Accelerating Conv2D and DepthwiseConv2D Tensor Operations for TensorFlow Lite on Embedded FPGA with Hybrid Custom Floating-Point Approximation

1st
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

2nd
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

3rd
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

***Abstract*—Convolutional neural networks (CNNs) have become ubiquitous in the field of image processing, computer vision, and artificial intelligence (AI). Given the high computational demands of CNNs, dedicated hardware accelerators have been implemented to improve compute performance in FPGAs and ASICs. However, most commercial general-purpose deep learning processing units (DPUs) struggle with support for low-power, resource-limited embedded devices. In this paper, we present a tensor processor (TP) as a dedicated hardware accelerator for TensorFlow (TF) Lite on embedded FPGA. We accelerate Conv2D and DepthwiseConv2D tensor operations with fixed-point and floating-point. The proposed compute optimization performs vector dot-product with hybrid custom floating-point and logarithmic approximation. This approach accelerates computation, reduces energy consumption and resource utilization. To demonstrate the potential of the proposed architecture, we address a design exploration with four compute engines: (1) fixed-point, (2) Xilinx floating-point LogiCORE IP, (3) hybrid custom floating-point approximation, and (4) hybrid logarithmic approximation. The hardware design is implemented with high-level synthesis (HLS). A single TP running at 150 MHz on a Xilinx Zynq-7020 achieves 45X runtime acceleration and 954X power reduction on Conv2D tensor operation compared with ARM Cortex-A9 at 666MHz, and 4.59X compared with the equivalent implementation with floating-point LogiCORE IP. The entire hardware design and the implemented TF Lite software extensions are available as an open-source project.**

***Index Terms*—Artificial intelligence, convolutional neural networks, depthwise separable convolution, hardware accelerator, TensorFlow Lite, embedded systems, FPGA, custom floating-point, logarithmic computation, approximate computing**

I. Introduction

THE constant research and the rapid evolution of artificial neural networks (ANNs) are driving the transition to smarter and more powerful AI applications, where CNN-based models represent the essential building blocks of deep learning algorithms in computer vision tasks [1]. Applications such as smart surveillance, medical imaging, natural language processing, robotics, and autonomous navigation have been powered by CNN-based models in industry and academia [2]. Nonetheless, dedicated hardware is often required to accelerate execution due to the high computational demands of CNNs. In terms of pure computational throughput, graphics processing units (GPUs) offer the best performance. In terms of power consumption, FPGA solutions are well known to be more energy efficient (than GPUs) [3]. As a result, numerous FPGA accelerators have been proposed, targeting both high performance computing (HPC) for data-centers and embedded systems applications [4]–[6]. However, most commercial deep learning processing units (DPUs) are not designed for low-power, resource-limited embedded FPGAs.

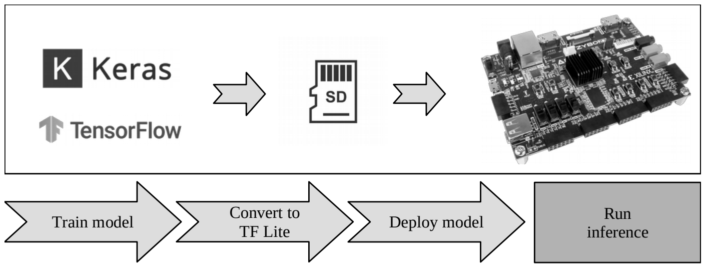

Fig. 1. Deployment workflow.

In this paper, we present a tensor processor compatible with TensorFlow Lite to accelerate *Conv2D* and *DepthwiseConv2D* operations on embedded FPGA. This implementation is integrated in a hardware/software co-design framework to accelerate tensor operations on FPGAs. This framework integrates TensorFlow Lite library and implements delegate interfaces [7] as bridge between the TF Lite runtime and the proposed hardware architecture. To control resource utilization and energy consumption, we implement the tensor operations as individual hardware engines, where they are optionally instantiated in the FPGA fabric as needed. Further on, to accelerate floating-point computation, we adopt the hybrid custom floating-point and logarithmic dot-product approximation technique [8], which exploits the intrinsic error-resilience of neural networks [9]. This approach can efficiently trade off quality-of-result (QoR) and resource utilization.

To operate the proposed system, the user would train a custom CNN model using TensorFlow or Keras, then this is converted into a TensorFlow Lite model, then the model is stored in a micro SD card along with the embedded software and configuration bitstream. See **Fig.** 1.

Our main contributions are as follows:

1) We present a tensor processor as a dedicated hardware accelerator for TensorFlow Lite on embedded FPGA. We accelerate *Conv2D* and *DepthwiseConv2D* tensor operations with fixed-point and floating-point computation.
2) We develop a hardware/software co-design framework targeting low-power and resource-constrained AI applications. The parameterized and modular architecture enables design exploration with different compute hardware approaches.
3) We demonstrate the potential of the proposed architecture by address a design exploration of *Conv2D* and *DepthwiseConv2D* operations with four compute engines: (1) fixed-point, (2) floating-point LogiCORE, (3) hybrid custom floating-point approximation, and (4) hybrid logarithmic approximation. We evaluate compute performance and classification accuracy implementing half-precision, brain floating-point, TensorFloat, and custom reduced formats for approximate processing, including logarithmic computation. Detailed performance reports are presented.

To promote the research in this field, our entire work is made available to the public as an open-source project at X.

II. Related work

A. Google's Edge TPU

The Edge Tensor Processing Unit (TPU) is an ASIC designed by Google that provides high performance machine learning (ML) inference for TensorFlow Lite models [10]. This implementation uses PCIe and I2C/GPIO to interface with an iMX 8M system-on-chip (SoC). The reported throughput and power efficiency are 4 TOPS and 2 TOPS per watt, respectively [11]. The Edge TPU supports 40 tensor operations including *Conv2D* and *DepthwiseConv2D* [12].

However, the Edge TPU does not support floating-point computation. The Edge TPU supports only TensorFlow Lite models that are 8-bit quantized and then compiled specifically for the Edge TPU [13]. Regarding power dissipation, the Edge TPU system-on-module (SoM) requires up to $15W$ power supply [11], which can be inadequate for very low-power applications.

B. Xilinx Zynq DPU

The Xilinx deep learning processing unit (DPU) is a configurable computation engine optimized for CNNs. The degree of parallelism utilized in the engine is a design parameter and can be selected according to the target device and application. The DPU IP can be implemented in the programmable logic (PL) of the selected Zynq-7000 SoC or Zynq UltraScale+ MPSoC device with direct connections to the processing system (PS) [14]. The peak theoretical performance reported on Zynq-7020 is 230 GOP/s.

However, the DPU does not support floating-point computation. The DPU requires the CNN model to be quantized, calibrated, converted into a deployable model, and then compiled into the executable format [14].

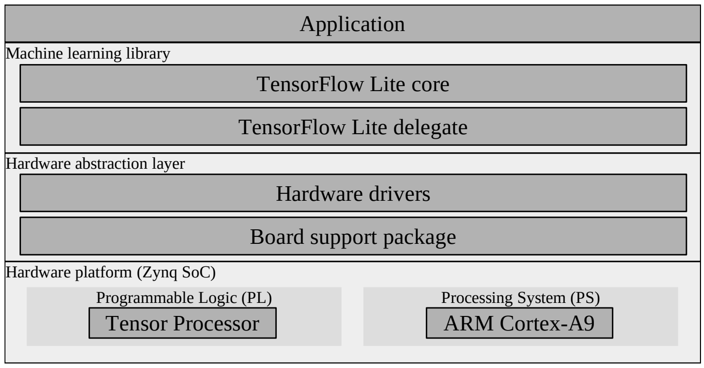

Fig. 2. System-level overview of the proposed embedded software stack.

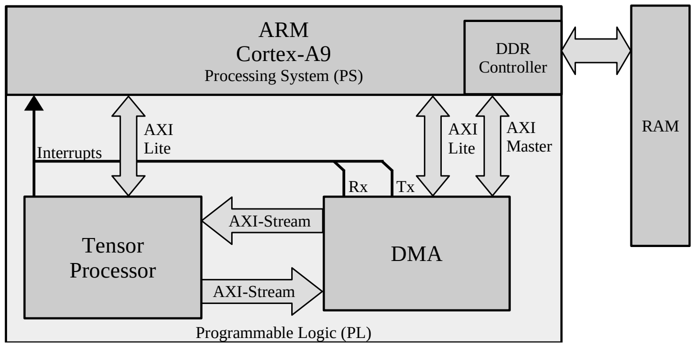

Fig. 3. System-level architecture of the proposed embedded platform.

III. Background

A. Conv2D tensor operation

The *Conv2D* tensor operation is described in **Eq.** (1). Where X represents the input feature maps, W represents the convolution kernel (known as filter) and b represents the bias for the output feature maps [15].

$$Conv2D\,(W,X)_{i,j,o} = \sum_{k,l,m}^{K,L,M} W_{(o,k,l,m)} \cdot X_{(i+k,j+l,m)+b_o} \quad (1)$$

B. DepthwiseConv2D tensor operation

The *DepthwiseConv2D* tensor operation is described in **Eq.** (2). Where X represents the input feature maps, W represents the convolution kernel (known as filter), and b represents the bias for the output feature maps.

$$DConv2D\,(W,X)_{i,j,n} = \sum_{k,l}^{K,L} W_{(k,l,n)} \cdot X_{(i+k,j+l,n)} + b_n \quad (2)$$

IV. System Design

The proposed system architecture is a hardware/software framework to investigate tensor acceleration with target on Zynq devices. The SoC architecture is illustrated in **Fig.** 3. The software stack is shown in **Fig.** 2.

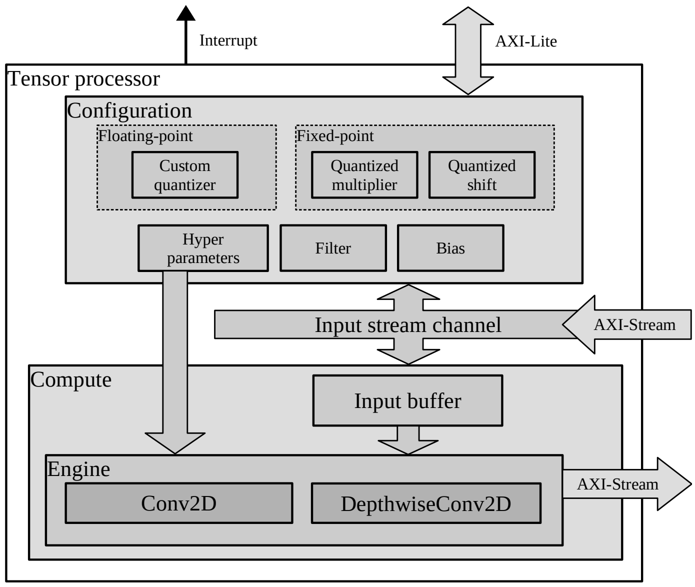

Fig. 4. Hardware architecture of the proposed tensor processor.

a) Tensor processor: The TP is a dedicated hardware module to compute tensor operations. The hardware architecture is described in **Fig.** 4. This architecture implements high performance off-chip communication with AXI-Stream, direct CPU communication with AXI-Lite, and on-chip storage utilizing BRAM. This hardware architecture is implemented with HLS. The tensor operations are implemented based on the C++ TF Lite micro kernels [16].

b) Modes of operation: This accelerator offers two modes of operation: *configuration* and *execution*.

In configuration mode, the TP receives the operator ID and the hyperparameters for execution: stride, dilation, padding, offset, activation, quantized activation, depth-multiplier, input shape, filter shape, bias shape, and output shape. Afterwards the accelerator receives the *filter*, *bias* and quantization vectors.

In execution mode, the TP executes the tensor operator according to the hyperparameters given in the configuration mode. During execution, the input and output tensor buffers are moved from/to the TF Lite memory regions via direct memory access (DMA).

c) Compatibility: This TP is compatible with TF Lite 8-bit quantized models and standard floating-point. For this purpose, we implement the compute engines with regular fixed-point and floating-point LogiCORE IPs. Vivado HLS accomplishes floating-point arithmetic operations by mapping them onto Xilinx LogiCORE IP cores, these floating-point operator cores are instantiated in the resultant RTL [17].

d) Dot-product with floating-point optimization: We optimize the floating-point computation adopting the dot-product with hybrid custom floating-point and logarithmic approximation [8]. This approach: (1) denormalizes input numbers, (2) executes computation with integer format for exponent and mantissa, and finally, (3) it normalizes the result into IEEE 754 format. This design implements a pipelined vector dot-product

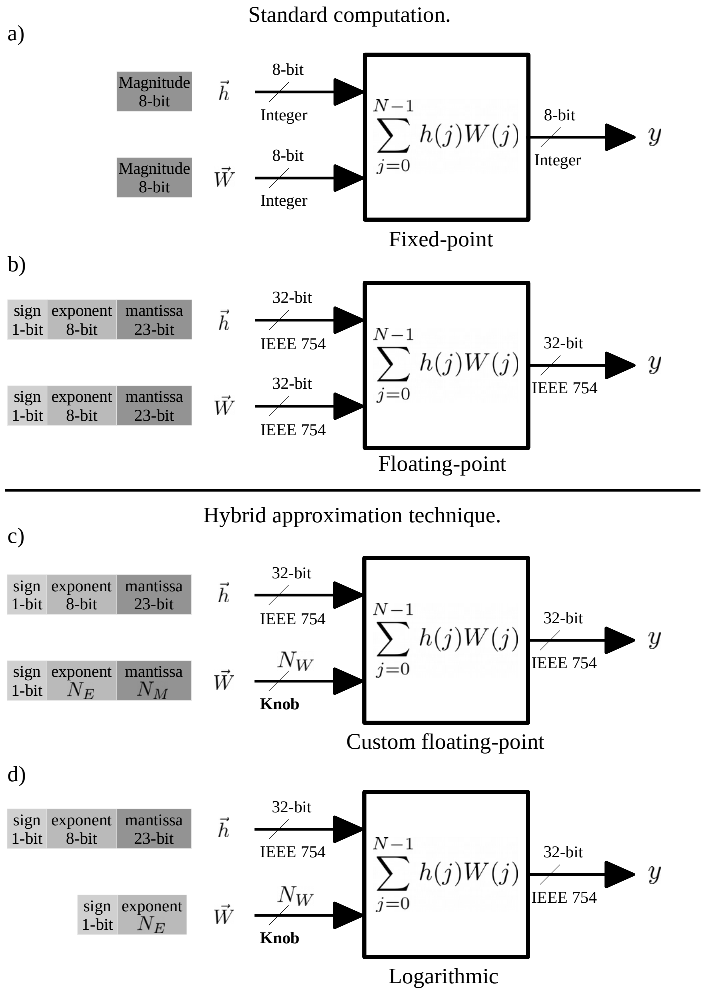

Fig. 5. Hardware alternatives for vector dot-product.

with a latency of $2N + II$, where N and II are the vector length and initiation interval, respectively. This implementation achieves up to $5\times$ latency reduction compared with a pipelined vector dot-product using Xilinx floating-point LogiCORE [8]. The hardware dot-product is illustrated in **Fig.** 5. As a design parameter, the mantissa bit-width of the weight vector provides a tunable knob to trade-off between resource-efficiency and QoR [18]. Since the lower-order bits have smaller significance than the higher-order bits, truncating them may have only a minor impact on QoR [19].

V. Experimental results

The proposed hardware/software framework is demonstrated on a Xilinx Zynq-7020 SoC (Zybo-Z7 development board). On the PL, we implement the proposed hardware architecture with a clock frequency at $150MHz$. On the PS, we execute TF Lite Micro (bare-metal) on the ARM Cortex-A9 at $666MHz$ with NEON floating-point unit (FPU) [20].

To evaluate the performance, we build models A and B in TensorFlow, see **Fig.** 6. To evaluate *DConv* tensor operation, model B incorporates depthwise separable convolution

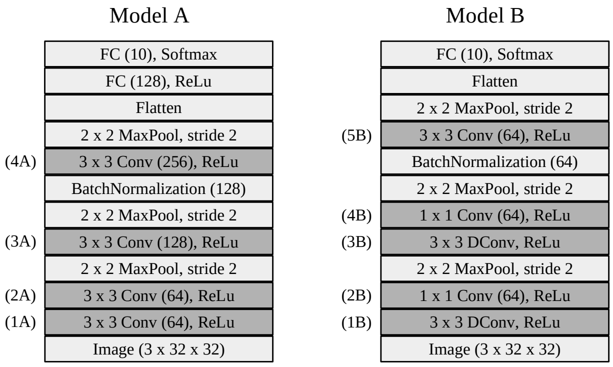

Fig. 6. CNN-based models for case study.

TABLE I
COMPUTE PERFORMANCE WITH FIXED-POINT ON MODEL A AND B.

Tensor operation		CPU	TP (fixed-point)			Accel.
Operation	**MOP**	**t (ms)**	**t (ms)**	**MOP/s**	**GOP/W**	
Model A						
(1A) Conv	1.769	700.22	55.19	32.06	0.23	**12.69**
(2A) Conv	37.748	12,666.91	297.08	127.06	0.93	**42.64**
(3A) Conv	18.874	6,081.01	142.99	131.99	0.97	**42.53**
(4A) Conv	18.874	5,543.77	122.58	153.97	1.13	**45.23**
Model B						
(1B) DConv	0.027	13.43	0.63	43.74	0.25	**21.25**
(2B) Conv	0.196	129.95	11.57	16.98	0.12	**11.23**
(3B) DConv	0.147	69.18	3.33	44.26	0.25	**20.77**
(4B) Conv	1.048	378.78	9.96	105.25	0.77	**38.02**
(5B) Conv	2.359	694.60	16.46	143.22	1.05	**42.20**

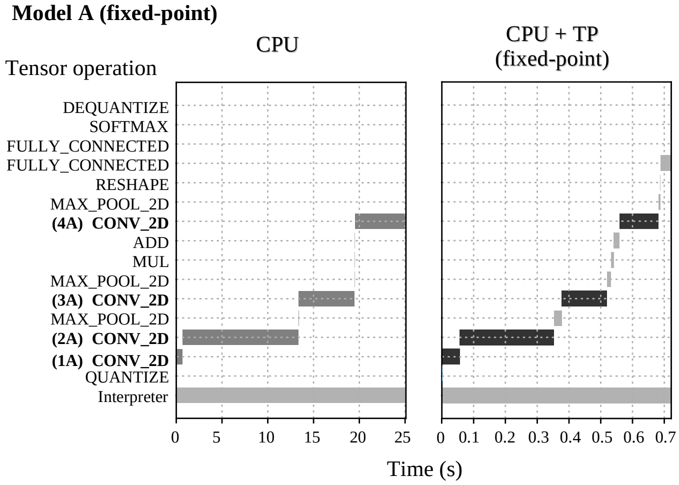

Fig. 7. Compute performance with fixed-point on model A.

operations (a depthwise convolution followed by a pointwise convolution).

A and B are evaluated with the following hardware implementations: (1) fixed-point, (2) floating-point LogiCORE, (3) hybrid custom floating-point approximation, and (4) hybrid logarithmic approximation.

A. Hardware implementations

1) Fixed-point: To evaluate the compute performance on fixed-point, we convert A and B to TF Lite models with 8-bit fixed-point quantization. The compute performance is presented in **Tab.** I. A runtime execution of A is illustrated in **Fig.** 7. This implementation achieves a peak acceleration of $45.23\times$ in model A at the tensor operation *(4A) Conv*, see **Tab.** I.
2) Floating-point LogiCORE: To evaluate the compute performance on floating-point models, we convert A and B to TF Lite without quantization. The compute performance is presented in **Tab.** II. This implementation achieves a peak acceleration of $9.77\times$ in model A at the tensor operation *(4A) Conv*.
3) Hybrid custom floating-point approximation: This implementation presents a peak acceleration of $44.87\times$ in model A at the tensor operation *(4A) Conv*. See **Tab.** III. This implementation achieves a $4.59\times$ acceleration over the LogiCORE floating-point implementation. The runtime execution of model B with *DConv* tensor operations is illustrated in **Fig.** 9.
4) Hybrid logarithmic approximation: This implementation is presented for comparison in **Fig.** 8, which shows the runtime executions of model A with the proposed floating-point solutions including hybrid logarithmic approximation.

TABLE II
COMPUTE PERFORMANCE WITH FLOATING-POINT LOGICORE ON MODELS A AND B.

Tensor operation		CPU	TP (floating-point LogiCORE)			Accel.
Operation	**MOP**	**t (ms)**	**t (ms)**	**MOP/s**	**GOP/W**	
Model A						
(1A) Conv	1.769	670.95	120.07	14.73	0.21	**5.59**
(2A) Conv	37.748	12,722.13	1,328.08	28.42	0.40	**9.58**
(3A) Conv	18.874	6,094.85	636.53	29.65	0.42	**9.58**
(4A) Conv	18.874	5,564.79	569.30	33.15	0.47	**9.77**
Model B						
(1B) DConv	0.027	11.51	1.557	17.75	0.23	**7.39**
(2B) Conv	0.196	94.82	20.487	9.59	0.13	**4.62**
(3B) DConv	0.147	58.84	8.355	17.64	0.23	**7.04**
(4B) Conv	1.048	368.66	40.271	26.03	0.37	**9.15**
(5B) Conv	2.359	697.08	72.981	32.32	0.46	**9.55**

B. Classification accuracy

We evaluate the classification accuracy of the CNN models under the effects of custom floating and logarithmic quantization. **Tab.** IV presents the list of custom formats proposed for evaluation. In this case, the *filter* and *bias* tensors are quantized from base floating-point representation (IEEE 754) into custom reduced formats with bit-truncation and -rounding methods. For this evaluation, we train A an B for image classification with CIFAR-10 dataset. We deploy the models with a baseline accuracy of 76.6% for A, and 68.8% for B. See **Fig.** 10.

C. Resource utilization and power dissipation

The resource utilization and power dissipation of the TP is listed in **Tab.** V. The power dissipation of the Zynq device is presented in **Fig.** 11.

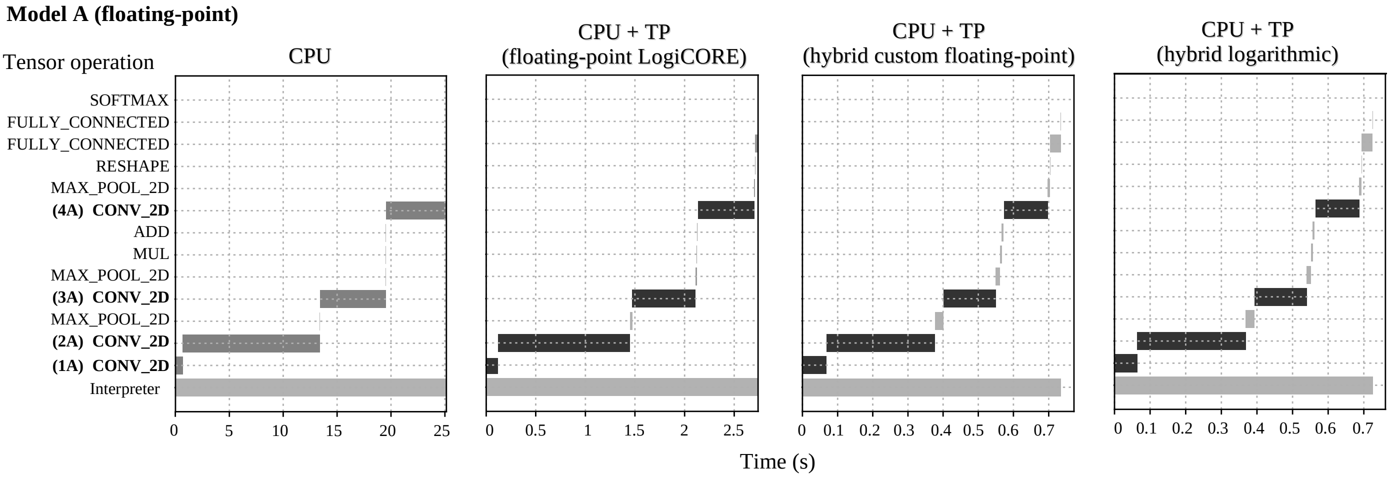

Fig. 8. Compute performance with the proposed floating-point solutions on model A.

TABLE III
COMPUTE PERFORMANCE WITH HYBRID CUSTOM FLOATING-POINT APPROXIMATION ON MODELS A AND B.

Tensor operation		**CPU**	**TP (H. custom floating-point)**			**Accel.**
Operation	**MOP**	**t (ms)**	**t (ms)**	**MOP/s**	**GOP/W**	
Model A						
(1A) Conv	1.769	670.95	68.50	25.83	0.39	**9.8**
(2A) Conv	37.748	12,722.13	307.83	122.63	1.85	**41.33**
(3A) Conv	18.874	6,094.85	147.97	127.55	1.93	**41.19**
(4A) Conv	18.874	5,564.79	124.03	152.17	2.30	**44.87**
Model B						
(1B) DConv	0.027	11.51	1.41	19.63	0.27	**8.17**
(2B) Conv	0.196	94.82	20.34	9.43	0.14	**4.66**
(3B) DConv	0.147	58.84	6.58	22.41	0.31	**8.94**
(4B) Conv	1.048	368.66	12.75	82.23	1.24	**28.91**
(5B) Conv	2.359	697.08	17.14	137.68	2.08	**40.68**

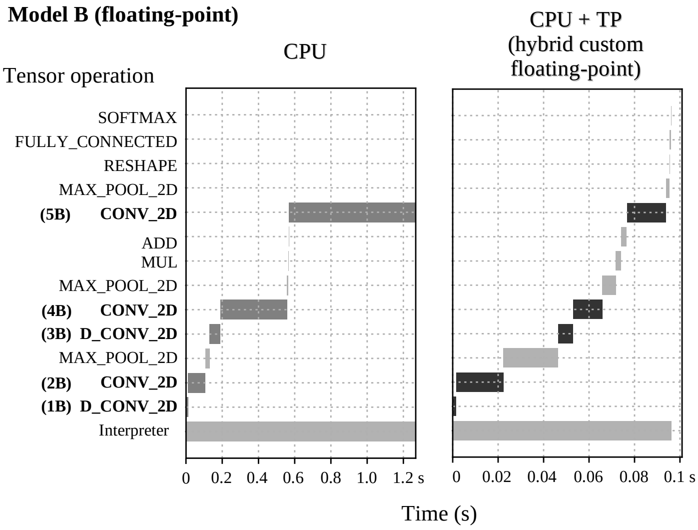

Fig. 9. Compute performance on model B (floating-point).

D. Discussion

1) Acceleration: The dot-product with hybrid custom floating-point approximation achieves better performance with larger vectors. Since *DConv* performs a channel-wise spatial dot-product, this implementation presents very limited acceleration compared with the LogiCORE-based implementation.

2) Energy: The implementations with hybrid custom floating-point and logarithmic approximation are the most efficient with energy reduction of $954\times$ and $1,055\times$, respectively. **Tab.** VI presents the energy-delay product (EDP) and energy reduction in *(4A) Conv* operator.

3) Resource utilization: In the case of fixed-point implementation, the TP includes additional logic to support the quantized multiplier and shifter channel-wise corrections

TABLE IV
IMPLEMENTED FLOATING-POINT FORMATS FOR ACCURACY EVALUATION.

Floating-point formats				
Name	**Size (bits)**	**Sign**	**Exponent**	**Mantissa**
Logarithmic	6	1	5	0
S1-E5-M1	7	1	5	1
S1-E5-M2	8	1	5	2
S1-E5-M3	9	1	5	3
S1-E5-M4	10	1	5	4
Float16	16	1	5	10
BFloat16	16	1	8	7
Tensor Float	19	1	8	10
Float32	32	1	8	23

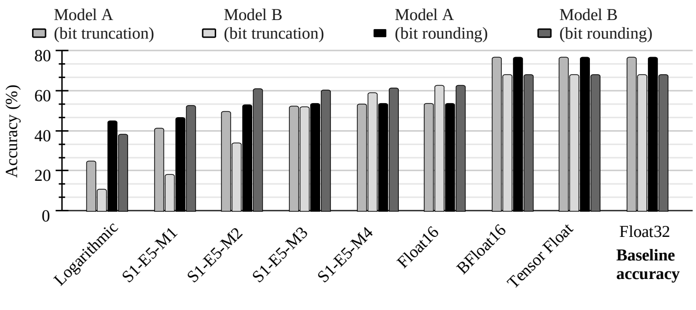

Fig. 10. Accuracy performance using hybrid custom floating-point approximation with various formats. Samples: CIFAR-10 test dataset ($10,000$ images).

TABLE V
RESOURCE UTILIZATION AND POWER DISSIPATION OF THE PROPOSED TP ENGINES.

TP engine	Post-implementation resource utilization LUT	FF	DSP	BRAM 18K	Power (W)
1) Fixed-point					
Conv	5,677	4,238	78	70	0.136
DConv	7,232	5,565	106	70	0.171
Conv + DConv	12,684	8,015	160	70	0.248
2) Floating-point LogiCore					
Conv	4,670	3,909	59	266	0.070
DConv	6,263	5,264	82	266	0.075
Conv + DConv	10,871	7,726	123	266	0.119
3) Hybrid custom floating-point approximation					
Conv	6,787	4,349	56	74	0.066
DConv	8,209	5,592	79	74	0.072
Conv + DConv	14,590	8,494	117	74	0.108
4) Hybrid logarithmic approximation					
Conv	6,662	4,242	54	58	0.060
DConv	8,110	5,380	77	58	0.066
Conv + DConv	14,370	8,175	113	58	0.105

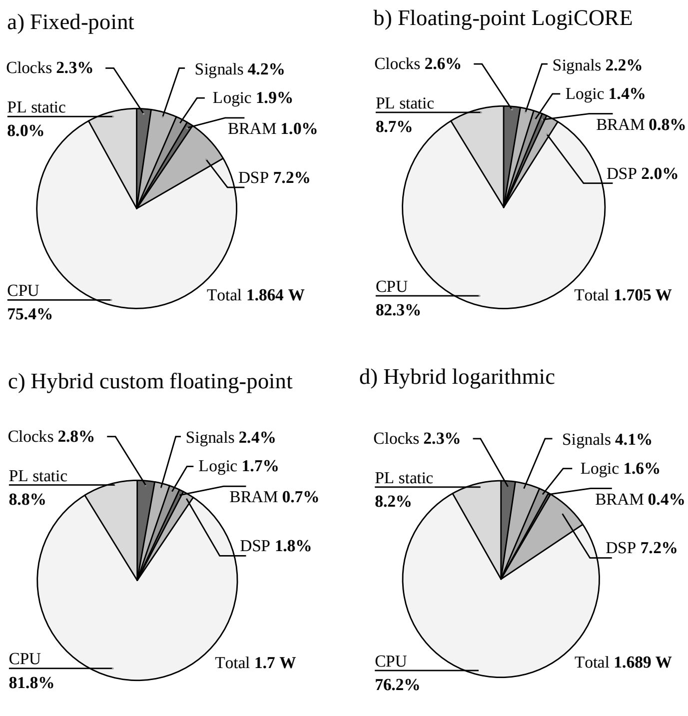

Fig. 11. Estimated power dissipation of the Zynq-7020 SoC with different TP engines.

TABLE VI
ENERGY CONSUMPTION IN TENSOR OPERATION *(4A) Conv*.

Engine	t (ms)	Power (W)	EDP (J)	Reduction
CPU	5,564.79	1.404	7,812.97	1.00
Fixed-point	122.58	0.136	16.67	468.66
Floating-point LogiCORE	569.30	0.070	39.85	196.05
Hybrid custom floating-point	124.03	0.066	8.19	**954.43**
Hybrid logarithmic	123.32	0.060	7.40	**1,055.92**

implemented for TF Lite 8-bit quantization. This TP presents the highest power dissipation.

4) Accuracy: Based on the presented classification accuracy, the hybrid custom floating-point approximation presents the best trade off between QoR and energy-efficiency. Theres is no accuracy degradation when using fixed-point and floating-point LogiCORE IP.

VI. CONCLUSIONS

In this paper, we present a tensor processor as a dedicated hardware accelerator for TensorFlow Lite on embedded FPGA. We accelerate *Conv2D* and *DepthwiseConv2D* tensor operations for fixed-point and floating-point computation. The proposed optimization technique performs vector dot-product with hybrid custom floating-point and logarithmic approximation. This approach accelerates computation, reduces energy consumption and resource utilization. To demonstrate the potential of the proposed architecture, we presented a design exploration with four compute engines: (1) fixed-point, (2) Xilinx floating-point LogiCORE IP, (3) hybrid custom floating-point approximation, and (4) hybrid logarithmic approximation.

A single tensor processor running at 150 MHz on a Xilinx Zynq-7020 achieves $45\times$ runtime acceleration and $954\times$ power reduction on $Conv2D$ tensor operation compared with ARM Cortex-A9 at 666MHz, and $4.59\times$ compared with the equivalent implementation with floating-point LogiCORE IP.

REFERENCES

[1] M. Hassaballah and A. I. Awad, *Deep learning in computer vision: principles and applications*. CRC Press, 2020.
[2] A. Dhillon and G. K. Verma, “Convolutional neural network: a review of models, methodologies and applications to object detection,” *Progress in Artificial Intelligence*, vol. 9, no. 2, pp. 85–112, 2020.
[3] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra *et al.*, “Can fpgas beat gpus in accelerating next-generation deep neural networks?” in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 5–14.
[4] K. Abdelouahab, M. Pelcat, J. Serot, and F. Berry, “Accelerating cnn inference on fpgas: A survey,” *arXiv preprint arXiv:1806.01683*, 2018.
[5] S. Moini, B. Alizadeh, M. Emad, and R. Ebrahimpour, “A resource-limited hardware accelerator for convolutional neural networks in embedded vision applications,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 10, pp. 1217–1221, 2017.
[6] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, “Angel-eye: A complete design flow for mapping cnn onto embedded fpga,” *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 37, no. 1, pp. 35–47, 2017.
[7] “Tensorflow lite delegates.” https://www.tensorflow.org/lite/performance/.
[8] Y. Nevarez, D. Rotermund, K. R. Pawelzik, and A. Garcia-Ortiz, “Accelerating spike-by-spike neural networks on fpga with hybrid custom floating-point and logarithmic dot-product approximation,” *IEEE Access*, 2021.
[9] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, “Approximate computing and the quest for computing efficiency,” in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2015, pp. 1–6.
[10] A. Yazdanbakhsh, K. Seshadri, B. Akin, J. Laudon, and R. Narayanaswami, “An evaluation of edge tpu accelerators for convolutional neural networks,” *arXiv preprint arXiv:2102.10423*, 2021.
[11] “Coral. dev board datasheet.,” https://coral.ai/docs/dev-board/datasheet/., accessed September 15, 2021.
[12] “Coral. tensorflow models on the edge tpu.,” https://coral.ai/docs/edgetpu/models-intro/., accessed September 15, 2021.
[13] S. Cass, “Taking ai to the edge: Google’s tpu now comes in a maker-friendly package,” *IEEE Spectrum*, vol. 56, no. 5, pp. 16–17, 2019.
[14] P. Xilinx, “Zynq dpu v3.1, product guide,” 2019.

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
[16] "Tensorflow lite for microcontrollers," https://github.com/tensorflow/tflite-micro.
[17] J. Hrica, "Floating-point design with vivado hls," *Xilinx Application Note*, 2012.
[18] J. Park, J. H. Choi, and K. Roy, "Dynamic bit-width adaptation in dct: An approach to trade off image quality and computation energy," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 18, no. 5, pp. 787–793, 2009.
[19] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, pp. 1–33, 2016.
[20] U. Xilinx, "Zynq-7000 all programmable soc: Technical reference manual," 2015.