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

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

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

3rd Given Name Surname
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 efficiency 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 publication, we present a dedicated hardware accelerator for TensorFlow (TF) Lite on embedded FPGA emulating an Edge TPU coprocessor to delegate Conv2D and DepthwiseConv2D tensor operations. The hardware design is implemented with high-level synthesis (HLS). This accelerator incorporates the support for TF Lite quantization for fixed-point and floating-point. The proposed optimization decomposes floating-point calculation for the dot-product. This approach accelerates computation, reduces energy consumption and resource utilization. To demonstrate the potential of the proposed accelerator, we address a design exploration with custom-built CNNs covering fixed-point quantization, floating-point single precision, half-precision, brain floating-point, TensorFloat, and custom reduced formats for approximate processing, including logarithmic computation. A single accelerator running at 150 MHz on a Xilinx Zynq-7020 achieves 45X runtime acceleration on Conv2D tensor operation compared with ARM Cortex-A9 at 666MHz, and 5X compared with the equivalent standard floating-point implementation in HLS with Xilinx 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 network (ANN) architectures 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 a requirement 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]. However, most commercial deep learning processing units (DPUs) are not designed for low-power, resource-limited embedded FPGAs.

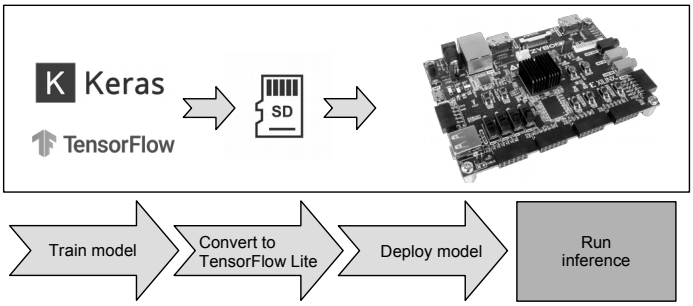

Fig. 1. Deployment workflow.

In this paper, we present a hardware/software co-design framework to accelerate tensor operators for TF Lite Micro on resource-constrained and low-power embedded FPGAs. Based on their high computational cost, in this paper, we accelerate Conv2D and DepthwiseConv2D operators. The proposed architecture supports TF Lite with 8-bit fixed-point quantization and standard floating-point. To enhance resource utilization and energy consumption, the tensor operators are designed as separate hardware engines, where they are optionally instantiated in the FPGA fabric. Further on, to accelerate floating-point computation, we propose a decomposed floating-point calculation for the vector dot-product. To enhance on-chip memory usage and for quality reconfigurability, the proposed design employs hybrid custom floating-point formats, where filter and bias tensors are quantized and stored with a reduced format on the hardware accelerator. The hardware design is implemented with high-level synthesis (HLS).

To operate the proposed accelerator framework, we train a CNN model on TensorFlow or Keras, then this is converted into a TensorFlow Lite model with either 8-bit fixed-point quantization or floating-point, then the model is stored in a micro SD card along with the software and the FPGA configuration bitstream for deployment. See **Fig.** 1.

Our main contributions are as follows:

1) We develop a dedicated hardware accelerator for TensorFlow Lite on embedded FPGA emulating an Edge TPU coprocessor to delegate Conv2D and DepthwiseConv2D tensor operations.
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 for different target applications.
3) We demonstrate the potential of the proposed framework addressing a design exploration with two custom-built CNN architectures trained for CIFAR-10 image classification. We demonstrate the performance of the Conv2D and DepthwiseConv2D operation engines with fixed-pint quantization, floating-point with Xilinx LogiCORE IP, decomposed floating-point with single precision, half-precision, brain floating-point, TensorFloat, and custom reduced formats for approximate processing, including logarithmic computation. We present a detailed runtime and accuracy performance reports.

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

II. Related work

A. TensorFlow models on the 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 [5]. This implementation uses PCIe and I2C/GPIO to interface with an iMX 8M SoC. The reported throughput and power efficiency are 4 trillion operations per second (TOPS) and 2 TOPS per watt, respectively [6]. The Edge TPU supports 40 tensor operators including Conv2d and DepthwiseConv2d [7].

However, the Edge TPU has disadvantages.

- Power dissipation: The Edge TPU System-on-Module (SoM) requires up to 3A at 5V DC power supply [6], which can be unsuitable for very low-power applications.
- **Model compatibility**: It supports only TensorFlow Lite models that are fully 8-bit quantized and then compiled specifically for the Edge TPU [8]. The 8-bit quantization method requires a representative dataset that can be inaccessible, which limits its usability.

III. Background

A. Conv2D tensor operation

$$Conv2D\,(W,y) = \sum_{k,l,m}^{K,L,M} W_{(k,l,m)} \cdot y_{(i+k,j+l,m)} \qquad (1)$$

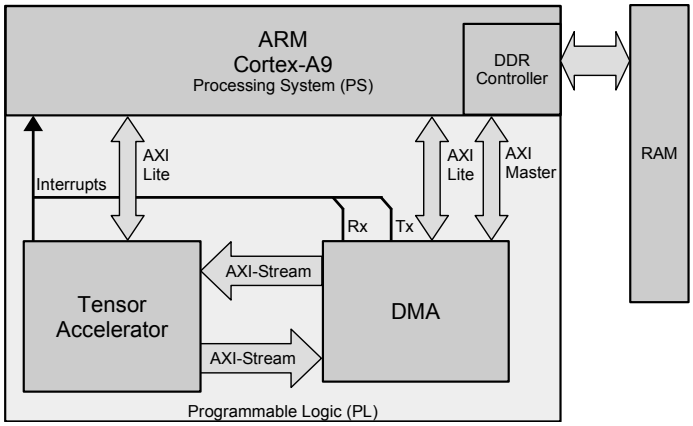

Fig. 2. System-level architecture of the proposed embedded platform.

B. DepthwiseConv2D tensor operation

$$DepthwiseConv2D\,(W,y) = \sum_{k,l}^{K,L} W_{(k,l)} \odot y_{(i+k,j+l)} \qquad (2)$$

IV. System Design

A. Embedded system architecture

The system architecture is an embedded CPU+FPGA-based platform illustrated in **Fig.** 2. The CPU communicates with the tensor accelerator (TA) trough AXI-Lite, and the TA communicates to the external memory though AXI-Stream interface via Direct Memory Access (DMA).

B. Tensor accelerator

The tensor accelerator is a dedicated hardware module to compute tensor operations. In this paper, we present Conv2D and DepthwiseConv2D operators implemented using HLS with an approximate computing approach. For each operator, the compute engine represents the TFLite micro kernel C++ code adapted for hardware HLS. The accelerator architecture is illustrated in **Fig.** 3.

This accelerator offers two modes of operation: *configuration* and *computation*.

1) Setup mode: In this mode, the accelerator receives the tensor 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: filter tensor, bias tensor, and quantized multiplier/shift vectors in the case of using 8-bit quantization. These hyperparameters and tensors are stored in on-chip memory for reusage during compute mode.

2) Computation mode: In this mode, the accelerator performs the tensor operator according to the hyperparameters received during setup mode.

C. Compatibility

This hardware accelerator is compatible with TF Lite 8-bit quantized and floating-point formats. The designer is able to select the compute and number formats prior hardware synthesis.

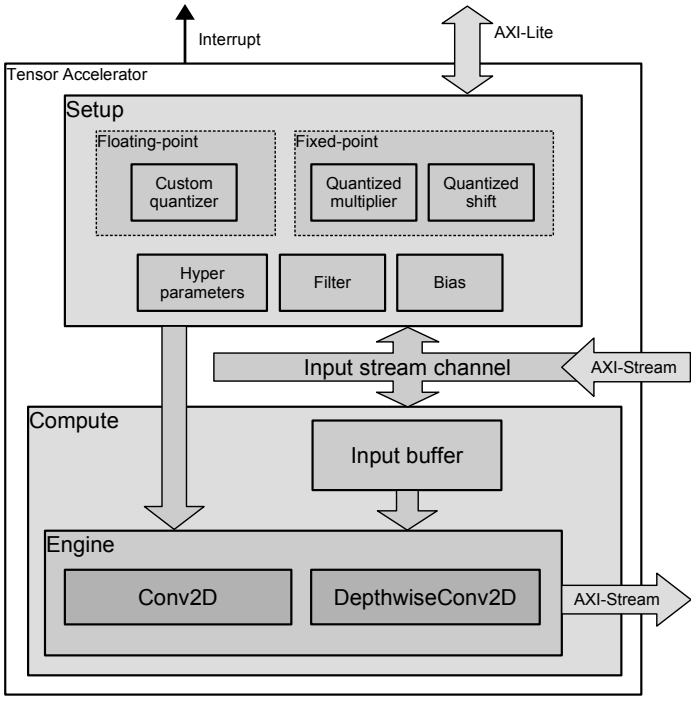

Fig. 3. Hardware architecture of the proposed accelerator.

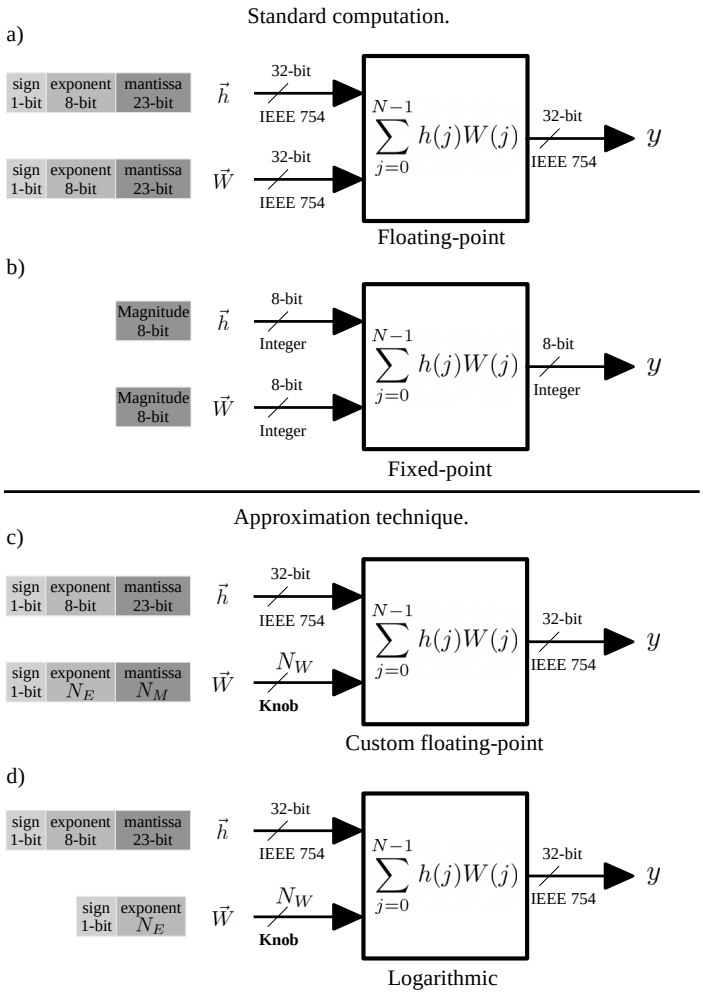

Fig. 4. Hardware alternatives for vector dot-product.

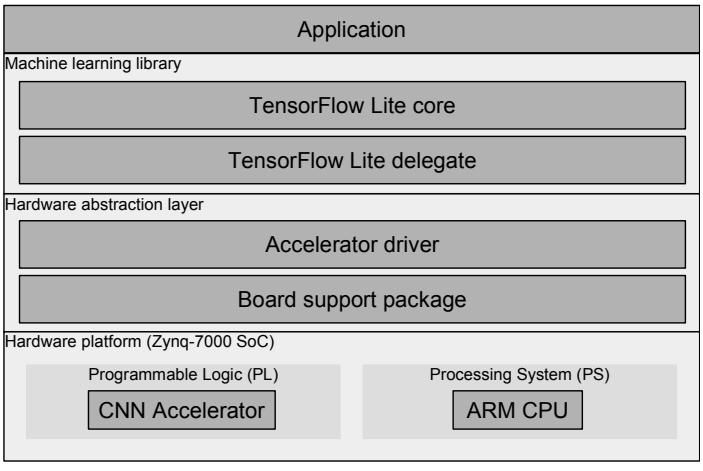

Fig. 5. System-level overview of the embedded software architecture.

D. Dot-product computation

Internally, the accelerator performs floating-point computation employing either: (1) HLS Xilinx LogiCORE IP, (2) decomposed custom floating-point, and (3) logarithmic computation. This design is proposed in [9], where the authors present the hybrid custom floating-point and logarithmic dot-product approximation to accelerate Spike-by-Spike neural networks on embedded FPGA. In this paper, we adopt this approach to accelerate the vector dot-product on the convolution operators. With this approach, the designer is able to select the desired vector dot-product. See **Fig.** 4.

E. Embedded software architecture

The software architecture is an extended version of TFLite for microcontrollers [10]. We implemented a delegate class for TFLite. This class encapsulates the accelerator and DMA low-level drivers. The TFLite library delegates the Conv2D and DepthwiseConv2D tensor operations to the hardware accelerator, while the rest of the tensor operators are executed on the embedded CPU. At the application level, the end user would use the regular TFLite application programming interface (API) to load TFLite models, set input tensors, invoke inference, and get output tensors. The overall structure is depicted in **Fig.** 5.

V. Experimental results

The proposed hardware/software framework is demonstrated on a Xilinx Zynq-7020 SoC (Zybo-Z7 development board). We deploy TFLite micro on the processing system (PS), and we implement the proposed hardware architecture on the programmable logic (PL). We address a design exploration by building and training custom CNN models in TF, and we evaluating their computational and accuracy performance on the proposed hardware architecture.

build and train two CNN-based models for CIFAR-10 dataset classification

The PS ARM Cortex-A9 CPU at 666MHz, this processor includes NEON SIMD Engine and single/double precision floating-point unit (FPU) [11].

In this platform, we implement the proposed hardware/software solution to deploy TFLite models on the Zynq-7020 network structure shown in **Fig. ??** for handwritten digit classification task using MNIST data set. The SbS model is trained in Matlab without any quantization method, using standard floating-point. The resulting synaptic weight matrices are deployed on the embedded system. There, the SbS network is built as a sequential model using the API from the SbS embedded software framework [**?**]. This API allows to configure the computational workload of the neural network, which can be distributed among the hardware processing units and the embedded CPU.

For the evaluation of our approach, we address a design exploration by reviewing the computational latency, inference accuracy, resource utilization, and power dissipation. First, we benchmark the performance of SbS network simulation on the embedded CPU, and then repeat the measurements on hardware processing units with standard floating-point computation. Afterwards, we evaluate our dot-product architecture, addressing a design exploration with hybrid custom floating-point approximation, as well as the hybrid logarithmic approximation. Finally, we present a discussion of the presented results.

A. Computational performance

TABLE I
COMPUTE PERFORMANCE ON FIXED-POINT IMPLEMENTATION

Tensor operation		**CPU**	**HW (8-bit fixed-point)**			**Gain**
Operation	**MOP**	**RT (ms)**	**RT (ms)**	**MOP/s**	**GOP/W**	
Model A						
Conv	1.769	700.22	55.19	32.06	2.09	**12.69**
Conv	37.748	12,666.91	297.08	127.06	8.30	**42.64**
Conv	18.874	6,081.01	142.99	131.99	8.60	**42.53**
Conv	18.874	5,543.77	122.58	153.97	10.06	**45.23**
Model B						
DConv	0.027	13.43	0.63	43.74	2.85	**21.25**
Conv	0.196	129.95	11.57	16.98	1.11	**11.23**
DConv	0.147	69.18	3.33	44.26	2.89	**20.77**
Conv	1.048	378.78	9.96	105.25	6.87	**38.02**
Conv	2.359	694.60	16.46	143.22	9.36	**42.20**

1) Fixed-point:

TABLE II
COMPUTE PERFORMANCE ON FLOATING-POINT IMPLEMENTATION WITH XILINX LOGICORE IP

Tensor operation		**CPU**	**HW (LogiCORE IP)**			**Gain**
Operation	**MOP**	**RT (ms)**	**RT (ms)**	**MOP/s**	**GOP/W**	
Model A						
Conv	1.769	670.95	120.07	14.73	0.21	**5.59**
Conv	37.748	12,722.13	1,328.08	28.42	0.40	**9.58**
Conv	18.874	6,094.85	636.53	29.65	0.42	**9.58**
Conv	18.874	5,564.79	569.30	33.15	0.47	**9.77**
Model B						
DConv	0.027	11.51	1.557	17.75	0.25	**7.39**
Conv	0.196	94.82	20.487	9.59	0.13	**4.62**
DConv	0.147	58.84	8.355	17.64	0.25	**7.04**
Conv	1.048	368.66	40.271	26.03	0.37	**9.15**
Conv	2.359	697.08	72.981	32.32	0.46	**9.55**

TABLE III
COMPUTE PERFORMANCE ON HYBRID CUSTOM FLOATING-POINT IMPLEMENTATION

Tensor operation		**CPU**	**HW (Hybrid floating-point)**			**Gain**
Operation	**MOP**	**RT (ms)**	**RT (ms)**	**MOP/s**	**GOP/W**	
Model A						
Conv	1.769	670.95	68.50	25.83	0.38	**9.8**
Conv	37.748	12,722.13	307.83	122.63	1.80	**41.33**
Conv	18.874	6,094.85	147.97	127.55	1.87	**41.19**
Conv	18.874	5,564.79	124.03	152.17	2.23	**44.87**
Model B						
DConv	0.027	11.51	1.41	19.63	0.29	**8.17**
Conv	0.196	94.82	20.34	9.43	0.14	**4.66**
DConv	0.147	58.84	6.58	22.41	0.33	**8.94**
Conv	1.048	368.66	12.75	82.23	1.21	**28.91**
Conv	2.359	697.08	17.14	137.68	2.02	**40.68**

2) Floating-point:

3) DepthwiseConv2D operator:

B. Accuracy performance

C. Resource utilization and power dissipation

TABLE IV
RESOURCE UTILIZATION AND POWER DISSIPATION OF THE PROPOSED ACCELERATOR.

HW		**Post-implementation resource utilization**				**Power (W)**
	Operators	**LUT**	**FF**	**DSP**	**BRAM 18K**	
	Conv	5,677	4,238	78	70	0.136
a)	DConv	7,232	5,565	106	70	0.171
	Conv + DConv	12,684	8,015	160	70	0.248
	Conv	4,670	3,909	59	266	0.070
b)	DConv	6,263	5,264	82	266	0.075
	Conv + DConv	10,871	7,726	123	266	0.119
	Conv	6,787	4,349	56	74	0.066
c)	DConv	8,209	5,592	79	74	0.072
	Conv + DConv	14,590	8,494	117	74	0.102

a) 8-bit fixed-point.
b) Floating-point with Xilinx LogiCORE IP.
c) Hybrid custom floating-point/logarithmic.

VI. CONCLUSIONS

ACKNOWLEDGMENTS

This work is funded by the *Consejo Nacional de Ciencia y Tecnologia – CONACYT* (the Mexican National Council for Science and Technology).

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] 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.

[6] "Coral. dev board datasheet.," https://coral.ai/docs/dev-board/datasheet/., accessed September 15, 2021.

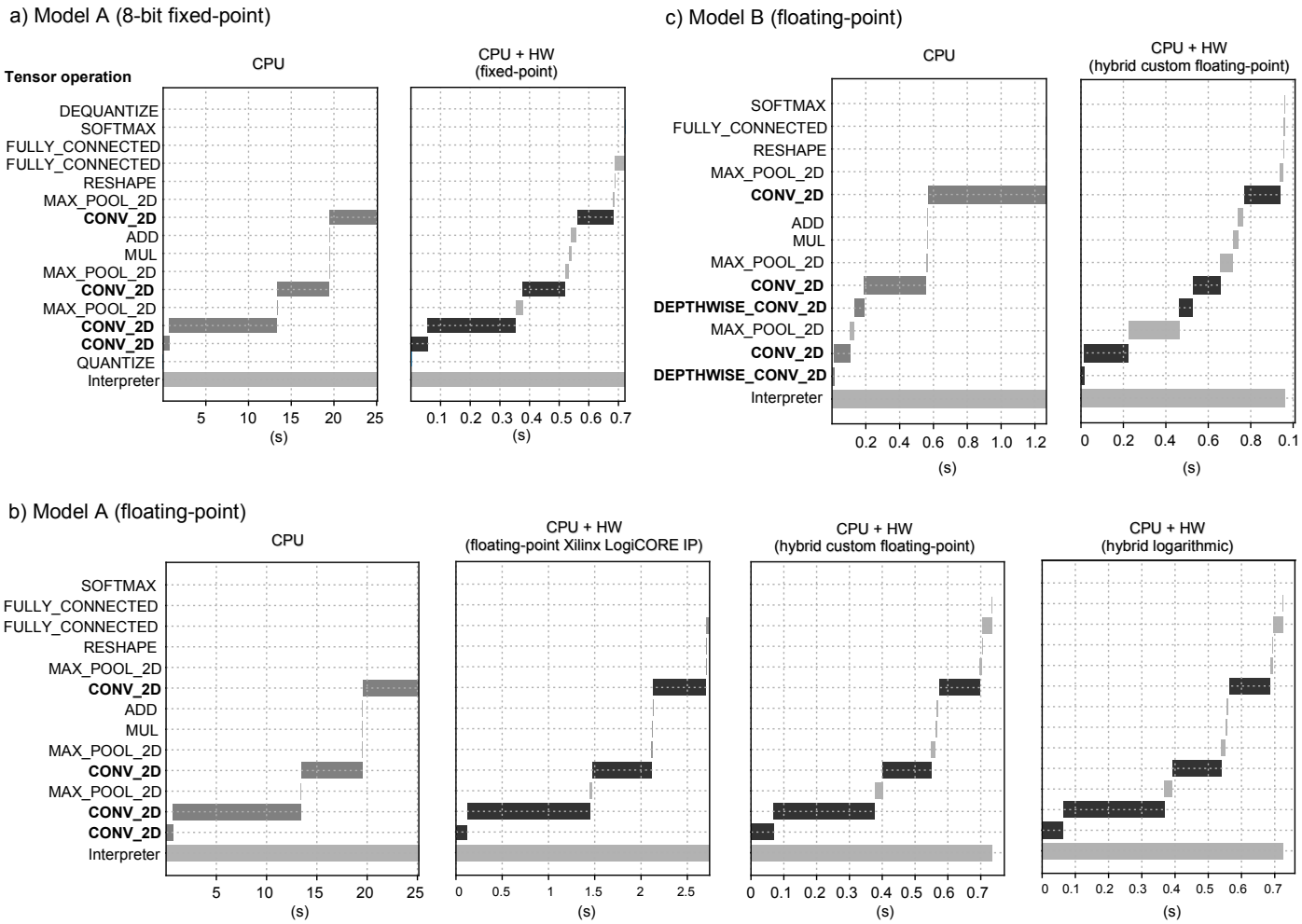

Fig. 6. Accuracy performance of depthwise separable CNN architecture with custom floating-point approximation.

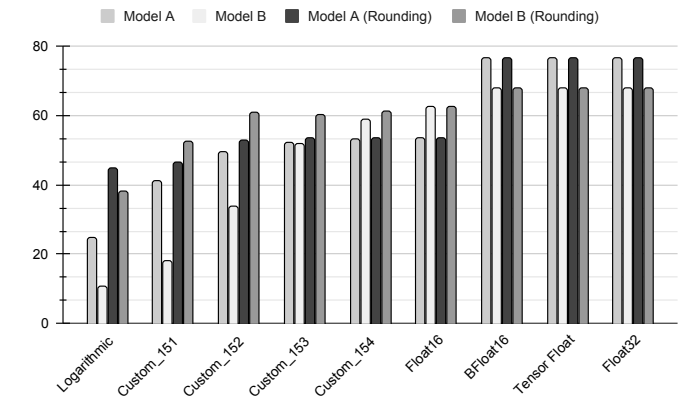

Fig. 7. Accuracy performance of depthwise separable CNN architecture with custom floating-point approximation.

[7] "Coral. tensorflow models on the edge tpu.," https://coral.ai/docs/edgetpu/models-intro/., accessed September 15, 2021.

[8] 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.

[9] 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.

[10] "Tensorflow lite for microcontrollers. understand the c++ library.," https://www.tensorflow.org/lite/microcontrollers/library., accessed September 15, 2021.

[11] U. Xilinx, "Zynq-7000 all programmable soc: Technical reference manual," 2015.

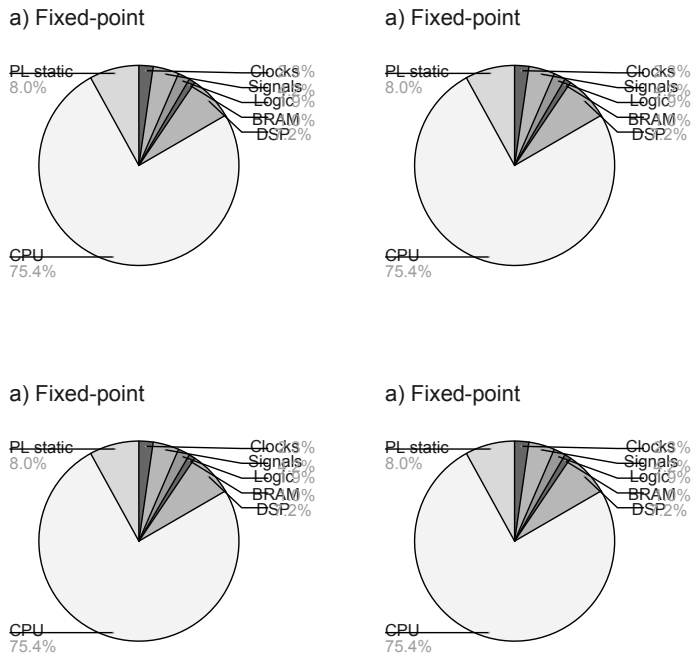

Fig. 8. Accuracy performance of depthwise separable CNN architecture with custom floating-point approximation.