A Low-Power Sparse Convolutional Neural Network Accelerator With Pre-Encoding Radix-4 Booth Multiplier

Quan Cheng, Liuyao Dai, Mingqiang Huang, Ao Shen, Wei Mao, *Member, IEEE*, Masanori Hashimoto, *Senior Member, IEEE*, and Hao Yu, *Senior Member, IEEE*

***Abstract*—Working on edging device, convolutional neural network (CNN) inference application demands low-power consumption and high-performance computation. Therefore, exploiting energy-efficient multiply-and-accumulate (MAC) unit and high-throughput sparse CNN accelerator is of great importance. In this brief, we develop a sparse CNN accelerator achieving a high MAC-unit utilization ratio and great power efficiency. The accelerator includes a radix-4 Booth multiplier for pre-encoding weights to reduce the number of partial products (PPs) and the encoder power consumption. The proposed accelerator has the following three features. Firstly, we reduce the bit number of PPs exploiting the features of radix-4 Booth algorithm and offline weight pre-processing. Secondly, we extract eight encoders from relevant multipliers and merge them into one pre-encoding module to reduce area. Finally, after encoding non-zero weights offline, we design an activation selector module to select the activations corresponding to non-zero weights for subsequent multiple-add operations. The proposed work is designed by Verilog HDL language and implemented in a 28nm process. The proposed accelerator achieves 7.0325 TOPS/W with 50% sparsity and scales with sparsity up to 14.3720 TOPS/W at 87.5%.**

***Index Terms*—Accelerator, CNN, MAC, radix-4 Booth, low-power, sparsity.**

I. Introduction

NOWADAYS, convolutional neural network inference on edge devices is widely studied since CNN models are becoming more complex with larger parameters. Currently, state-of-the-art CNNs [1], [2] often contain up to hundreds of megabytes of weight parameters and 600k operations per input pixels. Such large networks require huge computing power in hardware. Constrained by the limited hardware resources of edge devices, CNN inference accelerators need high energy efficiency and area efficiency in general matrix multiplication (GEMM) which accounts for more than 90% of CNN calculations. Namely, more than 90% CNNs computations are occupied by MAC operations [3].

Data sparsity has drawn great attention since it can be exploited in CNN inference accelerators [4] as zero elements reduce the computation and storage cost significantly. Meanwhile, structural sparsity, which is based on the traditional sparse algorithm with modified penalty terms, aims to organize more regular matrices. Besides, to run these neural networks (NNs), multipliers are necessary in hardware platforms to compute the model data. The radix-4 Booth algorithm can improve the performance of multiplication because it reduces the number of partial product (PP) rows by half [5].

This brief introduces a novel sparse CNN accelerator with custom radix-4 Booth multiplier with high data-reuse performance. The main contributions are summarized as follows:

- **Pre-encoding radix-4 Booth multiplier**: A pre-encoding radix-4 Booth algorithm, highly compatible with data reuse and weight offline pre-processing, is proposed. Reducing the bit number of PPs and merging eight encoders into one pre-encoding module based on the features of our accelerator, could simplify multiplication in CNN calculations, which reduces area and power consumption.
- **Sparse CNN accelerator**: For low-power design, the weights above 50% sparsity are steered into arithmetic units, such that each block supporting the computation of sparse data can reduce the number of MACs by half and realize the same amount of computation as regular accelerator without sparsity mechanism.

Manuscript received 4 November 2022; revised 29 November 2022; accepted 20 December 2022. Date of publication 22 December 2022; date of current version 8 June 2023. This work was supported in part by the National Key Research and Development Program of the Ministry of Science and Technology under Grant 2021YFE0204000; in part by the National Natural Science Foundation of China (NSFC, Key Program) under Grant 62034007; and in part by the Shenzhen Science and Technology Program under Grant KQTD20200820113051096. This brief was recommended by Associate Editor X. Li. *(Corresponding author: Hao Yu.)*

Quan Cheng and Masanori Hashimoto are with the Department of Communications and Computer Engineering, Kyoto University, Kyoto 606-8501, Japan.

Liuyao Dai, Mingqiang Huang, Ao Shen, Wei Mao, and Hao Yu are with the School of Microelectronics, Southern University of Science and Technology, Shenzhen 518055, China (e-mail: yuh3@sustech.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSII.2022.3231361.

Digital Object Identifier 10.1109/TCSII.2022.3231361

II. Proposed Work

Structural sparsity could effectively compress the computation amounts in NNs and reduce the power consumption thanks to the regular predictable weight property of structural sparsity with low index overhead. Fig. 1 shows the architecture of the proposed sparse accelerator consisting of 16 engines. Each engine including 8 blocks can steer the same weights, corresponding control signals and different activations into these 8 blocks. Each block including 4 MACs can select activations corresponding to non-zero weights and realize the

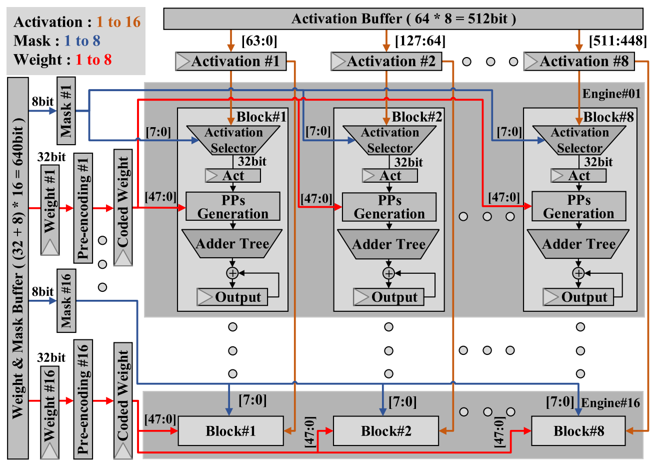


Fig. 1. The architecture of proposed sparse CNN accelerator.

computation operations in channel dimension. The 4 MACs integrated into the PPs generation and adder tree modules can realize the operation of the sum of 4 multiplications. Different blocks in the same engine can generate different pixels in the same output channel, and different engines can generate pixels in different output channels. As shown by the red and yellow lines in Fig. 1, the parallel processing is implemented such that each weight is shared to drive 8 blocks to generate 8 pixels in the same output channel and each activation is shared to drive 16 blocks having the same number in 16 different engines to generate pixels in different output channels simultaneously.

Our design introduces two novel methods and refers to two existing strategies to achieve low-power computation. Firstly, we separate the encoders from the multipliers and combine them into a single pre-encoding module to drive 8 blocks, reducing 7× area of the encoders. Secondly, we propose a method to reduce the bit number of resultant PPs exploiting the feature of radix-4 Booth algorithm and offline weights pre-processing. Then, we propose a spatially-unrolled sparse architecture similar to [6] and improve it to support a full range of sparsity from 0 to 100%. Finally, we apply a low-power adder tree design [7] to the PPs accumulation and design some special standard cells to further reduce the power consumption.

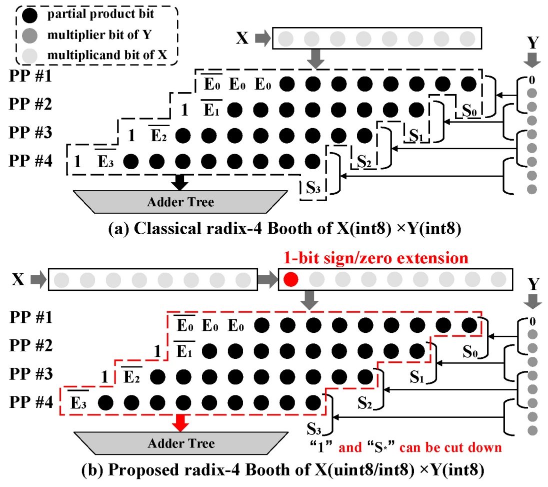
A. Pre-Encoding Radix-4 Booth Multiplier

The radix-4 Booth algorithm can facilitate the multiplication and improve the timing. A pre-encoded mechanism [8] of radix-4 Booth algorithm is proposed for low-power multiplier-level design via detecting and gating the "±0X" case. For a system-level low-power design, we propose a pre-encoding structure exploiting the feature of radix-4 Booth multiplier, the data-reuse structure of our accelerator, and the pre-processing of weights. To support both *unsigned×signed* and *signed×signed* in convolutional computation, we provide a solution that extends 1-bit sign/zero in the highest bit of multiplicand. Taking *X(uint8/int8)×Y(int8)* as an example, the PPs generation based on radix-4 Booth algorithm is given by:

$$\sum_{i=0}^{4-1}(-1)^{sign(M_i)}2^{2i}|M_i|concat\{sign(X)\&is\_sign, X\},$$

TABLE I
RADIX-4 BOOTH ALGORITHM

$y_{2i+1}\ y_{2i}\ y_{2i-1}$	M_i	Operation	neg_i	two_i	one_i
0 0 0	0	$+0X$	0	0	0
0 0 1	+1	$+1X$	0	0	1
0 1 0	+1	$+1X$	0	0	1
0 1 1	+2	$+2X$	0	1	0
1 0 0	-2	$-2X$	1	1	0
1 0 1	-1	$-1X$	1	0	1
1 1 0	-1	$-1X$	1	0	1
1 1 1	-0	$-0X$	1	0	0


partial product bit
multiplier bit of Y
multiplicand bit of X
PP #1
PP #2
PP #3
PP #4
Adder Tree
(a) Classical radix-4 Booth of X(int8) ×Y(int8)
1-bit sign/zero extension
"1" and "S" can be cut down
(b) Proposed radix-4 Booth of X(uint8/int8) ×Y(int8)
Fig. 2. Partial products of 8*bit* × 8*bit* radix-4 Booth multiplier.

where *concat*{} is bit-concat operation, *sign*() returns the sign bit of data, the coefficient M_i is shown in Table I, and *is_sign* is 1 if X is a signed type, 0 otherwise.

On the basis of above formula and sign extension structure [9], we can get the PPs of $X(uint8/int8) \times Y(int8)$ as shown in Fig. 2b, where E_i is the sign bit of each PP, "1" is a normal constant, and S_i is the sign of M_i. Since the formula of the convolutional layer is $\boldsymbol{w} \times \boldsymbol{x} + \boldsymbol{b}$, the weights are known in advance before being loaded into accelerator, and the S_i bits are only related to the weights, we can add "1" and S_i to bias parameter $\boldsymbol{b}$ beforehand, which means we do not need to construct any circuits related to "1" and S_i compared with classical radix-4 Booth as shown in Fig. 2a. Namely, an offset value added to bias $\boldsymbol{b}$ offline for adjustment per multiplier can be expressed as follow:

$$offset = 2^{12} + 2^{14} + \sum_{i=0}^{3} 2^{2i}S_i$$

As a result, the parts only surrounded by the red dotted line in Fig. 2b are summed up in the accelerator, reducing the bit number of PPs. The bit numbers of each PP (#1 to #4) given to the adder tree are 12, 10, 10, and 10, respectively. Namely, the bit number that could be reduced in each multiplier is 6.

Morever, according to the data-reuse feature of our accelerator, we propose to extract the encoder from radix-4 Booth multiplier as a sharable pre-encoding module to further reduce the power consumption. Fig. 3 shows the sharable feature of pre-encoding module. Since the input/output of encoder is only related to weights and each weight can be reused to drive eight blocks in each engine, the pre-encoding module can be shared

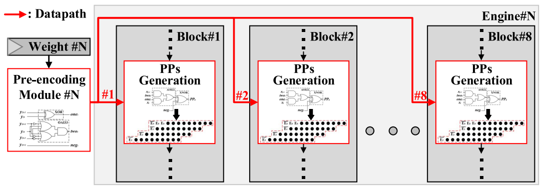


Fig. 3. Sharable feature of pre-encoding module.

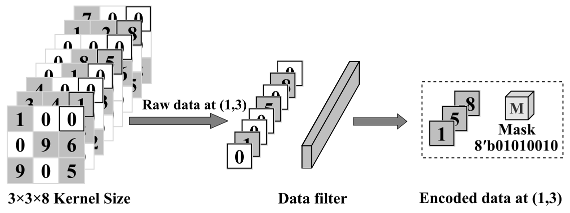


Fig. 4. Weight offline encoding in the channel dimension.

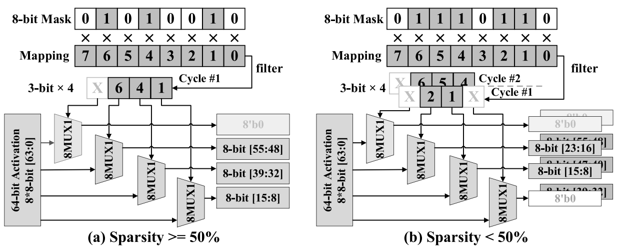


Fig. 5. Activation selector at different sparsity.

by the eight blocks in each engine. Namely, we extract eight encoders from eight blocks in each engine and merge them into just one encoder to reduce area and power consumption.

B. Activation Selector

Basically, CNNs contain four kinds of structural sparsity from irregular level to regular level (i.e., fine-grained, vector-level, kernel-level, filter-level). Regular sparsity makes the design of hardware accelerator easier. To exploit the structural sparsity at different levels, a flexible accelerator is necessary.

A sparse PE [12] accepts compressed sparse column (CSC) format to skip zero elements. However, the control logic of PE is too complex. A time-unrolled systolic structure [6] achieves 100% utilization of MACs. However, this structure increases not only register accesses for data accumulation but also the number of registers in the data flow. On the other hand, since the weights are known in advance, we could pre-process the raw weights. Based on [6], Fig. 4 gives an example of using a kernel size $= 3 \times 3 \times 8$. The weights are coded by removing the zero elements and appending the bitmask M indicating the location of non-zero elements. M signal can be shared by the eight blocks as depicted in Fig. 1.

Besides, a spatially-unrolled structure [6] supporting variable sparsity density bound block (VDBB) is presented. However, it does not support the sparsity under 50%. We then propose an improved activation selector to support a full range of sparsity. As shown in Fig. 5, this selector can complete the filtering of activations using 8-bit mask data. We adopt 1-bit control signal in each engine to set whether the data could be computed within one or two clocks. When the sparsity is greater than or equal to 50%, the selector can finish selecting eight data within one clock cycle (Fig. 5a), otherwise in two consecutive clock cycles (Fig. 5b). Firstly, we map 8-bit mask into 3-bit×4 data as selection signals for 8:1 multiplexers (8MUX1s). Secondly, we integrate four 8MUX1s into our selector selecting at most four 8-bit data from eight activations using four 3-bit selection signals. Finally, the selected data is output. In addition, when some engines execute one-cycle computations with sparsity above 50% and the others execute two-cycle computations with sparsity under 50%, one-cycle computation engines are gated for one extra cycle.

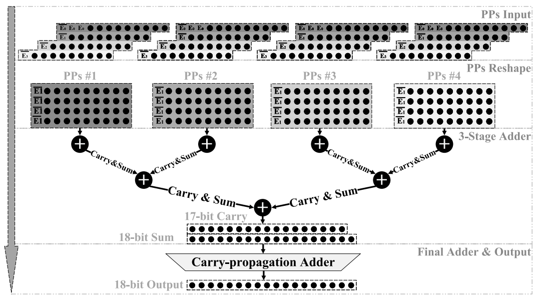


Fig. 6. Improved wallace tree based on customized standard cells.

TABLE II
TRUTH TABLE OF FAN AND HAN

A	B	Cin	FAN_CoN	FAN_SN	HAN_CoN	HAN_SN
0	0	0	1	1	1	1
0	0	1	1	0	1	1
0	1	0	1	0	1	0
0	1	1	0	1	1	0
1	0	0	1	0	1	0
1	0	1	0	1	1	0
1	1	0	0	1	0	1
1	1	1	0	0	0	1

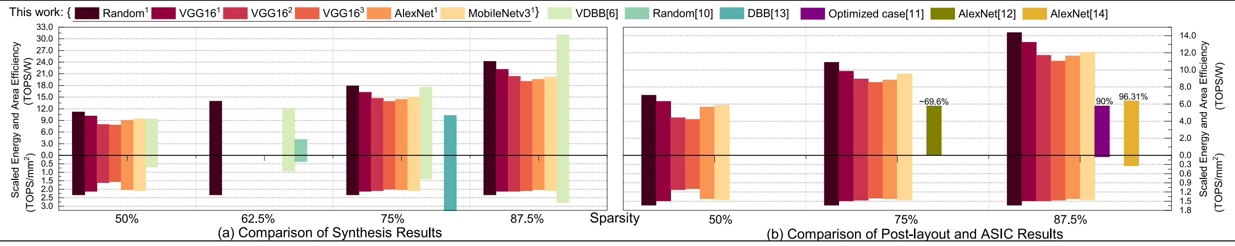
C. Adder Tree

An optimized wallace tree [7] is proposed to implement the accumulation operation. Carry save addition (CSA) is applied to avoid the carry propagation delay. We refer to this strategy to build the adder tree structure in our design depicted in Fig. 6. To reduce the bit extension during the accumulation phase, we first reshape the PPs from #1 to #4, then add the same level PPs up at the first stage. Then, to optimize the timing of adder tree, we only generate the carry and sum outputs at each stage using CSA, avoiding carry propagation. Finally, we use a carry-propagation adder (CPA) to calculate the final result once we get the final two numbers.

To further reduce the power and area consumption, we develop special standard cells to form an adder tree. Full adder (FA) and half adder (HA) are the most frequently-used standard cells in our adder tree. Taking FA with CoN and SN (FAN) and HA with CoN and SN (HAN) as an example, the truth table is shown in Table II, and the output of FAN and HAN is the invert of carry (CoN) and the invert of sum (SN), which could reduce the number of transistors in our circuit for invert operation. We adopt these cells (e.g., FA, FAN, HA, HAN) to construct the adder tree collaboratively.

TABLE III
EVALUATED ACCELERATOR

Accelerator	Technology (nm)	Freq.	Chip memory	Throughput[†] (TOPS)	Energy Efficiency[t] (TOPS/W)	Area Efficiency[t] (TOPS/mm²)	Weight Sparsity
This work	28	550 MHz	128KB	0.5632	$^s11.1846/^p7.0325$	$^s2.3278/^p1.6304$	50.0%Random[1]
				0.5632	$^s17.8278/^p10.8980$	$^s2.3278/^p1.6304$	75.0%Random[1]
				0.5632	$^s24.1410/^p14.3720$	$^s2.3278/^p1.6304$	87.5%Random[1]
Liu et al. [6]	16	1 GHz	2.5MB	4	$^s16.8/^s_e9.3644$	$^s2.13/^s_a0.6955$	50.0%VDBB
	16	1 GHz	2.5MB	4	$^s31.3/^s_e17.4468$	$^s4.29/^s_a1.4008$	75.0%VDBB
	16	1 GHz	2.5MB	4	$^s55.7/^s_e31.0475$	$^s8.52/^s_a2.7820$	87.5%VDBB
SMT-SA [10]	16	1 GHz	2.5MB	4	$^s7.4/^s_e4.1248$	$^s1.13/^s_a0.3690$	62.5%Random
Matsubara et al. [11]	12	800 MHz	~41.6MB	66	$^c13.8/^c_e5.7692$	$^c0.2621/^c_a0.0481$	Optimized case (90% sparsity)
EyerissV2 [12]	65	200 MHz	246KB	0.1536	$^p0.9629/^p_e5.9813$	$^p0.5053/2695$K gates	sparse AlexNet (~69.6% sparsity)
Kang et al. [13]	65	1 GHz	58KB	0.5	$^s1.65/^s_e10.2495$	$^s1.01/^s_a5.4429$	75%DBB
Sticker [14]	65	200 MHz	170KB	0.1024	$^c2.82/^c_e6.3857$	$^c0.0630/^c_a0.3395$	Pruned AlexNet (96.31% sparsity)

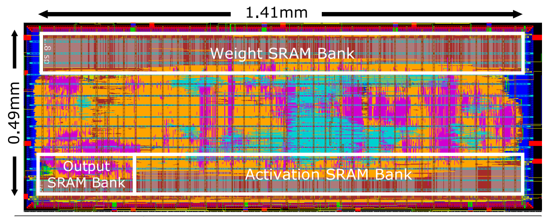


^{s}Synthesis result; ^{p}Post-layout result; ^{c}ASIC result; †Peak hardware throughput; ^{t}Calculated with sparsity and network structure

$_e$Energy efficiency scaling = Energy efficiency$\times \frac{process}{28nm} \times (\frac{Voltage}{0.81V})^2$; $_a$Area efficiency scaling = Area efficiency $\times(\frac{process}{28nm})^2$

^{1}Channel-level sparsity; ^{2}VDBB sparsity; ^{3}Random sparsity

*Note that all synthesis and post-layout results of our accelerator are obtained based on 0.81V/125C SSG corner and RC-worst case.



Technology	SMIC 28nm
Supply Voltage	pre-driver: 0.81V-0.99V; post-driver: 1.62V-1.98V
Core Area	1.41mm×0.49mm
SRAM	128KB
Frequency	550MHz
Power	160.17 mW at 50% random sparsity with SSG corner
Precision	INT8/UINT8

Fig. 7. Layout result of the proposed accelerator.

TABLE IV
PERFORMANCE AT 50% RANDOM SPARSITY

Component	Power, mW	Area, mm²(%)
Pre-encoding Module	$^s0.762/^p1.112$	$^s1.860$e-4 (0.038%)
Activation Selector	$^s12.768/^p25.796$	$^s1.784$e-2 (3.686%)
PPs Generation	$^s14.592/^p24.265$	$^s2.368$e-2 (4.893%)
Adder Tree	$^s28.672/^p56.100$	$^s4.506$e-2 (9.311%)
Reg/Buffer/Controller/Oscillator	$^s33.906/^p41.715$	$^s3.318$e-2 (6.856%)
SRAM (128KB)	$^s10.010/^p11.182$	$^s0.3640$ (75.215%)
Total	$^s100.71/^p160.17$	$^s0.4839$ (100%)

^{s}Synthesis result; ^{p}Post-layout result

III. EXPERIMENTAL RESULTS

A. Methodology

The proposed accelerator is designed by Verilog HDL language and synthesized for SMIC 28nm using Synopsys Design Compiler. The clock frequency is set to be 550MHz with 20% uncertainty, and the input/output latency is set to be 65% clock period. Besides, we develop the above-mentioned standard cells with specified functions and generate relevant library. Furthermore, the layout design is implemented using Cadence Innovus with two corners: 1) 0.81V/125C SSG corner with RC-worst case, and 2) 0.99V/0C FF corner with RC-best case. The power, performance and area (PPA) values are obtained using Design Compiler, PrimeTime and Innovus.

The chip layout and its implementation results are shown in Fig. 7. The core area is 1.41mm×0.49mm with 128KB SRAM including 62.5KB weight SRAM, 50KB activation SRAM, and 15.5KB output SRAM. The typical pre-driver voltage is 0.81V, and the power consumption is 160.14mW with 50% sparsity at 550MHz based on the SSG corner and RC-worst case.

B. Performance Analysis

With 50% random weight sparsity, the PPA metrics of each part are provided in Table IV. Considering the computing amounts and convolutional layer size of NNs (e.g., VGG16, AlexNet), setting the size of SRAM to 128KB is appropriate and could reduce memory access. To evaluate the performance of our multiplier and accelerator, we construct three cases for comparison. Case 1, whose block realizes the sum of eight multiplications, adopts traditional 8 × 8 multiplier generating eight PPs without pre-encoding module and sparsity mechanism. Case 2, whose block realizes the sum of eight multiplications without sparsity mechanism, adopts pre-encoding radix-4 Booth multiplier. Case 3, whose block realizes the sum of four multiplications with sparsity mechanism, adopts radix-4 Booth multiplier without sharing

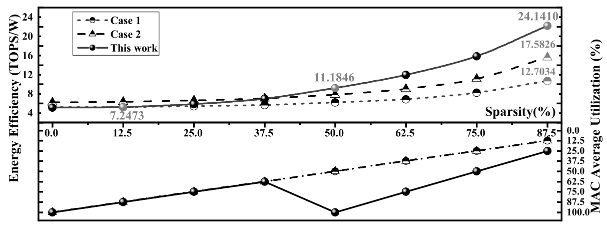


Fig. 8. Energy efficiency and MAC utilization at different channel-level sparsities.

TABLE V
PERFORMANCE ANALYSIS IN THIS BRIEF

Case	Area (mm^2)	Energy Efficiency (TOPS/W) at different sparsity		
		0%	50%	87.5%
Case 1	0.5975	7.1429	8.2090	12.7034
Case 2	0.5655	8.2090	9.8214	17.5826
Case 3	0.4864	7.0777	11.1354	24.0348
This work	0.4839	7.1088	11.1846	24.1410

*All the data is synthesis results in this table.

the pre-encoding module. The area and energy efficiency metrics are shown in Table V. The synthesis results of energy efficiency and MAC average utilization from 0% to 87.5% weight sparsity is shown in Fig. 8. Our accelerator achieves a 90.03% improvement in energy efficiency compared with case 1 and the highest energy efficiency (24.1410 TOPS/W) at 87.5% sparsity.

C. Comparison With Prior Work

Table III lists the metrics of the state-of-the-art sparse CNN accelerators for evaluation. To compare with these INT8 accelerators, we scale the area and energy efficiency of these works into our design as references following the approach of [15], [16]. From 50% to 75% weight sparsity, our design offers higher area and energy efficiency than others. Moreover, by applying the quantization [17] and group sparsity [18] strategies to channel-level CNN training, we get some benchmarks (i.e., VGG16, AlexNet, MobileNetV3) with different sparsity. Also, we train these CNNs based on VDBB and random format for comparison. Besides, the bit width of quantized weights and activations is 8. Since our accelerator cannot achieve high MAC utilization in small-size convolutional layers and fully connected layers, the energy efficiency is relatively lower than that of channel-level random data case.

IV. CONCLUSION AND DISCUSSION

In summary, this brief presents a low-power sparse CNN accelerator with pre-encoding radix-4 Booth multiplier. Based on the feature of radix-4 Booth algorithm, we lower the number and bit width of PPs. To reduce memory access, we reuse activations and weights based on the architecture of our accelerator, allowing us to share the encoders in multipliers. Furthermore, we filter the weights offline to generate the mask signals to realize sparsity mechanism. Implemented using 28nm technology, the proposed accelerator achieves 7.0325 TOPS/W at 50% sparsity and 14.3720 TOPS/W at 87.5%.

Future work will seek to further improve the energy efficiency of the proposed accelerator by combining the approximate radix Booth multiplier [19], [20] with our current design since the approximate multiplier could further decrease the complexity of logic circuits. However, the approximate multiplier may have a negative impact on the accuracy of NNs. We need to explore the efficiency and accuracy trade-off.

REFERENCES

[1] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi, "Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 11, pp. 5784–5789, Nov. 2018.
[2] B. Wu et al., "FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 10726–10734.
[3] B. Moons and M. Verhelst, "An energy-efficient precision-scalable ConvNet processor in 40-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 52, no. 4, pp. 903–914, Apr. 2017.
[4] S. Han et al., "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, 2016, pp. 243–254.
[5] X. Cui, W. Liu, X. Chen, E. E. Swartzlander, and F. Lombardi, "A modified partial product generator for redundant binary multipliers," *IEEE Trans. Comput.*, vol. 65, no. 4, pp. 1165–1171, Apr. 2016.
[6] Z.-G. Liu, P. N. Whatmough, and M. Mattina, "Sparse systolic tensor array for efficient CNN hardware acceleration," 2020, *arXiv:2009.02381*.
[7] F. U. D. Farrukh et al., "Power efficient tiny YOLO CNN using reduced hardware resources based on booth multiplier and WALLACE tree adders," *IEEE Open J. Circuits Syst.*, vol. 1, pp. 76–87, 2020.
[8] Y.-J. Chang, Y.-C. Cheng, S.-C. Liao, and C.-H. Hsiao, "A low power radix-4 booth multiplier with pre-encoded mechanism," *IEEE Access*, vol. 8, pp. 114842–114853, 2020.
[9] M. D. Ercegovac and T. Lang, *Digital Arithmetic*, 1st ed. Los Altos, CA, USA: Morgan Kaufmann, 2003.
[10] G. Shomron, T. Horowitz, and U. Weiser, "SMT-SA: Simultaneous multithreading in systolic arrays," *IEEE Comput. Archit. Lett.*, vol. 18, no. 2, pp. 99–102, Jul.–Dec. 2019.
[11] K. Matsubara et al., "4.2 A 12nm autonomous-driving processor with 60.4TOPS, 13.8TOPS/W CNN executed by task-separated ASIL D control," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2021, pp. 56–58.
[12] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 292–308, Jun. 2019.
[13] H.-J. Kang, "Accelerator-aware pruning for convolutional neural networks," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 7, pp. 2093–2103, Jul. 2020.
[14] Z. Yuan et al., "STICKER: An energy-efficient multi-sparsity compatible accelerator for convolutional neural networks in 65-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 55, no. 2, pp. 465–477, Feb. 2020.
[15] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. Boston, MA, USA: Addison-Wesley, 2010.
[16] W. Liu, J. Lin, and Z. Wang, "A precision-scalable energy-efficient convolutional neural network accelerator," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 10, pp. 3484–3497, Oct. 2020.
[17] B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer, "Mixed precision quantization of ConvNets via differentiable neural architecture search," 2018, *arXiv:1812.00090*.
[18] Y. Li, S. Gu, C. Mayer, L. Van Gool, and R. Timofte, "Group sparsity: The hinge between filter pruning and decomposition for network compression," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 8015–8024.
[19] V. Leon, G. Zervakis, D. Soudris, and K. Pekmestzi, "Approximate hybrid high radix encoding for energy-efficient inexact multipliers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 3, pp. 421–430, Mar. 2018.
[20] H. Waris, C. Wang, W. Liu, and F. Lombardi, "AxBMs: Approximate radix-8 booth multipliers for high-performance FPGA-based accelerators," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 5, pp. 1566–1570, May 2021.