editor: Ping-Feng Pai

Received: 13 July 2022
Accepted: 12 August 2022
Published: 17 August 2022

1. Introduction

Keyword spotting (KWS) is a task that aims to detect predefined keywords in a stream of audio signals. It is an important interface in human–machine interactions and usually acts as an always-on switch for more complex systems.

Over the past decade, with the success of deep learning, traditional Hidden Markov Models (HMMs) for KWS have been replaced by neural-network-based approaches [1,2], resulting in higher accuracy and better noise robustness. Since the fact that such tasks usually run on edge devices such as IoTs with restricted memory and computational resources, it remains a giant challenge to implement such complex structures of neural networks. To overcome this challenge, many deep neural network (DNN) accelerators for low-power speech recognition have been proposed. Giraldo et al. [3] employed a long short-term memory (LSTM) accelerator with the power consumption of 18.3 µW and supported 10 keywords in GSCD with an accuracy of 90.87%. Liu et al. [4] introduced a dual-mode KWS processor using a binary weight network (BWN), which could achieve a power of 10.8 µW under 22 nm process technology and could support up to 10 keywords with an accuracy of 87.9%. Shan et al. [5] designed a KWS chip using a binarized depthwise separable convolutional neural network (DSCNN) with an ultra-low power of 0.51 µW in 28 nm process technology implementation, achieving 94.6% accuracy on two words. Thus, hardware-friendly structures of neural network models, as well as their efficient deployment on embedded devices, should be considered by researchers in the meantime.

Recently, new network architectures based on temporal convolution have shown superior behavior in KWS tasks. Choi et al. [6] introduced the temporal convolutional residual network (TC-ResNet), a combination of temporal convolution and residual networks.

Unlike most CNN-based KWS approaches that receive features such as the mel-frequency cepstrum coefficient (MFCC) as a 2D input, TC-ResNet applies 1D convolution along the temporal dimension and treats the dimensions of the MFCC as input channels, which considerably reduces the number of operations while maintaining comparable accuracy. Paul et al. [7] proposed a configurable hardware architecture for TC-ResNet inference and achieved competitive metrics of power consumption and accuracy.

On the basis of TC-ResNet, Li et al. [8] introduced the idea of depthwise separable convolution and proposed a temporal efficient neural network (TENet). TENet is constructed by inverted bottleneck blocks (IBBs) with 1×1 pointwise (PW), 9×1 depthwise (DW), and 1×1 pointwise convolution. Benefitting from the PW and DW operators, the computational burden of TENet is further reduced. However, these new operators also raise great challenges for the dataflow scheduling and memory accessing of hardware architecture. Some accelerators for depthwise separable convolution have been proposed in recent years. Ref. [9] included parallel processing elements (PEs) for PW convolution and DW convolution, respectively, and implemented pipeline operations between convolutional layers. Refs. [10,11] designed the architecture of the PEs to support either DW, PW, or standard convolution operations. However, due to these accelerators being generally designed for image classification tasks, they cannot support temporal convolution networks well. As far as we know, there is no efficient hardware implementation of TENet-based neural networks.

In the rest of this paper, we describe the modification and optimization of the feature-extraction algorithm and the neural network model in Section 2. Our configurable accelerator architecture is presented in Section 3. The experimental results are shown in Section 4, and finally, the paper is concluded in Section 5.

2. Model Modification and Optimization

2.1. Simplification of the MFCC Algorithm

The traditional MFCC feature-extraction algorithm, as shown in Figure 1a, consists of several steps: firstly, applying pre-emphasis, framing, and windowing on the audio signals; then, transferring from the time domain to the frequency domain through the fast Fourier transform (FFT); lastly, executing mel filtering, logarithmic operation, and the discrete cosine transform (DCT). The output is the characteristic time and frequency feature matrix of the audio.

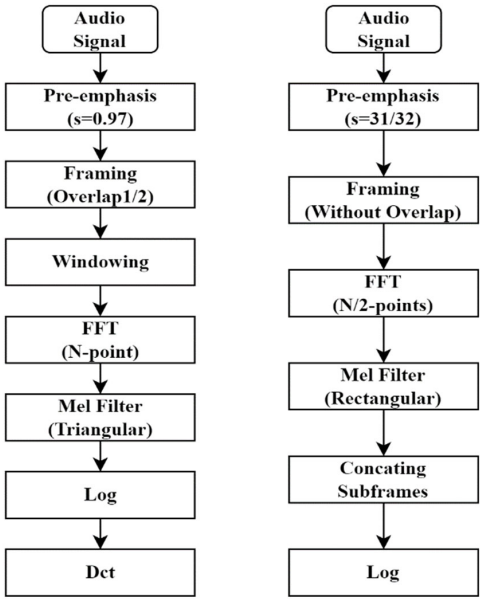

Figure 1. Processing flow of the (**a**) traditional and (**b**) simplified MFCC feature-extraction algorithm.

Feature extraction usually makes up 20–60% of the power consumption of a KWS system [12]. Most research focuses on the optimization of the feature-classification model, not on the simplification of feature extraction. Thus, we propose a simplified feature-extraction algorithm, a simplified MFCC. As shown in Figure 1b, we remove the windowing and DCT modules from the processing flow. Then, we set the frame length and step length of framing as a power of 2, so as to prevent executing redundant computation in the FFT stage. In hardware implementations, the cost of multiplication is much greater than that of addition, so we adjust the coefficient of the pre-emphasis to 31/32, to use shifting and addition operations instead of expensive multiplication. As described in [13],we replace the shape of triangular mel filters with rectangular ones, which has little effect on the KWS accuracy of the system while reducing the amounts of multiplication.

Furthermore, FFT operations take up most of the computational load of the MFCC algorithm, but directly decreasing the number of FFT points reduces the frame length, thereby increasing the number of frames of the feature matrix and improving the computational load of the neural network in the feature-classification part. To resolve this conflict, as illustrated in Figure 2, we propose a new framing method. In the traditional MFCC algorithm, the audio is sliced with a frame length of 32 ms, and neighboring frames share an overlap length of 16 ms. An audio file with a sampling rate of 16,000 Hz is divided into 61 frames per second, and each frame consists of 512 sampling points. In the simplified MFCC algorithm, the audio is sliced with a frame length of 16 ms without overlap. Thus, the audio is divided into 62 subframes per second, and each subframes contains 256 sampling points. The required amount of multiplications for the N-point FFT is ($N/2*log_2N$), so the number of computations during the FFT is reduced by more than half. After the FFT and mel-filtering stages, we add the feature vector of neighboring subframes; then, we obtain 61 feature vectors of 1 s audio. Each vector contains the information of a frame that is 32 ms in length, which maintains the same function as in the original method.

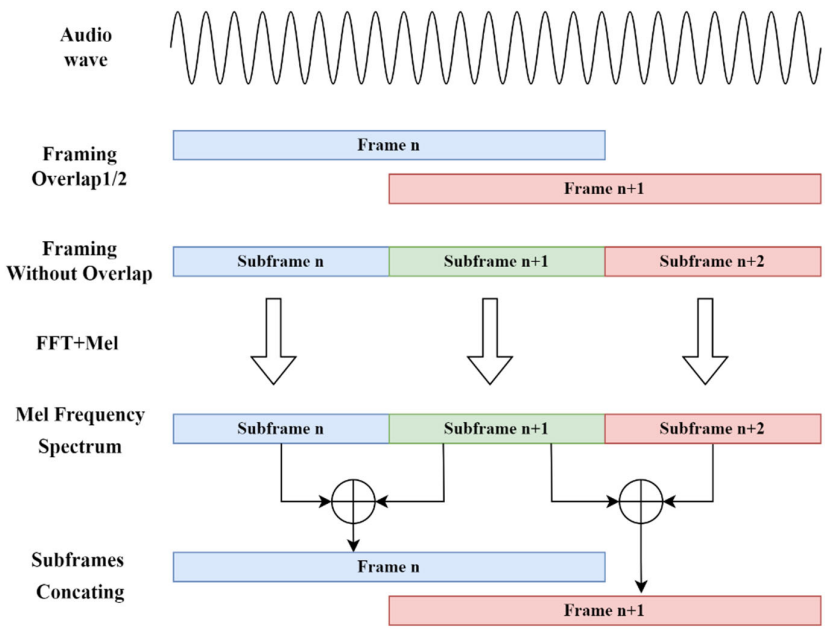

Figure 2. The new framing method of the simplified MFCC feature-extraction algorithm. Firstly, we frame the audio signals into subframes without overlap. Subframes are half the length of the complete frames. After the FFT and mel stages, the mel-frequency spectrum of neighboring subframes is added, and the frequency features of the complete frames are generated.

We use the traditional MFCC and the simplified MFCC to train the TENet model with the same parameters; the accuracy of the traditional MFCC is 96.07% while the accuracy of the simplified MFCC is 95.78%. With a cost of a negligible drop in accuracy, the

computational load of feature extraction can be reduced by 82% for multiplications and 66% for additions, as listed in Table 1.

Table 1. Computational cost of MFCC algorithm.

Algorithm Step	Traditional MFCC		Simplified MFCC	
	Add	**Mult**	**Add**	**Mult**
Pre-emph (s = 31/32)	16,000	/	16,000	/
Windowing	/	32,000	/	/
FFT [1]	281,088	140,544	126,976	63,488
Mel filtering [2]	31,354	31,354	7869	/
Subframe concatenating	/	/	1830	/
DCT	156,160	156,160	/	/
Overall	500,279	360,058	168,671	63,488

[1] The points of the FFT of the traditional MFCC are 512, and those of simplified MFCC are 256. [2] The shape of the mel filters in the traditional MFCC is triangular, while the shape of the mel filters in the simplified MFCC is rectangular.

2.2. Modification of the Neural Network Structure

The basic building blocks of TENet are IBBs, as shown in Figure 3a.

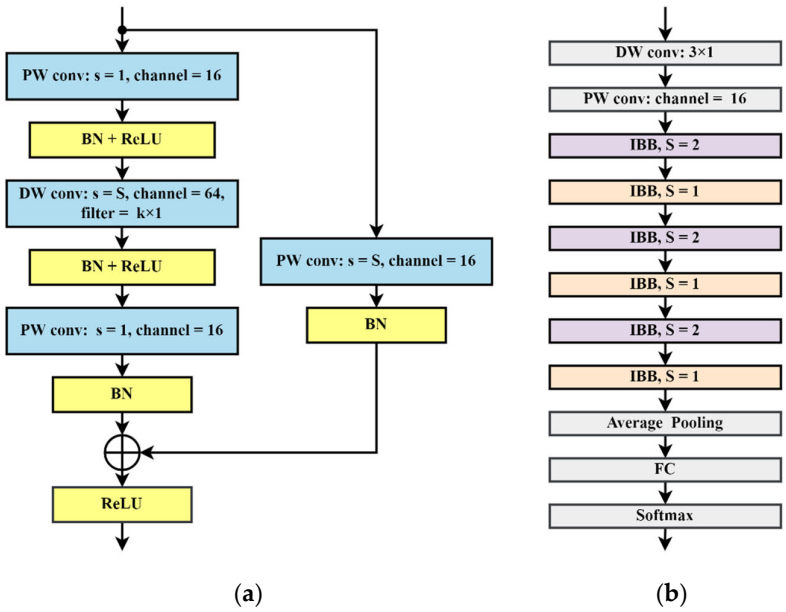

Figure 3. Basic building blocks of TENet (**a**) and structure of the modified TENet (**b**). BN denotes batch normalization, and "S" indicates stride.

The top and bottom PW convolution layers are the bottleneck layers of the block. The former PW layer aims to expand the number of channels so as to map the features into the high-dimensional regions, while the latter converts the features back to the low-dimensional regions for information fusion between channels and compacts the number of the channels for reducing the data transmission load between blocks. The k × 1 DW convolution layer is the expansion layer. As the key layer of temporal convolution, it performs the reception and fusion of features in the time dimension. The outputs of the main path are added to the residual path before the last activation layer. If the sizes of the input and output of a IBB do not match, a simple PW convolution layer followed by batch normalization (BN) at the residual path resamples the feature size, else the residual path directly forwards the input data.

We carefully select the expanding ratio between the bottleneck layer and the expansion layer as 4, in order to reach a trade-off between performance and efficiency. In consideration of this, we change the frame-length and step-length parameters in the feature-extraction part, and we adjust the shape of the DW convolution filter to 6×1 to achieve the same size of the time-domain receptive field.

Additionally, a depthwise separable convolution takes the place of the standard convolution in the first layer of the network, which further improves the calculation efficiency. As displayed in Figure 3b, our modified TENet starts with a depthwise separable convolution layer, consists of several IBB blocks, and ends up with the module of a pooling layer, a fully-connected (FC) layer, and a SoftMax layer.

2.3. Batch-Normalization Folding and Quantization for Hardware Deployment

As mentioned above, the convolution layers in a TENet are usually followed by BN layers, which normalize the output of the proceeding layer, make the process of network training converge faster, and allow a higher accuracy of the trained network to be obtained. For a trained network, the BN operation can be treated as a linear transformation with constant parameters. Thus, as shown in (1),

$$W_{inf} = \frac{\gamma W}{\sqrt{\sigma^2 + \varepsilon}} \; and \; \beta_{inf} = \frac{\gamma(b - \mu)}{\sqrt{\sigma^2 + \varepsilon}} + \beta, \tag{1}$$

The values of the weights and biases of the convolution layers before BN layers can be derived from trained parameters using a transformation. Parameters μ and σ denote the moving mean and variance of inputs over the training set, and γ and β are learned parameters, while W and b are the weight and bias of the proceeding Conv layer. Then, the BN layers can be removed from the network at inference time without loss of model accuracy.

To efficiently deploy the model on edge hardware, we execute layer-wise fix-point quantization [14] on the trained variables. As calculated in (2), we determine the integer part's bit width N based on the data range of each variable.

$$N = ceil\left(log_2\left(max\left(abs\left(V_{max}\right), abs\left(V_{min}\right)\right)\right)\right), \tag{2}$$

Then, using the method presented in (3), a floating-point value v_{float} is transferred to its signed integer 8-bit quantized representation v_q.

$$v_q = round\left(v_{float} * 2^{7-N}\right) / 2^{7-N}, \tag{3}$$

For feature maps in the network, we set the bit width of the integer part as 3, except for the MFCC inputs and the output of the FC layer, which we set as 4 and 5, respectively.

3. Accelerator Architecture

3.1. Architecture Overview

The architecture of the global system is shown in Figure 4. A multiply–accumulate (MAC) array is used to perform the convolutional computation and the remaining postprocessing, such as bias adding, average pooling, and activation function, which are executed by an output processing unit (OPU). Distributed memories surround the MAC array and the OPU, consisting of a weight memory (WMEM), a bias memory (BMEM), and a set of feature memories (FMEM0/1/2), to meet the bandwidth requirements and the residual topology of the network.

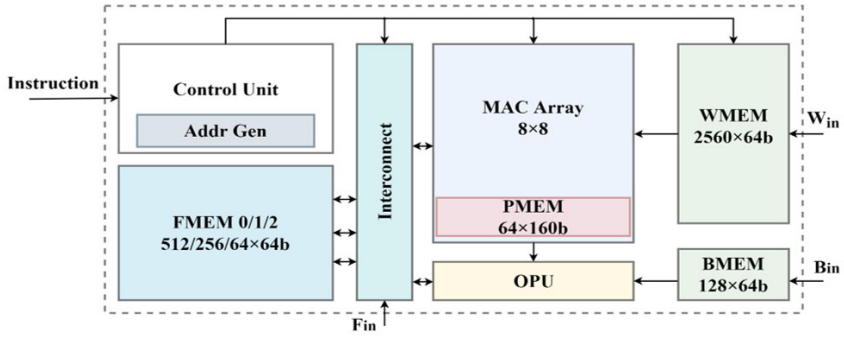

Figure 4. Overview of the architecture of the accelerator.

3.2. Dataflow and MAC Array

The MAC array is the primary processing unit of the accelerator. To execute the computationally intensive convolution layers, the array contains 64 MAC units in the shape of an 8 × 8 grid. A detailed schematic view is displayed in Figure 5. Each MAC unit contains a weight register and a result register. The input result from the left-side unit is accumulated by the product of the weight and the input feature, and the accumulated result is stored in the result register for the access of the right-side unit. A calculating enable signal (cal_en) decides whether the product is accumulated or not. The results of the calculation propagate from left to right along the row of the array. When they reach the rightmost unit, they are stored in the partial sum memory (PMEM) or fed into the output processing unit (OPU).

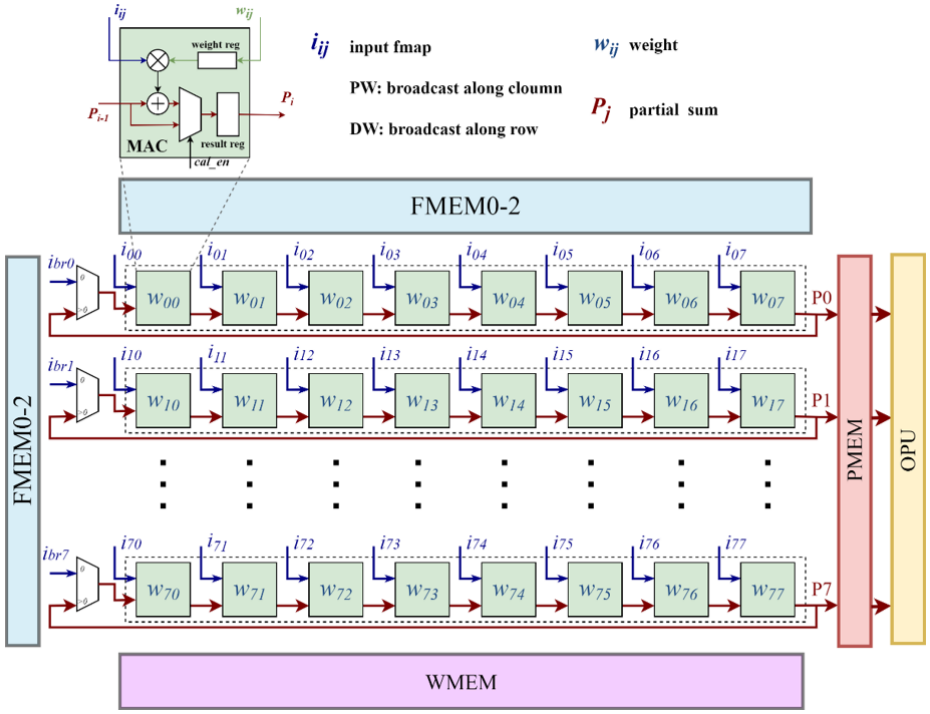

Figure 5. Schematic view of the MAC array.

In order to support the processing of both DW and PW Conv layers, we use the layers' spatially unrolled convolutional loop nest, which utilizes the parallel characteristic of convolutional calculation. While processing DW Conv layers, the MAC units in different rows store the weight of different filters, and units in different positions of the same row store the weights inside the kernels. The unrolled convolutional loop nest of DW is described in Figure 6a. In each cycle, eight input features from the corresponding feature

map memory (FMEM) are broadcasted to the array along the direction of the row. As the calculating results propagate from left to right in the array, the output features (OF) that have convolved the input features (IF) inside the receptive field are generated in the rightmost column of MAC units and sent to the OPU for postprocessing. Furthermore, we realize the characteristic of zero padding of the input feature and convolution with a stride of two by way of controlling the calculation enable signal of the MAC unit.

```
1: for I = 1 → I_c/8 do
2:     for x = 1 → X do
3:         parfor i = 1 → 8 do
4:             parfor k = 1 → K do
5:                 P[8·I + i][k] =
6:                 P[8·I + i][k − 1]+
7:                 F[8·I + i][x] · W[8·I + i][k]
8:             end parfor
9:             ofmap[8·I + i][x] = P[8·I + i][k]
```

(**a**)

```
1: for O = 1 → O_c/8 do
2:     for I = 1 → I_c do
3:         for x = 1 → X do
4:             parfor o = 1 →8 do
5:                 parfor i = 1 → 8 do
6:                     P[8·O + o][8·I + i] =
7:                     P[8·O + o][8·I + i − 1]+
8:                     F[8·I + i][x] · W[8·O + o][8·I + i]
9:                 end parfor
10:                psum[8·O + o][x] = P[8·O + o][8]
```

(**b**)

Figure 6. (**a**) Spatially unrolled convolutional loop nest of DW layers; (**b**) spatially unrolled convolutional loop nest of PW layers.

As for the PW Conv layers, the unrolled convolutional loop nest is shown in Figure 6b. MAC units in different rows store the weights of different filters, and units in different positions of the same row store the weights of the corresponding input channels of the filters. In each cycle, eight input features from the adjacent input channels are fetched from the FMEM and broadcasted to the units along the direction of the column. As the calculating results propagate along the direction of the output channel, the partial sums that have accumulated the products of the input features and weights of eight adjacent input channels are generated in the right side of the array and stored in the PMEM for the iteration of the next batch of input channels. Until the calculations of all input channels are finished, the partial sums are sent to the OPU for postprocessing. Notably, to fit the propagating delay of the partial sums, the eight input features should enter the array with an incremental delay. We realize this function with the setup of a shift register with incremental depth.

The dataflow of the accelerator has two major superiorities: First, as a weight stationary dataflow, the same weights remain stationary in an MAC unit and are used in consecutive cycles until they have been fully calculated in the loop nest. This means that the number of accesses to the weight memory (WMEM) is minimized. Second, using the two strategies of weight mapping and input-feature broadcasting, our MAC array can support both DW and PW operations. With the mapping of the parallelizable operation, our accelerator can execute a continuous inference computation at the maximal throughput of 64 MACs per cycle.

3.3. Distributed-Memory Setup

In order to meet the bandwidth requirements and parallel access of the MAC array and OPU, a setup of distributed memories with different sizes is used.

3.3.1. WMEM and BMEM

These two memories store the variables of the entire network model, and once loaded, they keep unchanged during runtime. The WMEM is 20 kB in size, with a port width of 64 bit and a depth of 2560. Every time the MAC array finishes the calculation of a batch of output channels in DW mode or finishes the calculation of a batch of input channels in PW mode, it reads the weights from the WMEM and updates the weight register inside the MAC units. In each cycle, the weight registers of a column of MAC units are updated. The bias memory (BMEM) has a size of 1 kB and a depth of 128. When the OPU receives

the output features of a batch of channels, it accesses the BMEM and updates the bias register inside.

3.3.2. FMEM

Three FMEMs are used to store the input and output feature maps during inference and are switched dynamically. For instance, the initial feature map is stored in FMEM0, and after convolution layer0, the output feature map is stored into FMEM1. For layer1, the MAC array reads the input feature map from FMEM1, and the generated output feature map is stored back into FMEM0. However, as mentioned in Section 2.2, due to the existence of the residual path in the IBBs, an additional memory is needed. Thus, FMEM is introduced to store the feature map of the residual path. Note that the number of channels of the residual path is equal to the bottleneck layer. The depths of FMEM0/1/2 are 512, 256, and 64, respectively. While scheming the FMEM, the feature maps of the residual path are always stored in FMEM2. As summarized in Table 2, each memory either provides the feature map of the main path or the residual path (IF/BR), receives a feature map (OF/MOVE_IN), or does not work (IDLE).

Table 2. FMEM arrangement for our TENet model.

Layer	FMEM0	FMEM1	FMEM2
IBB0-0:PW (residual)	IDLE	IF	OF
IBB0-1:PW	OF	IF	IDLE
IBB0-2:DW	IF	OF	IDLE
IBB0-3:PW	OF	IF	BR [1]
IBB1-0:PW	IF	OF	MOVE_IN [2]
IBB1-1:DW	OF	IF	IDLE
IBB1-2:PW	IF	OF	BR

[1] "BR" indicates that feature maps stored in this memory enter the MAC array through a partial-sum port. [2] "MOVE_IN" indicates that feature maps that read from the memory that is in an IF state are stored in this memory.

3.3.3. PMEM

Even though it is possible for a partial sum to directly remain in an FMEM, it would cause a huge precision loss due to the truncating too early from 20 b to 8 b. Thus, we introduce the PMEM with a port width of 160 b and a depth of 64 to support the high-precision partial sum within the MAC array before postprocessing while processing PW layers.

3.4. Output Processing Unit

As displayed in Figure 7, the OPU combines bias adding, ReLU activation, and average pooling in a combinational circuit and exports the output through a group of registers. The address and enable signals used to access the FMEM are delayed by one cycle in the OPU. The output feature maps from the MAC array firstly add to the bias and then are truncated from 20 b to 8 b with the reference of the location of the point configured in the instruction. Whether to execute the ReLU activation and average pooling or not is determined by the corresponding enable signals. Notice that the dividing operation in average pooling is implemented with a right shift, so the width of the pooling window should be a power of 2.

3.5. Control Unit

For each layer in the TENet, a 64-bit instruction is designed to configure the accelerator. Layer type, stride, padding, boundaries of the convolutional loop nest, the scheming of the FMEM, the base address of WMEM access, and the postprocessing steps to be performed are defined in the instruction. The instruction is decoded in the control unit, and the generated control signals are sent to corresponding modules. The address and enable

signals for accessing memories are generated by an address-generation unit and connected to the ports of the memories through an interconnecting module.

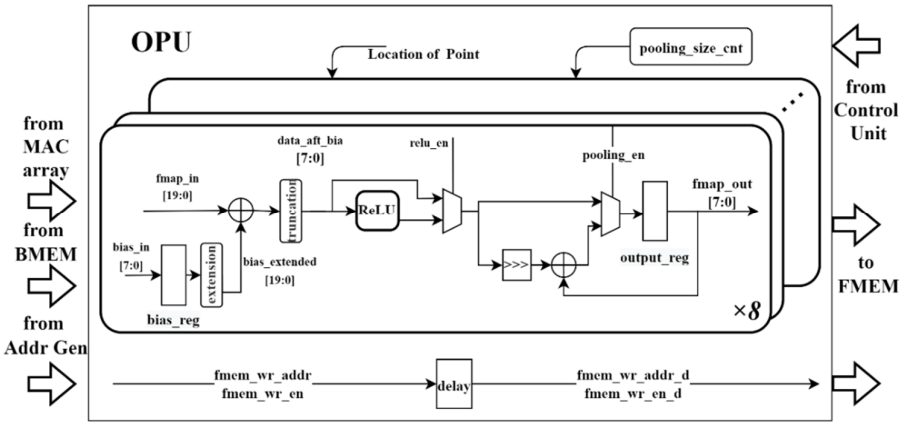

Figure 7. The structure of the output processing unit.

4. Experiment and Result

In this section, we first show the training and optimizing results of our modified TENet for KWS. Then, we introduce the implementation of the accelerator architecture and the comparison to related work.

4.1. Model Training and Optimization

We evaluate our modified model using Google Speech Commands Dataset [15]. Following Google's implementation, we seek to distinguish 12 classes: "yes", "no", "up", "down", "left", "right", "on", "off", "stop," "go", unknown, and silence. Using the SHA-1 hashed name of the audio files, we split the dataset into training, validation, and test sets in the ratio of 8:1:1. Following Google's processing procedure, we apply a random X second-time shift to the training data, where X is sampled from U(−0.1,0.1). After that, we randomly sample and crop the background noise provided in the dataset and multiply it with a random factor sampled from U(0,M). However, different from Google's method that fixes M as 0.1, we dynamically adjust M to a fixed signal–noise ratio (SNR) according to the energy of the audio and noise samples in order to enhance the trained model's noise immunity.

For feature extraction, compared with the traditional MFCC in [6,8,15], we use the simplified MFCC algorithm mentioned in Section 2.1. The raw audio is decomposed into a sequence of frames of 16 ms without overlap, and the number of the rectangular mel filters is 30. All the frames after the FFT and mel filtering are then added to the adjacent frames to form the input feature map of our model with a length of 61 frames per second.

We train and evaluate our model using TensorFlow and use the Adam optimizer, whose initial learning rate is 0.01 for 30 k iterations. The learning rate decays by 0.1 every 10 k iterations. The standard weight decay is set to 0.00004, and the batch size is set to 100.

We use accuracy as the main metric to evaluate the performance of the models. Additionally, the numbers of parameters and multiplications are also taken into consideration. We also plot the receiver operating characteristic (ROC) curves of the different models.

As shown in Table 3, maintaining a comparable accuracy to TC-ResNet8 [8], our model produces a 3.7× reduction in parameters and a 2.8× reduction in multiply operations. In the light of the global model that includes both feature extraction and feature classification, our model achieves similar performance, with a 2× reduction in multiply operations compared with TENet6-narrow [6]. This is a smaller footprint for easy deployment on hardware. We evaluated our model and baseline models with varying strengths of background noise. When working with a high level of noise, our model still achieves a competitive accuracy and shows a better noise resistance, which is suitable for application in complex scenarios.

Table 3. Comparison of our model and the baseline models.

Model	Param	Mult	Mult. (with FE)	Acc (%) with 95% CI [1]	Acc (%)/SNR (dB) [2]		
					10	0	−5
TC-ResNet8 [8]	66 K	1.12 M	1.47 M	96.14 [94.00,98.27]	87.35	66.16	46.50
TENet6-narrow [6]	17 K	553 K	913 K	96.07 [93.86,98.28]	92.77	83.27	70.15
This work	18 K	398 K	461 K	95.98 [93.78,98.18]	92.35	84.49	72.22

[1] "CI" means the confidence interval of the accuracy of the model. A short CI means a small margin of error and that we have a relatively precise estimate. [2] While evaluating the model with background noise, we randomly select and crop the noise slices from the dataset. The accuracy shown in this column is the average value of multiple evaluations.

As presented in Figure 8, our modified TENet model has a lower false-reject rate under the same false-alarm rate of the baseline models. The small AUC means that the model misses fewer target keywords on average for different rates of errors in detecting keywords.

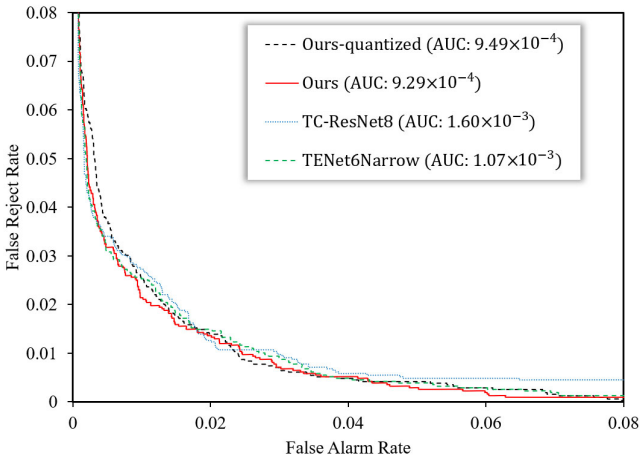

Figure 8. Receiver operating characteristic (ROC) curves for different models with corresponding values of area under the curve (AUC). The x-axis of the curve is the false-alarm rate, and the y-axis is the false-reject rate.

Furthermore, we quantized the network to an integer 8-bit model at the cost of a 0.62% drop in accuracy. The quantitated model only requires a quarter of the original space for parameter storage. In general, the hardware-friendly modified TENet in this paper is proved to be efficient and to have a small footprint.

4.2. Hardware Implementation

We use Vivado (2020.1) to synthesize and implement the accelerator and deploy it on a Xilinx ZYNQ 7Z020 field programmable gate array (FPGA) board. The proposed architecture achieves a peak performance of 5.36 GOPS at the clock frequency of 50 MHz with a power consumption of 209 mW.

The comparison of resources utilization and performance with related works is shown in Table 4. Following a hardware/model co-design approach, we reach a balance between performance and resource utilization. Compared with the FPGA accelerators, which are designed for DSCNN and show a performance in the same order of magnitude, our work uses the least FPGA resources. Our model's energy efficiency is worse than that of [11]; this may be due to our accelerator being configurable and being able to support the network of residual blocks.

Table 4. Comparison with related FPGA accelerator.

Metrics		[9]	[10]	[11]	This Work
Resources Utilization	**LUT**	39,634	9197	6797	5575
	FF	19,384	16,942	6694	2631
	DSP	209	109	97	64
	BRAM	402.75 kB	110.5	13.5	8
Platform		artix7 xc7a200T	zynq 7z045	zynq 7z020	zynq 7z020
Frequency (MHz)		80	100	50	50
Performance (GOPS)		12.16	3.14	9.6	5.36
Power consumption (W)		/	2.15	0.106	0.209
Energy efficiency (GOPS/W)		/	1.46	90.56	25.6

Keyword-spotting systems usually act as always-on switches in systems, so we are more concerned with power consumption rather than peak performance and throughput. It shows that we expect the circuit to work at the lowest clock frequency and occupy the least area possible to reduce power consumption.

Through the measurement on the board, our accelerator needs a runtime of 7266 cycles to process the features of 1 s length audio, which is reduced by 3.1× compared with related work [7]. Under a latency of 100 ms for real-time classification, our accelerator can work at the minimum clock frequency of 90 kHz. As shown in Table 5, the on-chip memory size of our accelerator is 2.6× smaller than that of Ultratrail [7], and our work achieves a higher accuracy after quantization. Limited by the basic circuit structure of the FPGA platform, the power of our accelerator is larger. Yet, the proposed architecture has the potential of reaching lower power consumption while using the same process technology.

Table 5. Comparison with related work.

Metrics	Ultratrail [7]	This Work
Network structure	TC-ResNet8	Modified-TENet6
On-chip MEM size (kB)	74.125	28.75
Bit width (weights/Fmaps)	6/8	8/8
Accuracy quantized (%)	93.09	95.36
Keywords	10	10
Runtime (cycles)	22481	7266
MFRTI [1] (kHz)	250	90
Implementation	GF22 nm	Xilinx ZYNQ 7z020
Power	8.2 µW	209 mW

[1] "MFRTI" indicates the minimum clock frequency for real-time inference, where lower MFRTI means lower dynamic power of the accelerator for real-time inferencing.

5. Conclusions

In this paper, we present a hardware-friendly small-footprint temporal efficient neural network for KWS. The proposed model reaches a trade-off between accuracy and model size via the modification and optimization of the feature-extraction algorithm and network structure. In order to efficiently deploy the TENet on hardware, we propose a dataflow for parallelly mapping the temporal depthwise separable network and design a configurable accelerator with a distributed-memory setup. We implement the accelerator architecture on FPGA and achieve a KWS accuracy of 95.36%.

Admittedly, one of the limitations of this work is that we only design the hardware architecture of the neural-network part of the model, while feature extraction is performed by software. Another concern about this paper is that the accelerator is implemented on an

FPGA platform, which limits the further optimization and special design of lower area and energy cost.

In future work, we plan to implement the accelerator using ASIC process technology, in combination with a specific hardware module for the MFCC. Additionally, based on the configurability of the existing accelerator, we envision an accelerator for a wider range of scenarios using temporal convolutional networks.

Author Contributions: Data curation, K.H.; Funding acquisition, D.C. and T.S.; Methodology, K.H., D.C. and T.S.; Project administration, K.H., D.C. and T.S.; Software, K.H.; Supervision, D.C. and T.S.; Writing—original draft, K.H.; Writing—review and editing, D.C. and T.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Science and Technology Program of Guangdong Province, grant number 2021B1101270007.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to privacy.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Chen, G.; Parada, C.; Heigold, G. Small-Footprint Keyword Spotting Using Deep Neural Networks. In Proceedings of the ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing—Proceedings, Florence, Italy, 4–9 May 2014; pp. 4087–4091. [CrossRef]
2. López-Espejo, I.; Tan, Z.H.; Hansen, J.; Jensen, J. *Deep Spoken Keyword Spotting: An Overview*; IEEE: Manhattan, NY, USA, 2021. [CrossRef]
3. Giraldo, J.S.P.; Lauwereins, S.; Badami, K.; Verhelst, M. Vocell: A 65-Nm Speech-Triggered Wake-Up SoC for 10-MW Keyword Spotting and Speaker Verification. *IEEE J. Solid-State Circuits* **2020**, *55*, 868–878. [CrossRef]
4. Liu, B.; Cai, H.; Wang, Z.; Sun, Y.; Shen, Z.; Zhu, W.; Li, Y.; Gong, Y.; Ge, W.; Yang, J.; et al. A 22nm, 10.8μW/15.1μW Dual Computing Modes High Power-Performance-Area Efficiency Domained Background Noise Aware Keyword-Spotting Processor. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2020**, *67*, 4733–4746. [CrossRef]
5. Shan, W.; Yang, M.; Xu, J.; Lu, Y.; Zhang, S.; Wang, T.; Yang, J.; Shi, L.; Seok, M. 14.1 A 510nW 0.41V Low-Memory Low-Computation Keyword-Spotting Chip Using Serial FFT-Based MFCC and Binarized Depthwise Separable Convolutional Neural Network in 28nm CMOS. In Proceedings of the 2020 IEEE International Solid- State Circuits Conference—(ISSCC), San Francisco, CA, USA, 16–20 February 2020; pp. 230–232. [CrossRef]
6. Choi, S.; Seo, S.; Shin, B.; Byun, H.; Kersner, M.; Kim, B.; Kim, D.; Ha, S. Temporal Convolution for Real-Time Keyword Spotting on Mobile Devices. *Proc. Interspeech* **2019**, 3372–3376. [CrossRef]
7. Bernardo, P.P.; Gerum, C.; Frischknecht, A.; Lübeck, K.; Bringmann, O. UltraTrail: A Configurable Ultralow-Power TC-ResNet AI Accelerator for Efficient Keyword Spotting. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2020**, *39*, 4240–4251. [CrossRef]
8. Li, X.; Wei, X.; Qin, X. Small-Footprint Keyword Spotting with Multi-Scale Temporal Convolution. *arXiv* **2020**, arXiv:2010.09960. [CrossRef]
9. Boonyuu, G.; Wisayataksin, S. Configurable Hardware Architecture of Multidimensional Convolution Coprocessor. In Proceedings of the 2021 Second International Symposium on Instrumentation, Control, Artificial Intelligence, and Robotics (ICA-SYMP), Bangkok, Thailand, 1–25 January 2021. [CrossRef]
10. Liao, J.; Cai, L.; Xu, Y.; He, M. Design of Accelerator for MobileNet Convolutional Neural Network Based on FPGA. In Proceedings of the 2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), Chengdu, China, 20–22 December 2019; Volume 1, pp. 1392–1396. [CrossRef]
11. Jiang, Y.; Ren, J.; Xie, X.; Zhang, C. Hardware Implementation of Depthwise Separable Convolution Neural Network. In Proceedings of the 2020 IEEE 15th International Conference on Solid-State & Integrated Circuit Technology (ICSICT), Kunming, China, 3–6 November 2020; pp. 1–3. [CrossRef]
12. Giraldo, J.; Verhelst, M. Hardware Acceleration for Embedded Keyword Spotting: Tutorial and Survey. *ACM Trans. Embed. Comput. Syst* **2021**, *20*, 6. [CrossRef]
13. Han, W.; Chan, C.F.; Choy, C.S.; Pun, K.P. An Efficient MFCC Extraction Method in Speech Recognition. In Proceedings of the 2006 IEEE International Symposium on Circuits and Systems, Kos, Greece, 21–24 May 2006. [CrossRef]
14. Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Adam, H.; Kalenichenko, D. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 2704–2713. [CrossRef]
15. Warden, P. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *arXiv* **2018**, arXiv:1804.03209.