IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 26 December 2022, accepted 7 January 2023, date of publication 16 January 2023, date of current version 19 January 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3236974

RESEARCH ARTICLE

An Exploration of State-of-the-Art Automation Frameworks for FPGA-Based DNN Acceleration

FUMIO HAMANAKA, TAKASHI ODAN, KENJI KISE, (Member, IEEE), AND THIEM VAN CHU, (Member, IEEE)
Tokyo Institute of Technology, Tokyo 152-8550, Japan

Corresponding author: Fumio Hamanaka (hamanaka@arch.cs.titech.ac.jp)

This work was supported by the New Energy and Industrial Technology Development Organization (NEDO) under Project JPNP16007.

ABSTRACT FPGA-based acceleration is considered a promising approach to improve the performance and power efficiency of Deep Neural Network (DNN) inference tasks. However, mapping a DNN onto an FPGA is not trivial. To make this easier, various automation frameworks have been proposed. Among them, FINN and Vitis AI, both developed by Xilinx, are two key players. They represent two different philosophies in designing FPGA-based DNN accelerators: dataflow-style and overlay-style architectures. Dataflow architectures are generally expected to provide better performance and power efficiency but have a major drawback in that they scale very poorly to the size of the target DNN. Advanced frameworks like FINN alleviate this drawback by transforming the target DNN into operations that can be time-multiplexed on fewer hardware resources. This approach, however, is challenging because of the difficulty in transforming the target DNN and raises a question as to whether the generated dataflow architectures retain a significant advantage over overlay architectures in terms of performance and power efficiency. This paper aims to clarify it by conducting an in-depth exploration of FINN and Vitis AI. For this purpose, we extend the FINN's development flow to be able to use the same target hardware and DNN model to evaluate each framework. We demonstrate the effectiveness of the FPGA-based acceleration by providing a comparison with two reference platforms: an NVIDIA Jetson Nano Developer Kit with a similar power budget to our target FPGA hardware, and a high-performance desktop computer with an Intel Core i7-11700K CPU. The results show that despite the use of the time-multiplexing approach, the FINN-based accelerator can still outperform the Vitis-AI-based accelerator by a significant margin, 8.4x in terms of latency, 3.0x in terms of throughput, and 3.3x in terms of power efficiency. The outcome of the comparison with the NVIDIA Jetson Nano Developer Kit and the desktop computer is also overwhelmingly favorable to the FINN-based accelerator. This indicates that, even in the case of using automation frameworks, DNN accelerators on FPGAs can still yield significant performance and power efficiency gains compared with GPUs and CPUs.

INDEX TERMS Deep neural network, FPGA-based acceleration, automation framework.

I. INTRODUCTION

Deep neural networks (DNNs) are rapidly improving and making a significant impact in many fields such as language translations, healthcare, and self driving cars. In this paper, our area of interest is embedded systems, where various DNN inference applications require low latency and high power efficiency.

Prior works have shown that FPGAs offer attractive performance and power efficiency for DNN inference applications [1], [2], [3], [4]. For mapping a DNN model to an FPGA, the quantization of weights and activations is the key optimization. It reduces not only the computation complexity but also the memory footprint requirement. Some works [5], [6] have shown that quantized neural networks (QNNs) can achieve comparable inference accuracy and several times better performance than the original ones with double precision floating point arithmetic.

The associate editor coordinating the review of this manuscript and approving it for publication was Prakasam Periasamy.

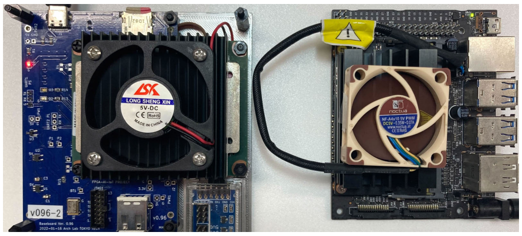


FIGURE 1. Left: a Kria-K26-SoM-based custom FPGA board that we use for evaluating FINN and Vitis AI. Right: an NVIDIA Jetson Nano Developer Kit that we use as a reference platform for evaluating the results of FINN and Vitis AI.

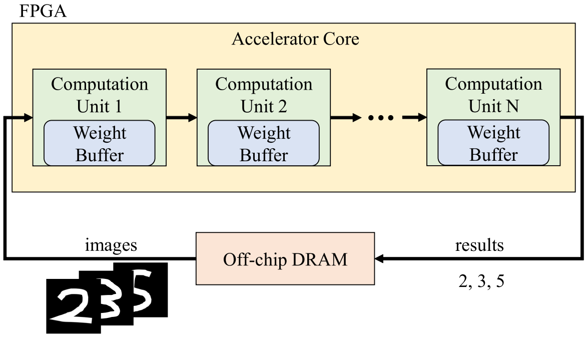


FIGURE 2. A typical dataflow architecture for processing an N-layer DNN. FINN employs this model.

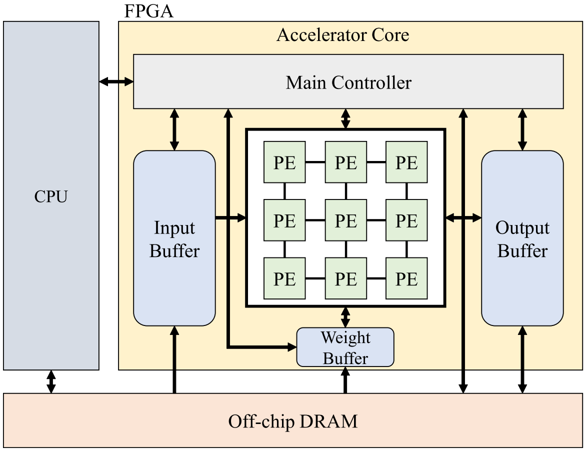


FIGURE 3. A typical overlay architecture for DNN accelerators. Vitis AI employs this model.

To make the mapping of a QNN onto an FPGA easier, a number of automation frameworks such as FINN [7], Vitis AI [8], hls4ml [9], and VTA [10] have been proposed. These frameworks can be categorized into two groups based on the types of accelerator architectures that they generate: dataflow-style or overlay-style (Figure 2 and Figure 3). With fixed-function pipelines, dataflow architectures are expected to provide better performance and power efficiency. However, realizing them is not easy. Our experiments show that a straightforward mapping of even an extremely small DNN of several tens of thousands of parameters to a dataflow architecture can result in a large circuit that a typical FPGA for embedded systems cannot accommodate. To support larger and more useful DNNs, some frameworks such as FINN propose transforming the target DNNs into common operations like matrix-vector multiplications and time-multiplexing these operations onto fewer hardware resources. This approach, however, raises a question: with the time-multiplexed operations, is the generated dataflow architecture faster and more efficient than highly optimized overlay architectures provided by frameworks like Vitis AI? If the answer is yes, then how large is the gap? Given the fact that the transformation task for generating dataflow architectures is highly dependent on the target DNNs and thus requires extensive trials and experiments, a detailed and quantitative analysis would help users to determine which type of framework is suitable for their needs.

In this paper, we aim to answer the above questions by conducting an in-depth exploration of the performance and power efficiency characteristics of DNN accelerators generated by FINN and Vitis AI, two prominent representatives of the two automation framework groups. There have been some efforts to evaluate FINN and Vitis AI individually, with the comparison targets being the executions on CPUs or GPUs [11], [12], [13]. Investigating the gap between these two frameworks, however, is still an open question. No conclusion can be drawn because different DNN models, FPGA devices, and evaluation methodologies are used in different studies. The closest to our work is the study of Plagwitz et al. [14], where both FINN and Vitis AI are present in their survey of different automation frameworks for DNN acceleration on FPGAs. The analysis in [14], however, is mainly qualitative. There are only limited quantitative data regarding the inference latency of an extremely simple neural network model with one convolution layer and one fully connected layer on a PYNQ-Z1 board for FINN and a ResNet-50 model on a ZCU106 board for Vitis AI. Different architectural configurations are not investigated and no throughput and power efficiency data are reported. Because of these limitations, it is difficult to draw any conclusions about the gap of performance and power efficiency between using FINN and using Vitis AI.

A major obstacle for comparing FINN and Vitis AI is that the FPGA boards and DNN models supported by them are different. Different from Vitis AI, a commercial product, FINN and other dataflow-style frameworks are still research projects and have some usability limitations. Like in [14], we find that it is extremely difficult to use FINN for FPGA boards and DNN models that are different from those provided by the authors.[1] Therefore, in this work, we extend the development flow of FINN to be able to provide a comparison with Vitis AI using the same target hardware and the same DNN model. This helps us to explore the gap of performance and power efficiency between the DNN accelerators generated by FINN and Vitis AI.

[1] [14] ends up evaluating a simple 2-layer neural network on a PYNQ-Z1 board (an FPGA board supported by FINN) as mentioned earlier.

By extending the development flow, we have successfully ported FINN to our target FPGA hardware, a custom FPGA board featuring a Xilinx Kria K26 System on Module (SoM) (the left hand side of Figure 1). To interpret the significance of the results of FINN and Vitis AI, we compare them to the results obtained when using two reference platforms: an NVIDIA Jetson Nano Developer Kit (the right hand side of Figure 1) with a similar power budget to our FPGA board and a desktop computer with an Intel Core i7-11700K CPU.

The main contributions of this paper are summarized as follows:

- We extend the FINN's development flow to support a wide variety of FPGA boards and DNN models that are not supported by the original FINN. This extended version is named **FINN-S**.
- We conduct an exploration of the performance and power efficiency of the accelerators generated by FINN-S and Vitis AI using the same target FPGA board, the same DNN model (a ResNet [15] model), and a wide range of configurations. We compare the results with those obtained when using an NVIDIA Jetson Nano Developer Kit and a high-performance desktop computer.
- We clarify the gap of performance and power efficiency between the DNN accelerators generated by FINN-S and Vitis AI. We show that the former outperforms the latter by a significant margin, 8.4x in terms of latency, 3.0x in terms of throughput, and 3.3x in terms of power efficiency. We also provide our insights into these results based on the characteristics of the accelerators.
- We show that with regard to the comparison with the NVIDIA Jetson Nano Developer Kit and the desktop computer, though the NVIDIA Jetson Nano Developer Kit outperforms the accelerator generated by Vitis AI, the outcome is overwhelmingly favorable to the accelerator generated by FINN-S. These mixed results show that FPGA-based DNN accelerators generated by automation frameworks can still outperform both CPUs and GPUs but the use of an appropriate framework is necessary.

Throughout the paper, we use the term FINN-based accelerator to refer to an accelerator generated by FINN or FINN-S and Vitis-AI-based accelerator to refer to an accelerator generated by Vitis AI.

II. BACKGROUND AND RELATED WORKS

A. DNN INFERENCE ACCELERATORS ON FPGAs

FPGA-based DNN inference accelerators can be classified into two types of architectures: dataflow-style and overlay-style.

Figure 2 shows the organization of a typical dataflow architecture for processing an N-layer DNN where each layer is processed by a computation unit. In this architecture, all weights and biases of the DNN model are stored in FPGA on-chip memories (weight buffers in Figure 2). In general, they can also be hard-wired into the circuit. Typical automation frameworks generating dataflow architectures include FINN [7], [16] and hls4ml [9].

A straightforward implementation of dataflow architectures is to spatially unroll all operations of the target DNN model on the FPGA. While this approach can give superior performance, it comes at the cost of excessive FPGA resource usage even when the DNN model is small. Advanced frameworks such as FINN deal with this problem by transforming their target DNN models into common operations that can be time-multiplexed onto fewer FPGA resources. For example, in FINN, a convolutional layer is performed by using two operations, sliding window and matrix-vector multiplication, which can be processed with different parallelism degrees depending on the available FPGA resources.

Figure 3 shows the organization of a typical overlay architecture. In this architecture, the operations for an inference task are processed by an array of processing elements (PEs). The weights and biases of the target DNN model are stored in off-chip DRAM and cached in the on-chip weight buffer during the inference process. Input and output data of the PE array are also cached in on-chip memory units, namely input and output buffers. The main controller controls the PEs and the reads/writes from/to the buffers using a series of instructions. Typical automation frameworks generating overlay architectures include Vitis AI [8] and VTA [10].

In summary, dataflow architectures offer great flexibility in customizing the processing datapaths but do not scale well as overlay architectures which utilize off-chip DRAM during the inference process. With the time-multiplexing approach, dataflow architectures can support larger DNN models. However, the strict constraint of on-chip memory capacity is still a concern. It makes it difficult to set a high parallelism degree because a large number of resources need to be allocated for storing or hard-wiring the weights and biases on-chip. The flexibility in quantization helps but only to a certain extent. Therefore, it is not as obvious as it may seem that dataflow architectures always significantly outperform overlay architectures in terms of performance and power efficiency. We aim to clarify it in this paper.

B. DNN INFERENCE ACCELERATORS GENERATED BY FINN

FINN provides three main computation units: the Matrix-Vector-Threshold Unit (MVTU), the Sliding Window Unit (SWU), and the Pooling Unit (PU) [7], [16]. It maps convolutional layers to the SWU and MVTU and fully connected layers to the MVTU.

The MVTU contains a processing element (PE) array, and each PE has some SIMD lanes. Each PE performs the same number of multiplications in parallel as the number of SIMD lanes. Users can determine the folding factors, which are the number of PEs and the number of SIMD lanes per PE, for each convolutional and fully connected layer.

The SWU generates an input feature map matrix from incoming feature maps. It gives the MVTU one column of the input feature map matrix at a time for multiplication with the convolution kernels that have been packed into the form of a matrix.

C. DNN INFERENCE ACCELERATORS GENERATED BY VITIS AI

The accelerator core for Vitis AI is called the Deep Learning Processor Unit (DPU). It consists of a high performance scheduler module, a hybrid computing array module, an instruction fetch unit module, and a global memory pool module [17]. The DPU executes the microcode of the target DNN model, which is called *xmodel*.

DPUCZDX8G [17], which is used for Vitis AI, is available in eight different levels of parallelism. The architecture names for the eight levels of parallelism are B512, B800, B1024, B1152, B1600, B2304, B3136, and B4096. The numbers in the architecture names refer to the peak numbers of operations per cycle. For example, with the B512 DPU core, 256 multiply operations and 256 accumulate operations can be performed in one cycle.

D. RELATED WORKS

There have been some attempts to study FINN and Vitis AI individually. We summarize them in this section.

Ducasse et al. [18] study FINN using a multilayer perceptron trained on two datasets, MNIST and Fashion MNIST. In the evaluation, several parallelization configurations are used. Su et al. [19] also study FINN using a multilayer perceptron and two CNN models trained on three datasets, MNIST, CIFAR-10, and ImageNet. They propose estimation models to show how the quantization affects the hardware cost and throughput. Jentzsch et al. [11] compare the performance of FINN-based accelerators with GPUs on the RadioML modulation classification task. They extend FINN to support additional parallelism. With this, they report that the FINN-based accelerator achieves >100x lower latency and >10x higher throughput compared with an NVIDIA Jetson Xavier NX Developer Kit.

Some works [12], [13] evaluate the accelerators generated by Vitis AI. Ushiroyama et al. [12] evaluate the performance and power consumption using three different CNN models trained on the CIFAR-10 dataset. This work shows that fully connected layers become a performance bottleneck due to low DPU utilization. Verucchi et al. [13] compare the performance of an FPGA-based accelerator generated by Vitis AI with CPUs and GPUs. Their evaluation results show that the GPU platforms achieve the best performance. Two reasons for the inferior performance of the FPGA-based accelerator over GPUs have been shown. The first is that the bottleneck is not the inference process performed in the DPU, but the pre-processing and post-processing performed in the processing system. Second, the processing system performs operations not supported by the DPU instead.

We are aware of two studies that compare dataflow and overlay architectures generated by automation frameworks [14], [20]. Both FINN and Vitis AI are featured in these studies. However, it is difficult to draw any firm conclusions from them because of the reasons described below. In [14], the analysis is mainly qualitative. Only limited quantitative data regarding inference latency are reported but a different DNN model and a different FPGA board are used for each automation framework. In [20], Blott et al. provide a more detailed analysis with quantitative data on the inference accuracy, latency, throughput, power consumption, and power efficiency. However, similar to in [14], different DNN models are used for different automation frameworks, making it difficult to assess the comparison results.

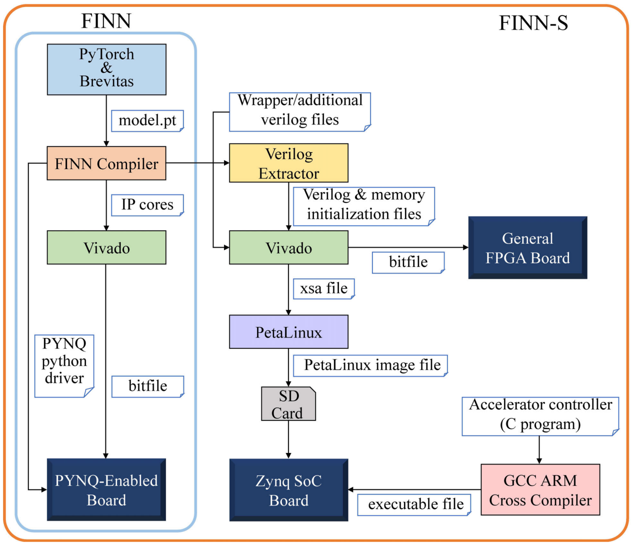


FIGURE 4. The original FINN development flow and our extension (FINN-S).

The obstacle that prevents [14] and [20] from using the same target DNN model and FPGA board for comparing FINN and Vitis AI is that it is challenging to use FINN for DNN models and FPGA boards different from those provided by its authors, which are very limited. In this work, by solving this problem, we provide a detailed and quantitative analysis of the performance and power efficiency of the FINN-based and Vitis-AI-based accelerators using the same target DNN model and FPGA board.

III. FINN-S: OUR EXTENSION OF FINN

A. A NEW DEVELOPMENT FLOW

The primary factor that motivates us to develop FINN-S is that FINN supports only several FPGA boards, and for each board provides only one or several simple or application-specific DNN models mostly without the training scripts. This makes it difficult for practitioners to draw a comparison between FINN and other automation frameworks using FPGA boards and DNN models that suit their objectives. As shown in the left part of Figure 4, the target of FINN is PYNQ-enabled boards. Specifically, FINN generates IP-based designs and provides Python drivers to run them on PYNQ. While this approach helps to greatly reduce the effort

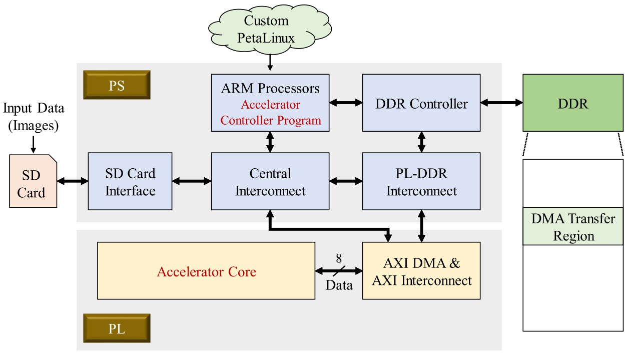


FIGURE 5. **The block diagram of our FINN-S-based accelerator system on a Zynq SoC FPGA board.**

needed when using PYNQ-enabled boards, extending it to support other platforms is not trivial. A possible solution is to exclude the PYNQ-related parts and extract only the DNN IP core. However, integrating this IP core into designs for other FPGA boards is difficult due to three problems. First, the IP core is device-specific in the sense that it is necessary to recreate the IP for each target device, and FINN allows only a limited number of Zynq SoCs. Second, even though the target device is supported, the IP core is highly dependent on the original FINN development environment where it was generated. Third, there is no framework for operating the IP core on general FPGA boards. The FINN-S extension allows us to generate Verilog code once and use it for any device.

The right part of Figure 4 shows our solution for the above problems. There are three steps to extend FINN into FINN-S: (1) extract the Verilog code of the IPs generated by FINN, (2) solve the problem that this Verilog code contains the file paths (e.g., of memory initialization files) that are valid only inside the FINN development environment (a docker container), and (3) add AXI-Stream interface, accelerator wrapper code, and controller program in the processing system (in the case of using Zynq SoCs).

FINN-S supports both general and Zynq SoC FPGA boards. Below we will focus on the latter since our target board in this study is Kria K26 SoM, which is based on a Zynq SoC.

In FINN-S, we follow the standard Vivado design flow to create a custom PetaLinux image file that contains the DNN accelerator core. We develop a controller program to communicate with the accelerator core running on the programmable logic (PL) side from the processing system (PS) side. We do not use Vitis but instead use a cross compiler to compile the program and send the generated executable file to the board to run on Linux.

Figure 5 shows the block diagram of our FINN-S-based accelerator system on a Zynq SoC FPGA board. The data exchange between the accelerator core on the PL and the controller program on the PS is via an AXI-Stream bus using DMA. We create a dedicated region on the DDR memory for this DMA transfer.

B. DNN MODEL TRAINING

An indispensable part of creating DNN accelerators is training DNN models. In FINN, how a DNN is trained not only affects the inference accuracy, performance, and efficiency of the resulting accelerator but also determines whether and how it can be transformed into hardware.

A major problem of FINN is the difficulty of DNN model training and transformation. When the transformation of a DNN model into hardware fails, it is hard to determine whether this is due to the training code, the lack of some essential transformation steps, or some operations in the DNN that have not been supported yet. FINN does provide some pre-trained DNN models and their transformation scripts. However, the training code is available in only several cases. For example, though there is a ResNet-50 sample design targetting an Alveo U250 board, no training code is provided. Therefore, even though our target DNN model in this work is a variation of ResNet, we have to develop almost everything from scratch. Even in the cases where the training code is made available, we find that modifying it to create a custom DNN model that can also be transformed into hardware is not trivial.

With FINN-S, we have developed a list of hardware-transformable operations, how to integrate them into a complete DNN, and what transformation steps are required. The currently supported operations include convolutions (normal/pointwise/depthwise), fully connected (linear), batch normalization, activation function ReLU, max/average pooling, dropout, and skip connection. The training code is written using PyTorch and Brevitas [21], a library for quantization-aware training.

IV. EVALUATION

A. TARGET DNN MODEL, TRAINING, AND EVALUATION METRICS

Our target DNN model is ResNet-8 (Figure 6) from the MLPerf Tiny benchmark [22]. We choose this model because of two main reasons. First, it features the most important and popular operations of modern DNN architectures: convolution, batch normalization, activation function ReLU, skip connection, average pooling, and fully connected. Second, it is a reasonably good model, achieving a TOP1 accuracy of over 85% on CIFAR-10 [23] even with coarse quantization, while being small enough for us to be able to investigate a wide range of different levels of parallelism of the accelerators generated by the automation frameworks, especially the FINN-based one. Since our area of interest is embedded systems, using a very large FPGA is not appropriate. However, when the target FPGA is not large, as mentioned in Section II-A, it is difficult for FINN to support a large DNN model due to the fact that all the weights and biases of the model are assumed to be hard-wired or stored entirely on-chip, and this requires a large number of FPGA resources. Extending FINN to provide the option of using off-chip DRAM for storing all or a part of the weights

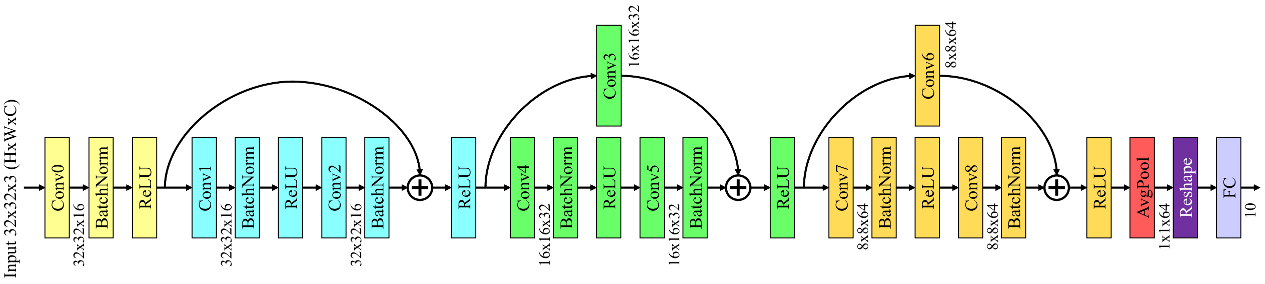


FIGURE 6. The structure of ResNet-8, our target DNN model.

and biases, thereby making it possible to implement large DNN models on small FPGAs, is an interesting problem but beyond the scope of this work.

We train the ResNet-8 models for evaluating the automation frameworks and the reference platforms using PyTorch. In all cases, the training batch size and the number of training epochs are set to 32 and 500, respectively. We use the Stochastic Gradient Descent (SGD) optimizer with the weight decay of 10^{-4} and the momentum of 0.9. The initial learning rate is 0.075, and we use cosine annealing as the learning rate scheduler.

Our evaluation metrics include the TOP1 inference accuracy, latency, throughput, power consumption, and power efficiency. The inference accuracy is measured on the test set of the CIFAR-10 dataset. The latency is the time from the start to the end of inference for a single image, as the MLPerf Tiny benchmark defines. Thus, the latency is measured with a batch size of 1, an input stream size of 1, and the number of threads of 1. On the other hand, for throughput evaluation, we vary these parameters and observe the convergence point. We use FPS (frames per second) as the measure of throughput. FPS can also be interpreted as the number of inferences per second. The power efficiency is the throughput per power consumption.

B. REFERENCE PLATFORM 1: NVIDIA JETSON NANO DEVELOPER KIT

1) EXPERIMENTAL SETUP

We use an NVIDIA Jetson Nano Developer Kit equipped with a quad-core Arm Cortex-A57 processor, a 128-core Maxwell GPU running at 921 MHz, and 4 GB memory. As the optimization method of the inference task, we convert our FP32 trained PyTorch model to the TensorRT model with FP16 precision using torch2trt [24]. For throughput evaluation, the batch size is varied from 1 to 256.

2) RESULT

We achieve a TOP1 inference accuracy of 89.58%. The other evaluation results are summarized in Table 1. The first column is the batch size, which is varied from 1 to 256. The second column is the inference latency. The third column is the throughput obtained by dividing the number of inferences by the elapsed time. The fourth column is the power consumption measured by using the *tegrastats* command. We sample 120 points for around 30 seconds and take the average of them. The last column is the power efficiency measured in FPS/W.

TABLE 1. Performance and power efficiency results on reference platform 1: NVIDIA Jetson Nano Developer Kit.

Batch size	Latency (ms)	Throughput (FPS)	Power (W)	Efficiency (FPS/W)
1	0.940	1,135	3.83	297
2	—	2,381	4.36	546
4	—	4,474	5.58	802
8	—	6,278	6.22	1,009
16	—	6,704	6.31	1,063
32	—	7,002	6.39	1,095
64	—	**7,091**	**6.43**	**1,103**
128	—	6,921	6.43	1,077
256	—	5,836	6.47	903

We can see that the inference latency is 0.940 ms. The throughput increases with increasing the batch size until 64. The peak throughput of 7,091 FPS is obtained when the batch size is set to 64. At this configuration, the average power consumption is 6.43 W, and thus the power efficiency is 1,103 FPS/W.

C. REFERENCE PLATFORM 2: HIGH-PERFORMANCE DESKTOP COMPUTER

1) EXPERIMENTAL SETUP

We use a computer with an 8-core Intel Core i7-11700K CPU and 128 GB memory. We conduct the evaluation using an FP32 trained model since the CPU does not support lower precisions. For throughput evaluation, we vary the batch size from 1 to 256.

2) RESULT

We achieve a TOP1 inference accuracy of 89.59%. The other evaluation results are summarized in Table 2, which has a structure similar to Table 1. The power consumption of the whole computer is measured using a TAP-TST8N power meter during an approximately 10-second execution of the inference program.

We can see that the inference latency is 0.428 ms. The throughput increases with increasing the batch size until 64.

TABLE 2. Performance and power efficiency results on reference platform 2: high-performance desktop computer.

Batch size	Latency (ms)	Throughput (FPS)	Power (W)	Efficiency (FPS/W)
1	0.428	2,302	170	13.5
2	—	2,610	169	15.4
4	—	4,458	170	26.2
8	—	6,984	169	41.3
16	—	9,583	170	56.4
32	—	10,189	171	59.6
64	—	**11,119**	**172**	**64.6**
128	—	8,004	173	46.3
256	—	6,395	174	36.8

TABLE 3. The folding configurations to explore the performance of our FINN-based accelerator.

Config	Folding factors (number of PEs, number of SIMD lanes per PE)								
	cnv0	cnv1	cnv2	cnv3	cnv4	cnv5	cnv6	cnv7	cnv8
C1	1, 1	1, 1	1, 1	1, 1	1, 1	1, 1	1,1	1, 1	1, 1
C2	1, 1	1, 2	1, 2	1, 1	1, 1	1, 2	1,1	1, 1	1, 2
C3	1, 1	1, 4	1, 4	1, 1	1, 2	1, 4	1,1	1, 2	1, 4
C4	1, 3	1, 8	1, 8	1, 1	1, 4	1, 8	1,1	1, 4	1, 8
C5	1, 9	1, 48	1, 48	1, 4	1, 16	1, 32	1, 4	1, 16	1, 32
C6	1, 27	1, 144	1, 144	1, 8	2, 48	2, 96	1, 8	1, 96	1, 192

TABLE 4. The hardware utilization and performance results of six configurations of our FINN-based accelerator. The input stream size is set to 1 when measuring the inference latency.

Config	LUTs	FFs	36Kb BRAMs	Latency (ms)	Throughput (FPS)
C1	34,041 (29.1)	46,172 (19.7)	22.5 (15.6)	20.12	91.2
C2	33,589 (28.7)	45,935 (19.6)	16.5 (11.5)	10.42	182
C3	33,366 (28.5)	46,504 (19.9)	15.0 (10.4)	5.31	365
C4	33,903 (28.9)	47,125 (20.1)	16.0 (11.1)	2.63	730
C5	44,343 (37.9)	54,162 (23.1)	15.5 (10.8)	0.64	3,013
C6	81,391 (69.5)	87,642 (37.4)	28.5 (19.8)	0.15	13,533

The peak throughput of 11,119 FPS is obtained when the batch size is set to 64. At this configuration, the average power consumption is 172 W, and thus the power efficiency is 64.6 FPS/W.

D. FINN-BASED ACCELERATOR

1) EXPERIMENTAL SETUP

We train a quantized ResNet-8 model using Brevitas [21], a library for quantization-aware training on PyTorch. For simplicity, the inputs, weights, biases, and activations are all quantized using a common 4-bit fixed-point format. However, note that a mixed-precision quantization scheme where the inputs, weights, biases, and activations are quantized differently (each layer can also have a different quantization scheme) is also possible. Our model is trained to satisfy the condition of the MLPerf Tiny benchmark: the TOP1 inference accuracy of the trained model should be more than 85% on the CIFAR-10 dataset.

The trained PyTorch model is first converted to the ONNX format and then input into the FINN compiler where it is translated to the corresponding HLS layers in C++ with the folding factors (number of PEs and number of SIMD lanes per PE) for processing each layer set as in Table 3. Then, the FINN compiler converts the HLS layers to Verilog code. We use the Verilog code to create a Vivado project and generate an XSA file. A PetaLinux image is created using the XSA file.

The parameters that we vary while evaluating FINN-S are the number of PEs, the number of SIMD lanes per PE, and the stream size which is the number of images continuously input into the accelerator. We evaluate six configurations (from C1 to C6) of the number of PEs and the number of SIMD lanes per PE for the nine convolutional layers of ResNet-8 as shown in Table 3. We do not include configurations with different numbers of PEs and numbers of SIMD lanes per PE for the FC (fully connected) layer because we find that varying the number of PEs and the number of SIMD lanes per PE for the FC layer has only a minor impact on the overall performance. This is because the FC layer is small and the number of operations required to execute it accounts for only 0.005% of the total number of operations. For DNNs with a significantly large FC layer, it is necessary to take into account the number of PEs and the number of SIMD lanes for the FC layer while designing the configurations. In Table 3, C1 is the configuration of the minimal hardware where all folding factors are set to 1. C6 is the configuration of the maximum hardware where large folding factors are carefully chosen considering the target FPGA hardware budget. C2 to C5 are set as their intermediate configurations. The stream size is varied from 1 to 256.

The version of the FINN compiler used is v0.7. We use Vivado 2021.2 to create the accelerator core of the PL part of the system, and PetaLinux 2021.2 to create the PetaLinux image for the PS part. The target device is Xilinx Kria K26 SoM with 117,120 LUTs, 234,240 FFs, and 144 BRAMs. The PS part features a quad-core Arm Cortex-A53 processor.

2) RESULT

Table 4 shows the hardware utilization, the latency in milliseconds, and the throughput in FPS for the six configurations running at 225 MHz, which is the maximum clock frequency of the accelerator of the largest configuration C6. Other configurations may operate at slightly higher clock frequencies but it does not fundamentally change our comparison result here. We use Verilog simulation to obtain the number of elapsed cycles and calculate the values in Table 4.

In Table 4, the second, third, and fourth columns show the number of occupied LUTs, FFs, and 36Kb BRAMs with the numbers in the brackets being the utilization ratios. We can see that the smallest configuration C1 consumes an almost similar number of hardware resources compared to the larger configurations C2, C3, and C4. This indicates that too small configurations may lead to inefficiency in utilizing the FPGA resources. In terms of performance, we can see that the largest configuration, C6 is the best. It provides over 4x better latency and throughput results compared with configuration C5. Therefore, we use C6 in the following evaluations.

TABLE 5. The performance and power efficiency results of our FINN-based accelerator which is generated using configuration C6 and operates at 225 MHz.

Input stream size	Latency (ms)	Throughput (FPS)	Power (W)	Efficiency (FPS/W)
1	0.154	6,499	5.29	1,229
2	—	8,775	5.46	1,609
4	—	10,654	5.63	1,894
8	—	11,915	5.75	2,074
16	—	12,672	5.82	2,179
32	—	13,089	5.85	2,236
64	—	13,306	5.87	2,268
128	—	13,415	5.88	2,280
256	—	**13,475**	**5.89**	**2,287**

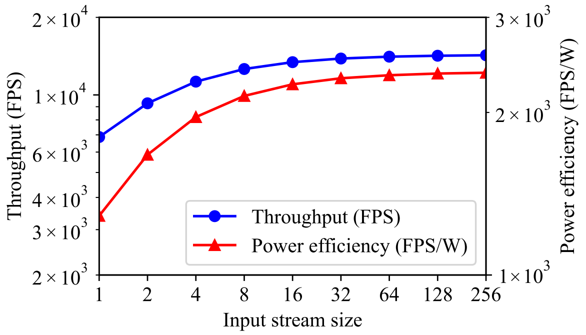


FIGURE 7. The throughput and power efficiency of our FINN-based accelerator with configuration C6.

Table 5 shows the evaluation results of our FINN-based accelerator using C6. The maximum operating frequency of the accelerator is 225 MHz, and the inference latency is calculated to be 0.154 ms. The inference accuracy measured on the FPGA is 85.88%. We vary the input stream size from 1 to 256 images. We measure the power consumption using a CT-3 power meter. We sample 1,000 points for around 10 seconds and take the average of them.

Figure 7 shows the throughput and power efficiency of our FINN-based accelerator with configuration C6. We can see that both throughput and power efficiency improve with increasing the stream size from 1 to 64 but almost plateau after that. The peak throughput and power efficiency of 13,475 FPS and 2,287 FPS/W are achieved when the input stream size is 256.

The left part of Figure 8 shows the place and route result of our FINN-based accelerator. The slices occupied by the modules that implement the nine convolution layers (conv0–8) and the fully connected layer (FC) are highlighted using the same colors as in Figure 6. The remaining occupied slices are highlighted in red.

E. VITIS-AI-BASED ACCELERATOR

1) EXPERIMETAL SETUP

Table 6 shows the configuration of the DPU used in our evaluation. Based on the number of available hardware resources and the performance trend, we vary the number of DPU cores from 1 to 4 and explore all eight different DPU architectures (B512, B800, B1024, B1152, B1600, B2304, B3136, and B4096) only for the 1-core configuration. For the other configurations, only the B512 architecture is used. All of the remaining parameters are set to default values except that the Ultra-RAM (URAM) Use per DPU is set to 64, the number of URAMs available on a Kria K26 SoM, so that there is enough on-chip memory.

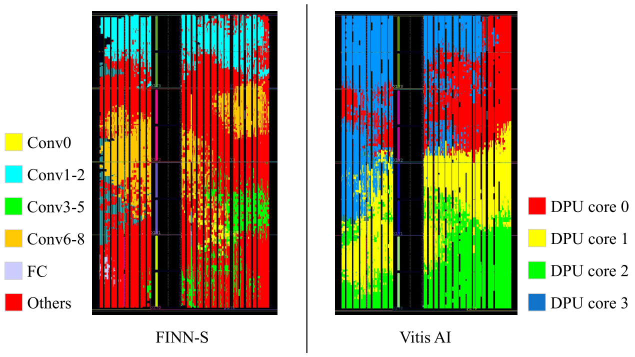


FIGURE 8. The place and route results of our FINN-based and Vitis-AI-based accelerators.

TABLE 6. Configuration parameters of the DPU used in our evaluation of Vitis AI.

Parameter	Value
Number of DPU Cores	1–4
DPU Architecture	1-core → all (B512–B4096) 2–4-core → B512
RAM Usage	Low
Channel Augmentation	Enabled
DepthWiseConv	Enabled
ElementWise Multiply	Disabled
Average Pool	Enabled
ReLU Type	ReLU + LeakyReLU + ReLU6
Number of SFM cores	0
DSP48 Usage	High
URAM Use per DPU	64

We follow the standard Vivado flow to create a PetaLinux image with the DPU. The versions of Vivado and PetaLinux and the target device are the same as in the evaluation of FINN-S.

For the evaluation of the DPU, we first train an FP32 ResNet-8 model using PyTorch. We next use Vitis AI (version 2.0) to quantize and compile this model into an *xmodel* file containing instructions for execution on the DPU. We set the quantization bitwidth to 8 for all weights, biases, and activations, which is the only configuration available for Kria K26 SoM devices. The number of running threads is varied from 1 to 10.

2) RESULT

The TOP1 inference accuracy measured on the FPGA is 89.23%, which is almost the same as that of the original FP32 model. The results for the other evaluation metrics are described below.

First, we set both the number of DPU cores and the number of threads to 1 and observe the inference latency

TABLE 7. FPGA resource usage, performance, and power efficiency results of our Vitis-AI-based accelerator obtained when varying the DPU architecture from B512 to B4096 and fixing both the number of DPU cores and the number of threads to 1. All designs operate at the same clock frequency of 300 MHz.

DPU architecture	LUTs		FFs		36Kb BRAMs		URAMs		DSPs		Latency (ms)	Throughput (FPS)	Power (W)	Efficiency (FPS/W)
	#	%	#	%	#	%	#	%	#	%				
B512	25,551	21.8	33,725	14.4	2	1.4	18	28.1	110	8.8	1.114	1,114	4.98	224
B800	28,366	24.2	40,557	17.3	2	1.4	40	62.5	157	12.6	1.120	1,029	5.19	198
B1024	31,525	26.9	47,849	20.4	2	1.4	26	40.6	218	17.5	1.041	1,153	5.12	225
B1152	30,901	26.4	47,007	20.1	2	1.4	44	68.8	212	17.0	1.115	1,059	5.28	201
B1600	35,454	30.3	58,350	24.9	2	1.4	56	87.5	312	25.0	1.039	1,235	5.46	226
B2304	39,017	33.3	68,994	29.5	2	1.4	60	93.8	422	33.8	1.001	1,157	5.45	212
B3136	43,517	37.2	79,738	34.0	2	1.4	64	100.0	548	43.9	1.008	1,119	5.55	202
B4096	49,418	42.2	98,734	42.2	17	11.8	64	100.0	690	55.3	0.955	1,024	5.65	181

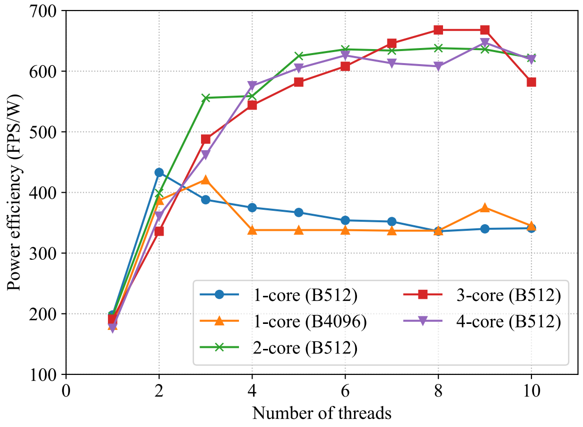


FIGURE 9. The power efficiency of our Vitis-AI-based accelerator with different numbers of DPU cores and threads.

TABLE 8. Performance and power efficiency results of our Vitis-AI-based accelerator with different numbers of B512 DPU cores and clock frequencies.

# of cores	Frequency (MHz)	# of threads	Latency (ms)	# of threads	Throughput (FPS)	Power (W)	Efficiency (FPS/W)
3	200	1	1.317	9	4,180	6.12	683
3	300	1	1.140	9	4,256	6.37	668
4	200	1	**1.293**	9	**4,458**	**6.42**	**695**
4	270	1	1.169	9	4,259	6.58	647

and the trends of the other metrics obtained when varying the DPU architecture from B512 to B4096. Table 7 shows the results. In every case, the design operates at a clock frequency of 300 MHz. As same as in the evaluation of FINN-S, we measure the power consumption of the FPGA board using a CT-3 power meter. We sample 1,000 points for around 10 seconds and take the average of them. From Table 7, we can see that increasing the parallelism of the DPU architecture does not improve the latency, throughput, and power efficiency.

Next, we investigate how the throughput and power efficiency change with increasing the number of threads and the number of DPU cores. Based on the observation that they do not change much with the DPU architecture when both the number of threads and the number of DPU cores are fixed to 1, we focus on only B512 and B4096, the smallest and largest ones. In the first experiment, we increase the number of threads while keeping the number of DPU cores fixed to 1. The clock frequency is also kept at 300 MHz. As shown in Figure 9, the peak power efficiency is achieved when the number of threads is increased to 2 for the B512 architecture and 3 for the B4096 architecture. However, no improvement is obtained when the number of threads is increased further. Since there is no significant difference in the results of the two architectures, in the experiment of varying the number of DPU cores, we focus on only the B512 architecture. We are able to keep the design operating at 300 MHz when the number of DPU cores is increased to 3. However, the frequency drops to 270 MHz when the number of DPU cores is 4. Figure 9 shows that the three multicore configurations scale better with the number of threads than the single core configuration. The 2-core configuration achieves a peak power efficiency that is 1.5x better than that of the single core configuration when the number of threads is 8. The 3- and 4-core configurations exhibit almost similar trends, achieving their peak power efficiencies when the number of threads is 9.

Finally, we explore how the results change when reducing the clock frequency. We evaluate the 3- and 4-core (B512) configurations running at 200 MHz with varying the number of threads. Table 8 shows the evaluation results. Here, latency is measured when the number of threads is 1, while throughput and power efficiency are measured at their peak when the number of threads is 9. For both configurations, reducing the clock frequency to 200 MHz leads to lower power consumption while retaining the throughput. As a result, the power efficiency is improved.

In summary, with the TOP1 inference accuracy of 89.23%, Vitis AI achieves a peak throughput and power efficiency of 4,458 FPS and 695 FPS/W, respectively, when the DPU architecture is B512, the number of DPU cores is 4, the clock frequency is 200 MHz, and the number of threads is 9. The latency, which is measured when the number of threads is set to 1, is 1.293 ms.

The right side of Figure 8 shows the place and route result of our Vitis-AI-based accelerator with the above configuration. We use four colors, namely red, yellow, green, and blue, to highlight the occupied slices. Each color indicates a DPU core.

F. COMPARISON OF DNN ACCELERATORS

In this section, using the results presented in the previous sections, we explore the gap between the FINN-based

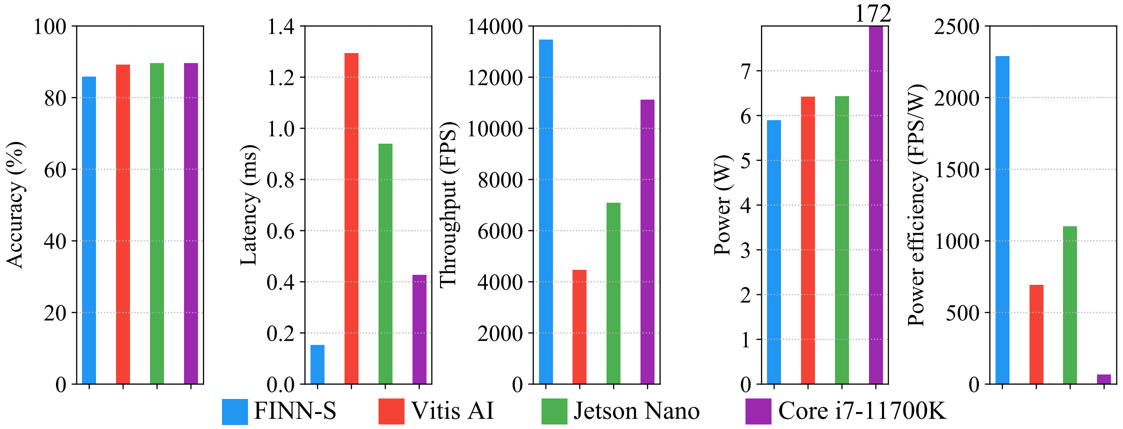


FIGURE 10. Illustration of the comparison results in Table 9.

TABLE 9. Comparison of FINN-S and Vitis AI with reference to the results obtained when using an NVIDIA Jetson Nano Developer Kit and a desktop computer with a Core i7-11700K CPU.

Platform	Precision	TOP1 Acc (%)	Latency (ms)	Throughput (FPS)	Power (W)	Efficiency (FPS/W)
FINN-S (FPGA)	INT4	85.88	0.154	13,475	5.89	2,287
Vitis AI (FPGA)	INT8	89.23	1.293	4,458	6.42	695
Jetson Nano (GPU)	FP16	89.58	0.940	7,091	6.43	1,103
Core i7-11700K (CPU)	FP32	89.59	0.428	11,119	172	64.6

and Vitis-AI-based accelerators and make a comparison with the NVIDIA Jetson Nano Developer Kit and the high-performance desktop computer. For the FINN-based accelerator, we use the configuration C6 (see Table 3); the latency is measured with the stream size set to 1, while the throughput and power efficiency are measured with the stream size set to 256. For the Vitis-AI-based accelerator, we use the configuration of four B512 DPU cores running at 200 MHz; the latency is measured with the number of threads set to 1, while the throughput and power efficiency are measured with the number of threads set to 9. The comparison results are summarized in Table 9.

Figure 10 depicts the data shown in Table 9 to clarify the differences between the four platforms. In terms of TOP1 inference accuracy, with the FP32 DNN model, the desktop computer (CPU-based) platform is the highest as expected. The FP16 and INT8 models running on the NVIDIA Jetson Nano Developer Kit (GPU-based) platform and the Vitis-AI-based accelerator have similar qualities with a decrease of only 0.01% and 0.36%, respectively. The INT4 model running on the FINN-based accelerator achieves a 3.71% lower TOP1 accuracy than the FP32 model. However, it is still higher than 85%, the quality target set by the MLPerf Tiny benchmark [22], [25]. Because of this, we do not fine-tune the training further to close the gap with the FP32 model despite there is still room for improvement. In terms of latency, the FINN-based accelerator is the best, which is 8.4x, 6.1x, and 2.8x better than the Vitis-AI-based accelerator, the GPU-based platform, and the CPU-based platform, respectively. In terms of throughput and power efficiency, the FINN-based accelerator also outperforms the others. With a throughput of 13,475 FPS and power consumption of just 5.89 W, its power efficiency is 3.3x, 2.1x, and 35.4x higher than the Vitis-AI-based accelerator, the GPU-based platform, and the CPU-based platform, respectively.

From the above results, we can see that both of the FPGA-based accelerators are much more efficient than the CPU-based platform. However, when compared with the GPU-based platform, only the FINN-based accelerator is better. This means that the selection of an automation framework suitable for the target DNN model and FPGA device is an important issue. For the case in our experiment, the advantage obtained when using Vitis AI is limited. Nevertheless, with the promising results of FINN-S, we conclude that even in the case of using automation frameworks, DNN accelerators on FPGAs can still yield significant performance, and power efficiency gains compared with both CPUs and GPUs.

V. DISCUSSION

A. INSIGHTS INTO THE COMPARISON RESULTS OF FINN-S AND VITIS AI

Our experimental results show that the FINN-based accelerator outperforms the Vitis-AI-based accelerator by a significant margin (8.4x, 3.0x, and 3.3x in terms of latency, throughput, and power efficiency, respectively). We also found that increasing the level of parallelism of the architecture helped to improve the FINN-based accelerator significantly but it was not the case for the Vitis-AI-based accelerator.

Our first insight is that these results reflect the architectural differences between the two accelerators. The FINN-based accelerator does not require access to off-chip DRAM during the inference process (see Figure 2), and therefore, as long as the DRAM bandwidth is enough for transferring input and output data of the accelerator, increasing the level of parallelism by n times would lead to a performance improvement of roughly n times. This is the case in our experiment (see Tables 3 and 4). We can see that the largest configuration C6 provides 134x better latency and 148x better throughput than the smallest configuration C1. On the other hand, the Vitis-AI-based accelerator has a different characteristic because it requires access to off-chip DRAM during the inference process (see Figure 3). Specifically, different from the FINN-based accelerator, the inference process is controlled by a series of instructions stored in DRAM. The weights and biases also need to be fetched from DRAM. The other sources incurring DRAM traffic are the store and load of intermediate feature maps and output meta-data. Depending on the adopted processing dataflow (e.g., weight-stationary, output-stationary, etc.) and the on-chip buffer capacity, a feature map or a weight/bias may need to be fetched from DRAM multiple times. For all of these reasons, the inference process of the Vitis-AI-based accelerator involves excessive off-chip DRAM access overhead. Therefore, due to limited DRAM bandwidth, we could not improve the performance of the Vitis-AI-based accelerator by increasing the level of parallelism of the architecture like in the case of the FINN-based accelerator. As shown in Table 7, the performance results of the architectures with the highest and lowest level of parallelism (B4096 and B512) are not significantly different from each other. On the other hand, when multiple DPU cores and multiple threads are used, data loaded from DRAM are reused more efficiently. As a result, the DRAM bandwidth requirement is reduced and we observe an improvement in the throughput and thus the power efficiency, as shown in Figure 9.

Our second insight is about how the comparison result of the two automation frameworks may change when a different target FPGA board is used. In our current experiment with the FINN-based accelerator, DRAM bandwidth is still not the performance bottleneck even for the largest configuration. Therefore, for the FINN-based accelerator, in the case of a larger target FPGA, the ability to increase the level of parallelism makes it possible to achieve better performance, and thus also better power efficiency; on the contrary, in the case of a smaller target FPGA, since the level of parallelism must be reduced to be able to fit the design onto the FPGA, lower performance and power efficiency results would be expected. The Vitis-AI-based accelerator has a different trend. We can see from the experimental results that further increasing the level of parallelism of the DPU architecture as well as the number of DPU cores would not help to improve the performance and power efficiency. Also, reducing the configuration to two B512 DPU cores results in only a small negative impact. Therefore, the size of the target FPGA has less influence on the performance and power efficiency than in the case of the FINN-based accelerator. In summary, using a larger target FPGA would sway the comparison result in favor of the FINN-based accelerator; and using a smaller target FPGA would sway the comparison result in favor of the Vitis-AI-based accelerator. Now let us consider what may occur if a DRAM with better or worse bandwidth is used. In our current experiment, DRAM bandwidth is the bottleneck of the Vitis-AI-based accelerator, but not of the FINN-based accelerator. Therefore, using a DRAM with better bandwidth would narrow the gap between the two accelerators; and using a DRAM with worse bandwidth would widen the gap between them.

B. ANALYSIS OF VITIS AI's RESULTS

Our experimental results show that the Vitis-AI-based accelerator is outperformed by the NVIDIA Jetson Nano Developer Kit. Our analysis reveals that this is due to the low utilization rate of the available computational resources of the DPU. For a system containing four B512 DPU cores running at 200 MHz, which is similar to the configuration used in our experiment, the peak performance is 409.6 GOPS. On the other hand, our measured throughput is 4,458 FPS. We estimate that, with the ResNet-8 DNN model, 25.0M operations are necessary for each frame inference. Therefore, by multiplying these two values, we have an effective performance of 111.5 GOPS. Thus, the computational efficiency is only 27.2% of the peak performance. This can be explained by the DRAM bandwidth bottleneck of the Vitis-AI-based accelerator. As we have described in Section V-A, the overlay architecture of the Vitis-AI-based accelerator requires frequent access to DRAM during the inference process. Therefore, if the DRAM bandwidth is not high enough, the computation units will frequently have to wait for accessing DRAM. This leads to the low utilization rate of the computation units, which limits the performance and power efficiency of the accelerator. By using an FPGA board with higher DRAM bandwidth, better performance and power efficiency results would be achieved.

C. TRENDS FOR LARGE DNN MODELS

We project the trends for a large DNN model as follows. If the target FPGA is large enough to be able to increase the level of parallelism (the number of PEs and the number of SIMD lanes per PE) of the accelerator generated by FINN-S, then FINN-S will still outperform Vitis AI in terms of latency, throughput, and power efficiency. However, if the target FPGA is small, then either it will be impossible for FINN-S to fit the DNN model onto the FPGA or the resulting accelerator will have a low level of parallelism and be worse than that generated by Vitis AI. This is because like the original FINN, FINN-S assumes that all the weights and biases of the model are hard-wired or stored entirely on-chip, which requires a vast amount of FPGA resources.

Note that our area of interest is in embedded systems, and therefore, it is undesirable to use a very large FPGA. To make it possible to implement large DNN models on small FPGAs using FINN-S, some fundamental changes to the framework (for example, by storing all or a part of the weights and biases in DRAM) is required. Exploring this is a promising direction to extend our work.

D. COMPARISON BETWEEN FPGA AND ASIC IMPLEMENTATIONS

It has been known that an ASIC implementation is typically about an order of magnitude faster and more efficient than an FPGA counterpart. For example, Nurvitadhi et al. [26] have quantitatively explored this in the case of accelerating binarized neural networks. They showed that with the same microarchitecture, a 14nm ASIC implementation was 4.5x faster and 11x more power efficient than an Aria 10 FPGA implementation. While FPGA-based accelerators generated by automation frameworks like FINN and Vitis AI may be less competitive than manual RTL designs, the difference is expected to be reasonable so that the gap with ASIC implementations is not significantly changed.

In addition to algorithm- and architecture-level innovations, which mostly can be applied to both ASIC and FPGA implementations, state-of-the-art ASIC studies have been investigating how to incorporate emerging technologies like digital-analog hybrid computing and processing in memory for better performance and power efficiency. PUMA [27] and ISAAC [28] are two representative examples in this category. The introduction of highly efficient memristor-based analog processing units enables these ASIC accelerators to outperform conventional designs by a significant margin. For example, up to 2.27× improvement in inference throughput and 1.65× improvement in power efficiency over Google TPU, a conventional ASIC accelerator, are reported in the study of PUMA.

On the other hand, FPGA vendors have also been adapting their FPGA architectures to better support DNN inference processing. For example, Intel has recently incorporated hardened AI Tensor Blocks that can perform matrix and vector operations into their Stratix 10 NX architecture [29]. AMD Xilinx has done a similar attempt with their VERSAL architecture [30]. With this trend, the gap of performance and power efficiency between FPGAs and ASICs in DNN inference processing may be narrowed in the future. Therefore, given the fact that FPGAs have two unique advantages over ASICs, the reconfigurability that is highly desirable in fast-moving fields like designing DNN models and the lower development effort and unit cost when the manufacturing volume is not so high, it can be expected that FPGA-based accelerators would become an even more attractive solution.

VI. CONCLUSION

In this paper, we conducted a fair and quantitative comparison of two representative automation frameworks for generating FPGA-based DNN inference accelerators: FINN and Vitis AI. First, we extended the development flow of FINN to provide a comparison with Vitis AI using the same target hardware and the same DNN model. With our extensions, a wide variety of FPGA devices and DNN models can now be supported. Then, we used the CIFAR-10 ResNet-8 model from the MLPerf Tiny benchmark and compared the FINN-based and Vitis-AI-based accelerators with reference to the results obtained when using an NVIDIA Jetson Nano Developer Kit with a similar power budget and a high-performance desktop computer with a Core i7-11700K CPU. In the comparison, we searched for the configuration where each platform performed best. We adjusted the folding factors and stream size for FINN, the DPU architecture, the numbers of DPU cores and threads, and the clock frequency for Vitis AI, and the batch size for both the Jetson Nano Developer Kit and the desktop computer.

Our results showed that the accelerator generated by FINN was the best in terms of latency, throughput, and power efficiency. This indicates the high potential of automation frameworks for DNN inference acceleration on FPGAs. We also discussed the gap between FINN and Vitis AI and provided our insights into it.

REFERENCES

[1] X. Wang, V. Goyal, J. Yu, V. Bertacco, A. Boutros, E. Nurvitadhi, C. Augustine, R. Iyer, and R. Das, "Compute-capable block RAMs for efficient deep learning acceleration on FPGAs," in *Proc. IEEE 29th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2021, pp. 88–96.

[2] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating binarized convolutional neural networks with software-programmable FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 15–24.

[3] H. Fan, S. Liu, M. Ferianc, H.-C. Ng, Z. Que, S. Liu, X. Niu, and W. Luk, "A real-time object detection accelerator with compressed SSDLite on FPGA," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2018, pp. 14–21.

[4] S. Yan, Z. Liu, Y. Wang, C. Zeng, Q. Liu, B. Cheng, and R. C. C. Cheung, "An FPGA-based MobileNet accelerator considering network structure characteristics," in *Proc. 31st Int. Conf. Field-Program. Log. Appl. (FPL)*, Aug. 2021, pp. 17–23.

[5] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713.

[6] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 1–30, 2017.

[7] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A framework for fast, scalable binarized neural network inference," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 65–74.

[8] Xilinx. *Vitis-AI: Vitis AI is Xilinx's Development Stack for AI Inference on Xilinx Hardware Platforms, Including Both Edge Devices and Alveo Cards*. Accessed: Jan. 2023. [Online]. Available: https://github.com/Xilinx/Vitis-AI

[9] F. Fahim et al., "Hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices," 2021, *arXiv:2103.05579*.

[10] T. Moreau, T. Chen, L. Vega, J. Roesch, E. Yan, L. Zheng, J. Fromm, Z. Jiang, L. Ceze, C. Guestrin, and A. Krishnamurthy, "A hardware-software blueprint for flexible deep learning specialization," 2018, *arXiv:1807.04188*.

[11] F. Jentzsch, Y. Umuroglu, A. Pappalardo, M. Blott, and M. Platzner, "RadioML meets FINN: Enabling future RF applications with FPGA streaming architectures," *IEEE Micro*, vol. 42, no. 6, pp. 125–133, Nov. 2022.

[12] A. Ushiroyama, M. Watanabe, N. Watanabe, and A. Nagoya, "Convolutional neural network implementations using vitis AI," in *Proc. IEEE 12th Annu. Comput. Commun. Workshop Conf. (CCWC)*, Jan. 2022, pp. 365–371.

[13] M. Verucchi, G. Brilli, D. Sapienza, M. Verasani, M. Arena, F. Gatti, A. Capotondi, R. Cavicchioli, M. Bertogna, and M. Solieri, "A systematic assessment of embedded neural networks for object detection," in *Proc. 25th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2020, pp. 937–944.

[14] P. Plagwitz, F. Hannig, M. Ströbel, C. Strohmeyer, and J. Teich, "A safari through FPGA-based neural network compilation and design automation flows," in *Proc. IEEE 29th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2021, pp. 10–19.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[16] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, "FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, pp. 1–23, Sep. 2018.

[17] Xilinx. *DPUCZDX8G for Zynq UltraScale+ MPSoCs Product Guide (PG338)*. Accessed: Jan. 2023. [Online]. Available: https://docs.xilinx.com/r/en-US/pg338-dpu/Core-Overview

[18] Q. Ducasse, P. Cotret, L. Lagadec, and R. Stewart, "Benchmarking quantized neural networks on FPGAs with FINN," in *Proc. DATE Friday Workshop Syst.-Level Design Methods Deep Learn. Heterogeneous Archit. (SLOHA)*, 2021, pp. 13–18.

[19] J. Su, N. J. Fraser, G. Gambardella, M. Blott, G. Durelli, D. B. Thomas, P. H. W. Leong, and P. Y. K. Cheung, "Accuracy to throughput trade-offs for reduced precision neural networks on reconfigurable logic," in *Proc. Int. Symp. Appl. Reconfigurable Comput. Archit. (ARC)*, 2018, pp. 29–42.

[20] M. Blott, N. J. Fraser, G. Gambardella, L. Halder, J. Kath, Z. Neveu, Y. Umuroglu, A. Vasilciuc, M. Leeser, and L. Doyle, "Evaluation of optimized CNNs on heterogeneous accelerators using a novel benchmarking approach," *IEEE Trans. Comput.*, vol. 70, no. 10, pp. 1654–1669, Oct. 2021.

[21] Xilinx. *Brevitas: Quantization-Aware Training in PyTorch*. Accessed: Jan. 2023. [Online]. Available: https://github.com/Xilinx/brevitas

[22] C. Banbury et al., "MLPerf tiny benchmark," 2021, *arXiv:2106.07597*.

[23] A. Krizhevsky, V. Nair, and G. Hinton. *CIFAR-10*. Accessed: Jan. 2023. [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[24] NVIDIA-AI-IoT. *Torch2Trt: An Easy to Use PyTorch to TensorRT Converter*. Accessed: Jan. 2023. [Online]. Available: https://github.com/NVIDIA-AI-IOT/torch2trt

[25] MLCommons. *MLPerf Tiny Benchmark Suite*. Accessed: Jan. 2023. [Online]. Available: https://github.com/mlcommons/tiny/tree/master/benchmark

[26] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2016, pp. 77–84.

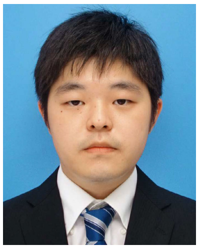
[27] A. Ankit, I. E. Hajj, S. R. Chalamalasetti, G. Ndu, M. Foltin, R. S. Williams, P. Faraboschi, W.-M.-W. Hwu, J. P. Strachan, K. Roy, and D. S. Milojicic, "PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference," in *Proc. 24th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Apr. 2019, pp. 715–731.

[28] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 14–26.

[29] M. Langhammer, E. Nurvitadhi, B. Pasca, and S. Gribok, "Stratix 10 NX architecture and applications," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2021, pp. 57–67.

[30] Xilinx. (2022). *Xilinx VERSAL*. [Online]. Available: https://www.xilinx.com/products/silicon-devices/acap/versal.html

FUMIO HAMANAKA received the B.E. degree from the Department of Electrical Engineering and Computer Science, University of Hyogo, Japan, in 2021. She is currently pursuing the master's degree with the School of Computing, Tokyo Institute of Technology, Japan. Her research interests include computer architecture and FPGA computing.



TAKASHI ODAN received the B.E. degree from the Department of Computer Science, Tokyo Institute of Technology, Japan, in 2021, where he is currently pursuing the master's degree with the School of Computing. His research interests include computer architecture and FPGA computing.

KENJI KISE (Member, IEEE) received the B.E. degree from Nagoya University, in 1995, and the M.E. and Ph.D. degrees in information engineering from The University of Tokyo, in 1997 and 2000, respectively. He is a Professor with the School of Computing, Tokyo Institute of Technology. His research interests include computer architecture, adaptive computing, and parallel processing.

THIEM VAN CHU (Member, IEEE) received the Ph.D. degree from the Tokyo Institute of Technology, in 2018. He joined as an Assistant Professor with the School of Information Science, Japan Advanced Institute of Science and Technology. In 2020, he joined with the Tokyo Institute of Technology. His research interests include the intersection of computer architecture, reconfigurable computing, and machine learning.