Efficient FPGA-Based Convolutional Neural Network Implementation for Edge Computing

Cuong Pham-Quoc [1,2,*] and Tran Ngoc Thinh [1,2]

[1] Department of Computer Engineering, Ho Chi Minh City University of Technology (HCMUT), Ho Chi Minh City, Vietnam

[2] Department of Computer Engineering, Vietnam National University, Ho Chi Minh City (VNU-HCM), Ho Chi Minh City, Vietnam

*Correspondence: cuongpham@hcmut.edu.vn (C.P.-Q.)

***Abstract*—In recent years, accelerating convolutional neural networks on Field Programmable Gate Array (FPGA) to improve the performance of the inference phase of artificial intelligent edge computing applications is a promising approach. This paper presents our proposed architecture for building a convolution neural network acceleration core on FPGA. The proposed FPGA-based core targets edge computing platforms where hardware resources and power efficiency are essential requirements. We use the MobileNet neural network model for image classification as a case study to evaluate our proposed system. We compare our work with a quad-core ARM Cortex processor at 1.2GHz and achieve speed-ups by up to 14.77× convolution operators. Although our system is worse than a 6-core Intel Core i7 processor, it is more energy-efficiency than the Intel processor. Our proposed system is the best fit for edge computing.**

***Keywords*—Field Programmable Gate Array (FPGA), convolutional neural network, hardware accelerator, MobileNet**

I. Introduction

Recently, with the rapid development of Artificial Intelligence (AI), many research fields such as image/voice recognition, object detection, anomaly-based detection systems, etc., have been applied in many different areas of life from IoTs-based applications processed on edge computing devices to expert systems executed on high-performance computers. Among famous neural network models, Convolutional Neural Network (CNN) attracts more studies in various application domains and dominates hardware accelerator-based research due to their effectiveness. However, recent CNNs have become deeper and thus require more and more computing power and storage of the processing platforms.

CNN models are becoming increasingly accurate by utilizing considerable amounts of data and needing enormous computational power [1]; for instance, the VGG19 CNN model takes over 500MB of memory for parameters and processes up to 39B+ Floating-Point Operations (FLOPs) to classify images with the resolution of 224×224 pixels [2]. In contrast, edge computing devices only provide good performance, less memory, and a modest energy budget [3]. Even conventional CPUs are incapable of delivering sufficient computational power for CNN-based applications without consuming more than 1 joule (J) of energy per 1 GOP. The training phase of AI-based applications can be done with many high-performance computing platforms like GPUs or supercomputers. Meanwhile, the inference phase for edge computing platforms, e.g., IoT-based applications, still needs to be executed on small devices with low processing ability and limited power consumption.

At the end of Moore's Law [4], the hardware accelerator architecture that includes General Purpose Processors (GPP) augmented with some unique hardware-based computing cores for a particular purpose is a promising approach. The strategy of co-designing the hardware (special cores) and software (GPP) of computing systems, which includes embedded and edge computing, has the potential to continue increasing computing efficiency in terms of processing time. The paradigm's most promising strategy involves using specialized hardware cores to handle the intensive application functions. These processing units have been tuned for the tasks and the targeting system. Currently, researchers typically focus on the two best-known products: Application Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGAs). However, due to their reconfigurability, FPGAs provide more significant optimization and reuse opportunities than ASICs.

This paper proposes an efficient FPGA-based architecture for developing CNN-based systems targeting edge computing platforms. Our work aims to accelerate the performance and use hardware resources efficiently so that edge devices can process the inference phase in real time. To validate the proposed FPGA-based system, we deploy the MobileNet model [5, 6] on an Ultrascale+Xilinx FPGA board [7] as our experiments. The experimental results show that our system achieves 14.77× higher performance than a Quadcore ARM processor while using fewer hardware resources than other work in the literature. The main contributions of our work include three folds.

Manuscript received January 16, 2023; revised March 2, 2023; accepted April 5, 2023; published May 24, 2023.

(1) We propose an architecture to build CNN on FPGA devices targeting edge computing systems efficiently.
(2) We build the MobileNet on a low-cost FPGA device suitable for edge computing.
(3) We present our experimental results for other researchers to reference in the future.

The rest of the paper is organized as follows. Section II presents the background on CNN, the MobileNet model, and literature-related work. We then introduce the proposed FPGA-based architecture for CNN targeting edge computing platforms in Section III. We show our FPGA-based system implementation with the MobileNet model to validate the architecture in Section IV. Section V introduces our experimental results and compares them with other studies. Finally, we conclude our paper in Section VI.

II. Background and Related Work

In this section, we first briefly introduce the background of CNN and MobileNet. Then, we summarize related work in the literature.

A. Convolutional Neural Networks

As mentioned above, Convolution Neural Network (CNN) model is mainly used for deep learning-based applications due to its efficiency. Therefore, many CNN-based models have been proposed in recent years for different purposes. In general, the principal operation of the standard convolution is shown in Fig. 1 (adapted from [8]).

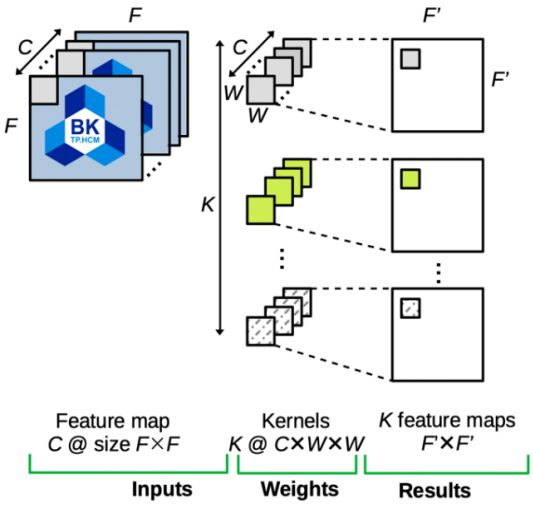

Figure 1. The standard convolution operation.

As illustrated in Fig. 1, the input feature map includes C channels, each of them is a matrix of $F \times F$ size. Convolution operations will be conducted between the input feature maps and K kernels (also called filters) which are $W \times W$ matrices ($W < F$). Each kernel also consists of C $W \times W$ matrices for convoluting with C matrices of each feature map channel. Kernels are applied to different elements of the input feature map to create 2D matrices output feature maps through multiple Multiply-Accumulate Operators (MAC). The convolution operator is depicted in Eq. (1), where *act_func* is an activation function (usually Tanh, Sigmoid, etc.) and b is a bias number.

$$F'[i,j] = act\_func\left(\sum_{m=i}^{m+w}\sum_{n=j}^{n+w} F[m,n] \times K[m-i, n-j]\right) + b \quad (1)$$

B. MobileNet Model

MobileNet model [5, 6], initially developed by Google, is one of the well-known CNN models that is effective for mobile-computing applications. It is hence a good fit for edge computing. As a result, we decided to utilize the MobileNet model as our case study to support the architecture designed in this paper. The model has 13 layers of depth-wise separable convolution, one layer with an average-pooling function, and a wholly linked layer. Two operations—depth-wise and pointwise convolutions—make up a depth-wise separable convolution layer.

The former operation, depth-wise, applies each filter (K_i) to each channel (C_i) of feature maps (F) in depth-wise convolution, which computes the same operations as the standard convolution. However, unlike the standard convolution operator, depth-wise convolution merely filters a feature map's channels to produce different multiple matrices. Meanwhile, the standard one combines them to generate a new single feature map (i.e., we get C feature maps F' per kernel instead of one as in Fig. 1). Consequently, a new feature map is constructed using a pointwise convolution layer and a 1×1 kernel. As a result, a computation reduction of $\frac{1}{C} + \frac{1}{W^2}$ can be achieved when compared to the standard convolution, where C represents the number of feature maps and W is the kernel size. For example, if we consider nine feature maps and kernel size is 3×3, we can reduce computing time by 4.5 times. Meanwhile, the pointwise convolution uses only 1×1 kernel. Therefore, a processing core can do both depth-wise and pointwise convolutions.

C. Related Work

This section briefly analyzes FPGA-based CNN accelerators proposed in the literature. In recent years, many studies have been conducted to improve the performance of the CNN inference phase on FPGA devices. However, they mainly focus on high-performance or high-end computing platforms instead of edge computing like our work. These proposals can be classified into three groups according to the primary optimization technique including: (1) exploiting massive FPGA resources to create multiple computing modules; (2) reducing computation complexity by accepting less accuracy; and (3) caching input data to reduce communication overhead.

1) Exploiting massive FPGA resources

The most significant advantage of FPGA devices is the massive number of resources for building computing modules. Hence, many approaches in the literature define an array of computing nodes or un-roll computationally

intensive loops to achieve higher performance in terms of processing time due to throughput increasing. To reach a throughput of 1 pixel per cycle, Guo and Han *et al.* presented the Aristotle architecture to improve CNNs computation's performance on FPGA [9]. Based on the systolic array form, this architecture connected many Processing Elements (PE) in FPGA fabrics. Li and Fan *et al*. demonstrated their CNN accelerator core, which enables the pipeline to compute all layers simultaneously [10]. An FPGA-based CNN core with several layer clusters was created by Lin and Yin *et al*. [11]. A calculating core for CNN using parallel structures and reconfigurable PEs was developed by Liu and Dou *et al*. [12]. Ma and Cao *et al*. synthesized an FPGA-based CNN architecture using all available hardware resources to get the best feasible performance [13]. Yang and He *et al*. presented the innovative parallel convolution with a binarized architecture [14]. An FPGA-based CNN acceleration pipeline system with a fine-grained and layer-based architecture was proposed by Zhang and Wang *et al*. [15]. Podili and Zhang *et al*. created a computing engine for the standard convolution using the Winograd technique, which is highly parallelized [16]. Motamedi and Gysel *et al*. used every hardware resource to create a parallel deep CNN processing core [17].

2) Reducing computation complexity

Although the previous approach can offer high throughput due to parallelism computing, it needs to improve its working frequency since several resources are used. Then, system performance in terms of processing time needs to improve, especially for real-time processing. Therefore, the second group of proposals in the literature tries to reduce the complexity of CNN computation to enhance further system performance and exploit parallelism. One of the most promising techniques for this approach is the Binarized Neural Network (BNN) [18]. In this approach, 1-bit kernels and the XNOR binary operator are used instead of floating-point and the MAC operators [19–25]. However, accuracy compared to the standard neural network with floating-point computation is the biggest drawback of this technique. Therefore, to improve the accuracy, some proposals only reduce the number of bits for representing kernel values instead of using binary [26–30].

3) Caching input data

Although FPGAs offer many hardware resources for computation, they suffer from the cache (Block RAM) limitation for storing feature maps and kernels. Hence, well-caching will improve system performance because data communication overhead may contribute up to 50% of execution time [31]. Researchers have taken this issue into account and proposed several techniques for caching data to reduce additional processing time [32–38].

III. Proposed Architecture

This section proposes an FPGA-based architecture for computing convolution operators in a pipeline model to improve performance. Then, we introduce a dataflow for processing data in a pipeline model with the proposed FPGA core.

A. The FPGA-Based Pipeline Convolution Operator Core

As depicted in Section II.A, the standard convolution includes multiple multiply-accumulate operators where multiplications can be done in parallel. Therefore, we propose a so-called row-oriented FPGA-based pipeline architecture for efficiently calculating the standard convolution on FPGA devices. In this architecture, several MAC modules are organized in a matrix-based form for computing required operations. Ideally, the number of MAC modules equals the number of kernel elements to achieve the most optimized execution time.

Fig. 2 illustrates the proposed architecture for 3×3 kernels where nine MAC modules are organized in three rows and three columns (MACs are responsible for calculating all operators). Two pipeline registers are used to separate the columns so that the core can compute in the pipeline model. These registers help the core process convolution in the pipeline model to improve performance. For storing input feature maps, a buffer is organized as multiple FIFO; each FIFO stores one row of input maps (F). Finally, a summing module is responsible for adding values from MAC and a bias to create the results of output feature maps (F').

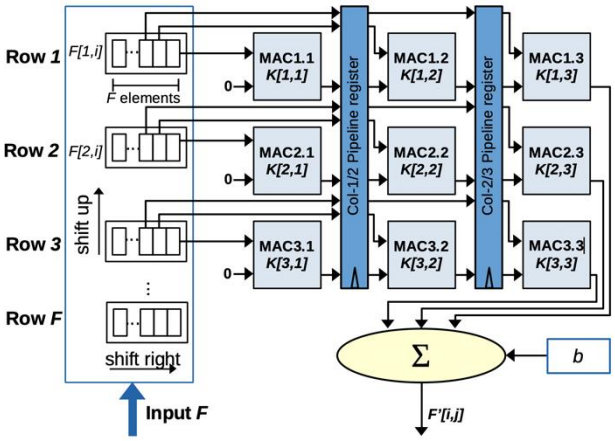

Figure 2. The proposed architecture for 3×3 kernels.

1) Buffer

The most optimized buffer size is the size of an input feature map (F). However, as mentioned in Section II.B, the amount of local memory in FPGA devices is a limitation. Therefore, the buffer usually cannot simultaneously store the entire input feature maps.

Our buffer is organized in a row-oriented form with several FIFOs; each of them hosts a row of F. In each FIFO, all elements in an F's row are stored from right to left. The FIFOs will be shifted to the right after each cycle to get the next element. As shown in Fig. 2 (an example of core for 3×3 kernel), the three first elements in FIFOs are collected per cycle to send to three columns of MACs. The first element is directly sent to the first column, while the next two are forwarded to the second and the third columns through pipeline registers. When the core finishes processing all elements of the first three rows, the buffer will shift data in FIFOs up to remove the first row and fetch

a new row to the last FIFO. In this way, data is cached from beginning to end. We only need to transfer data of the feature map from external memory to buffer one time.

2) Multiply-Accumulate (MAC) module

Fig. 3 depicts the micro-architecture of the MAC module used in the core. The primary purpose of this module is to first compute a multiplication of a kernel's element $K[i,j]$ with an input map's element $F[x,y]$. The multiplying result is then added with the result of the previous column $MAC[i,j-1]$, except the first column of MACs (0 input for the first column of MACs, as shown in Fig. 2). Next, the result of each MAC ($MAC[i,j]$) will be forwarded to the next column through pipeline registers except the last column. Finally, the output of MACs in the final column will be accumulated together and added with the bias (b) value by an accumulator (the ∑ block in Fig. 2). The result of the accumulator block is the value of one element in the output feature map F'.

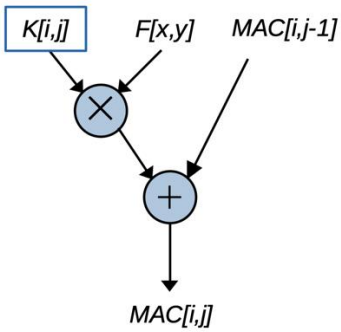

Figure 3. The MAC module architecture.

Since elements of a kernel are not changed for entire convolution, the core needs to collect them only one time and store each element into each MAC ($MAC[i,j]$ keeps value of $K[i,j]$). Due to data-independence, all MACs in the core can be executed in parallel. Next section will illustrate the processing of these MACs in a pipeline model.

B. The Pipeline Dataflow

In this subsection, we explain the processing of the pipeline convolution computing core by introducing the execution data flow. Consider a convolution with a 3×3 kernel like Fig. 2, Fig. 4 shows the processing flow of the first four cycles. We group the MACs of the core into columns and separate processing cycles by a rectangle (representing the pipeline registers).

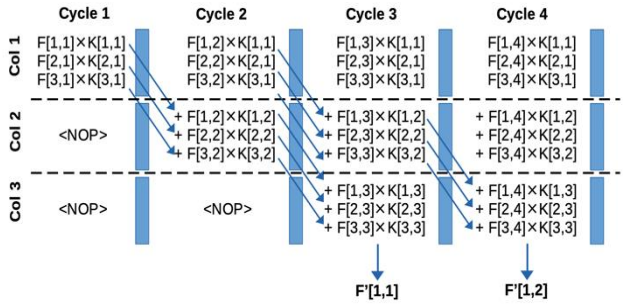

Figure 4. The dataflow example with 3×3 kernels in the first 4 cycles.

1) Cycle 1

In the first cycle, MAC modules in Column 1 (Col 1) get the first elements of the first three rows ($F[1,1]$, $F[2,1]$, and $F[3,1]$) to multiply with the kernel's first column ($K[1,1]$, $K[2,1]$, and $K[3,1]$). Please note that the values of kernels are stored in the MAC modules for the entire process. In this cycle, the second and third elements of each FIFO are sent to MAC modules in the two following columns ($F[1,2]$, $F[2,2]$, and $F[3,2]$ to the second column while $F[1,3]$, $F[2,3]$, and $F[3,3]$ to the third column). However, due to the pipeline registers, these elements will arrive in the second and third columns of MAC modules in Cycle 2 and Cycle 4, respectively. Therefore, in Cycle 1, MAC modules in Cols 2 and 3 do nothing (<NOP>). Finally, as mentioned above, MAC modules in Col 1 do not need to add the multiplication results to anything. Instead, the outputs of these MAC modules are forwarded to the next column through the Col-1/2 pipeline register.

2) Cycle 2

In Cycle 2, FIFOs shift elements to the right for the subsequent cycle computation. The MAC modules in Col 1 continue to fetch the first elements from FIFOs (the second one of each row $F[1,2]$, $F[2,2]$, and $F[3,2]$) for computing multiplication with the kernel's first row, like in Cycle 1. At this time, Cycle 2, values $F[i,2]$ and results of MAC modules in Col 1 ($MAC[i,1]$)) have passed the Col-1/2 pipeline register and arrived at Col 2 in the second row. Therefore, Cols 1 and 2 can compute in parallel with different data inputs. As shown in Fig. 4, values in the first column of the kernel ($K[i,1]$) are multiplied with elements at the second position of the first three rows from input maps ($F[i,2]$). Meanwhile, MAC modules in the second column (MACi,2) compute the multiply-accumulate operator of $F[i,2]$ (second elements of each input feature maps' row), $K[i,2]$ (the second column of the kernel), and the results of MACi,1 (first column of MAC modules). Again, like the first cycle, MAC modules in the last column are still idle in Cycle 2 because data inputs still have been delayed by the Col-2/3 pipeline register.

3) Cycle 3

From this cycle, as illustrated in Fig. 4, the three columns of MAC modules can fully compute in parallel for multiple-accumulate operators between different data input feature maps ($F[x,y]$) and kernel ($K[i,j]$). Like the previous cycle, FIFOs also shift elements to the right for continuing computation. MAC modules in the first column (MACi,1) now fetch and compute multiplication between elements at the third position of the input feature maps' three top rows ($F[i,3]$). At the same time, MAC modules in the second and third columns process data for the two following columns of the kernel. The core produces the convolution result per cycle from this cycle because the MAC modules in the third column have been done. As depicted in Fig. 4, we get output feature map element $F'[1,1]$ (the first element in the first row) at the end of Cycle 3 (from the output of the ∑ block).

4) Cycle 4 to (F+2)

From Cycle 4 to Cycle F+2, where F is the number of elements in one row of the input feature maps, the data flow continues with the same model discussed in Cycle 3.

At each cycle, the core can compute one element of the output feature map $F'[1,j]$. After F+2 cycles, the first row of the output feature maps has been created successfully. Consequently, the buffer will shift FIFOs up (remove the first row of input feature maps and fetch the fourth one) to further processing with the next batch of rows. Assume that the size of input feature maps is $F \times F$ (F rows and F columns), the number of cycles that the core needs to compute all multiple-accumulate operators is depicted in Eq. (2).

$$T_{total} = F \times (F + 2) + t_{comm} \quad (2)$$

where, t_{comm} is the number of cycles for fetching next row of input feature maps to the buffer. This value depends on the size of the input feature maps and the communication infrastructure for transferring data from external memory to the local buffer.

IV. System Implementation

This section introduces our implementation with the MobileNet model [5, 6] as our case study to verify and validate the proposed hardware architecture for the CNN computing core. Although our system is designed for all versions of MobileNet (including standard convolution, depth-wise, and pointwise operators), currently, we test the system with MobileNetV2 only for comparison. We use the Zynq FPGA UltraScale+MPSoC platform [7] for hosting the system. The testing platform includes a 4-core ARM Cortex processor and FPGA fabrics for building various components including DMA Controller, Buffers, and convolution cores. Fig. 5 briefly depicts our testing system with the MobileNet model on the platform. This picture only shows the system's main components due to space limitations.

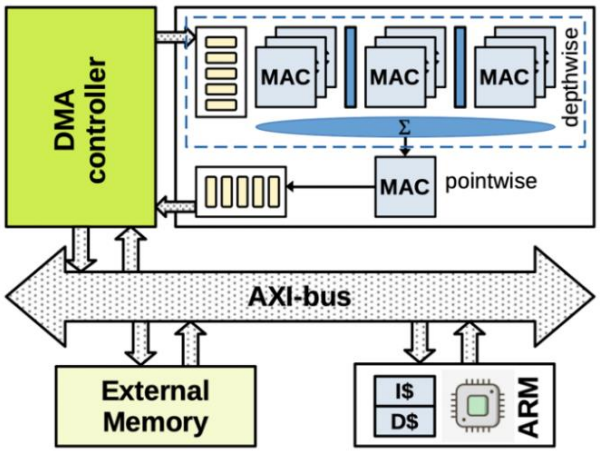

Figure 5. The MPSoC FPGA-based system for the MobileNet model using our hardware computing core.

In this implementation, we follow the hardware accelerator architecture in which part of the program will be executed by the host processor (the ARM one in our system), and the time-consuming part will be processed by the accelerator cores (the convolution cores in our work). Input feature maps, kernels, and other data are stored in the external memory that communicates with the core on FPGA and the ARM processor through an AXI bus, a specific type of bus for Xilinx FPGA. The ARM Cortex is responsible for handling the computing cores and DMA controller and processing non-critical parts of the application.

We build a DMA controller in the FPGA fabrics to transfer input feature maps (F) from the external memory to our local buffer and output feature maps (F') back to the external memory after processing. With DMA support, data communication overhead can be reduced because the host processor can do other tasks while data is being transferred. The convolution computing cores can start processing when the required data is fetched enough to the input buffer.

As mentioned above, the MobileNet model conducts several interleaved depth-wise and pointwise convolutions. While the depth-wise convolution will be done by our proposed computing core presented in the previous section, the pointwise computation takes only 1×1 kernel. Therefore, a MAC module is used for processing the pointwise convolution. This MAC module receives the depth-wise convolution core results to multiply with the 1×1 kernel. The multiplication results are then added with bias value (b), like the standard convolution operator. The final results will be stored in the output buffer so that the DMA can transfer back to the external memory to process further by the host processor.

While the DMA controller and AXI-bus are supported by Xilinx IP core, other components on FPGA (MAC, buffers, connections, accumulator) are developed manually by Verilog-HDL (a hardware description language). The entire system is then synthesized and mapped onto the FPGA MPSoC platform by tools provided by Xilinx. The following section will analyze the experiments regarding hardware resource usage, execution time, and power consumption.

V. Experiments

In this section, we introduce the experiments we conducted to validate the system implemented in the previous section. We present the setup of our experiment first. We then report various results of our experiments.

A. Experimental Setup

The Ultra96v2 board that houses a Xilinx Zynq FPGA with 70K+ Look-up Tables, 950 KB Block RAMs, and a 4-core ARM Cortex functioning at up to 1.5 GHz, is used to evaluate the system described in Section IV. All the computing cores and buffers in the FPGA chip's programmable logic section are created using Verilog-HDL. Additionally, an AXI bus is used by the ARM Cortex to connect to the computational cores as a host processor.

The Ultra96v2 board that houses a Xilinx Zynq FPGA with 70K+ Look-up Tables, 950 KB Block RAMs, and a 4-core ARM Cortex functioning at up to 1.5 GHz, is used to evaluate the system described in Section IV. All the computing cores and buffers in the FPGA chip's programmable logic section are created using Verilog-

HDL. Additionally, an AXI bus is used by the ARM Cortex to connect to the computational cores as a host processor.

To evaluate the efficiency of the proposed convolution core and the system for the MobileNet model, we compare our system with the processing of only an ARM processor and an Intel processor. In more detail, experiments with the following platforms are conducted.

(1) The proposed system with hardware accelerator architecture (so-called "Our system"): both ARM processor and the hardware cores for depth-wise and pointwise are used for processing the MobileNet model. While the hardware cores are responsible for executing convolutions operators of the model, the ARM processor processes all other parts of the model. In this system, the ARM processor functions at the maximum working frequency (1.5 GHz) while the hardware cores run at 150 MHz.

(2) The 4-core ARM processor (so-called "ARM system"): all the cores of the processor are used for processing the MobileNet model in parallel with the Pytorch library [39]. Like Our system, the ARM processor functions at the maximum working frequency of 1.5 GHz to make a fair comparison. However, due to only the ARM processor used, the FPGA fabrics (hardware cores) are set to idle.

(3) A high-end Intel processor (so-called "Intel system": like the ARM system, the entire MobileNet model is processed by a 6-core Intel Core-i7 9750H processor functioning at 2.4 GHz in parallel with the Pytorch library. We conduct this experiment to demonstrate the goodness of our system because the Intel Core-i7 represents high-performance platforms.

In our experiments, we do not compare the accuracy of our proposed system because our system produces the same output feature maps as the two other platforms.

B. Experimental Results

In this section, we present the experimental results with the system implemented in the previous section. We synthesize the system first to get the working frequency and usage of hardware resources. We also compare our system with state-of-the-art ones in terms of synthesis results. We then compare the three systems (Our system, ARM system, and Intel system) to show the goodness of our proposed system in terms of the system performance.

1) Synthesis results

As mentioned above, we use Verilog-HDL for building the proposed system. We then synthesize our project by Vivado Design Suite provided by Xilinx with the target frequency of 150MHz. Our hardware resource usage and the maximum working frequency are summarized in Table I. As shown in the table, we manage to use almost all available resources of the platform, especially the on-chip memory (BRAM). The synthesis report shows that our system can work with 159 MHz working frequency while consumes at most 4.15 W power consumption.

TABLE I. SYNTHESIS RESULTS OF THE PROPOSED SYSTEM IN TERM OF HARDWARE RESOURCES USAGE

Resource types	Used	Available	Utilization (%)
Look-up table (LUT)	57,887	70,560	82.04
Flip-flop (FF)	80,325	141,120	56.92
Block RAM (BRAM)	210.5	216	97.45
Digital signal processing unit (DSP)	244	360	67.78

2) State-of-the-art comparison

Table II compares our proposed system and state-of-the-art FPGA-based MobileNet implementations regarding hardware resources, working frequency, and power consumption. As shown in the table, our system uses the cheapest device (~250 USD [7]) and requires fewer hardware resources than work in [40–43]. Currently, our system is not better than the work in [44], but the system in [44] target high-end device (LUTs' size is bigger) instead of low-cost as ours. Therefore, our proposed system and platform are the most suitable for edge computing compared to the others. This is the ultimate goal of our work.

In terms of Giga Operation Per Second (GOPS), we achieve up to 96.3 GOPS, better than [40, 42] ([43] and [44] do not provide this value). However, our system is worse than the system in [41] because it uses extremely huge resources with a high-end FPGA family. The resources used for the system in [41] are 16.2× higher than ours. Therefore, it is not suitable for edge computing systems. Meanwhile, although we cannot achieve such a high performance in terms of GOPS, we use less resources and low-cost FPGA devices that are the best for edge computing. In terms of Digital Signal Processing (DSP) units, our system is better than all compared works when only 244 DSP units are used. The lower amount of DSP used can help the system's working frequency increase because DSP units in FPGA devices can be placed in a broad range of areas. In addition, more DSP used may lead to more extended wire connection that reduces operating frequency due to the longer critical paths.

TABLE II. COMPARISON WITH STATE-OF-THE-ART FPGA-BASED MOBILENET IMPLEMENTATIONS

	[40]	[41]	[42]	[43]	[44]	Ours
FPGA device	XCZU19EG	Stratix 10	Zynq 7100	XCZU19EG	Virtex 7	Zynq 7000
LUT	139K	926K	142K	369K	57K	57K
FF	55K	583K	187K	391K	79K	80K
DSP	1452	297	1926	1020	937	244
Frequency (MHz)	150	156	100	200	225	150
Power (W)	-	-	4.083	7.35	3.25	4.15
GOPS	91.2	3536	17.11	-	-	96.3

C. Performance Evalution

To evaluate the proposed system in terms of execution time (performance), we use images of 128×128×3 as input feature maps and $3 \times 3 \times 3$ kernels for depth-wise convolutions. In contrast, output feature maps of depth-wise layers are convoluted with 1×1 kernel in pointwise layers. The same data set is used for the three systems mentioned above (Our system, ARM system, and Intel system). Table III shows our experimental results (size of each convolution and total execution time) of the three systems and the speed-up of our system compared to the ARM system. Although the size of input feature maps and kernels are fixed for each layer, as shown in the table, execution times jitter due to the shared bus and system services on the host processor. Therefore, to collect the execution time for each system, we test with a data set containing 256 images and get the average execution time. As shown in the table, our system outperforms the ARM system in execution time. We achieve a speed-up of $14.77\times$ when compared to the ARM processor only. However, our system is worse than the Intel one 1.56× since the Intel processor used in this experiment is the high-end one.

In terms of power consumption, our system and the ARM system require the same power consumption of 4.15W (power consumption of the MPSoC platform), while the Intel processor consumes 45W. Therefore, our system is much more energy-efficient than the Intel processor when it needs only 0.04J per frame instead of 0.3J of the Intel processor.

TABLE III. PERFORMANCE ANALYSIS AND COMPARISON

Layer No.	Convolution	Type of data	Size (H×W×C)	Our system (ms)	ARM system (ms)	Intel system (ms)	Speed-up
1	Standard	Input Kernel	128×128×3 3×3×3	9.47	139.88	6.09	14.77×
2	Depth-wise	Input Kernel	64×64×32 3×3×32				
	Pointwise	Input Kernel	64×64×32 1×1×32				
3	Depth-wise	Input Kernel	64×64×64 3×3×64				
	Pointwise	Input Kernel	32×32×64 1×1×64				
4	Depth-wise	Input Kernel	32×32×128 3×3×128				
	Pointwise	Input Kernel	32×32×128 1×1×128				
5	Depth-wise	Input Kernel	32×32×128 3×3×128				
	Pointwise	Input Kernel	16×16×128 1×1×128				
6	Depth-wise	Input Kernel	16×16×256 3×3×256				
	Pointwise	Input Kernel	16×16×256 1×1×256				
7	Depth-wise	Input Kernel	16×16×256 3×3×256				
	Pointwise	Input Kernel	8×8×256 1×1×256				
8	Depth-wise	Input Kernel	8×8×512 3×3×512				
	Pointwise	Input Kernel	8×8×512 1×1×512				
9	Depth-wise	Input Kernel	8×8×512 3×3×512				
	Pointwise	Input Kernel	8×8×512 1×1×512				
10	Depth-wise	Input Kernel	8×8×512 3×3×512				
	Pointwise	Input Kernel	8×8×512 1×1×512				
11	Depth-wise	Input Kernel	8×8×512 3×3×512				
	Pointwise	Input Kernel	8×8×512 1×1×512				
12	Depth-wise	Input Kernel	8×8×512 3×3×512				
	Pointwise	Input Kernel	8×8×512 1×1×512				
13	Depth-wise	Input Kernel	4×4×1024 3×3×1024				
	Pointwise	Input Kernel	4×4×1024 1×1×1024				

VI. Conclusion

In this paper, we proposed an efficient FPGA-based convolution computing core that can process data in a pipeline model to improve performance. We develop a hardware accelerator system that utilizes our convolution core for the MobileNet model. Experiments with 256 128×128×3 images and 3×3×3 kernels are conducted to verify and evaluate the system. Experimental results show that we obtain a speed-up of 14.77× when compared to the 4-core ARM processor functioning at 1.2 GHz. Although our proposed system is worse than the 6-core Intel Core i7 processor, we achieve more energy efficiency than the Intel processor. It proves that our system best fits the edge computing system.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

C.P.-Q.: design, implement the system, and write the paper; T.N.T.: conduct experiments and proofread the paper; all authors had approved the final version.

Funding

This research is funded by Vietnam National University, Ho Chi Minh City (VNU-HCM) under grant number B2021-20-02.

Acknowledgment

We acknowledge Ho Chi Minh City University of Technology (HCMUT), VNU-HCM for supporting this study.

References

[1] G. Lacey, G. W. Taylor, and S. Areibi, "Deep learning on FPGAS: Past, present, and future," arxiv:1602.04283, 2016.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arxiv:1409.1556, 2015.

[3] R. Wu, X. Guo, J. Du, and J. Li, "Accelerating neural network inference on FPGA-based platforms — A survey," *Electronics*, vol. 10, no. 9, 2021.

[4] R. Williams, "What's next? The end of moore's law," *Computer Science Engineering*, vol. 19, no. 2, pp. 7–13, 2017.

[5] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," arXiv:1704.04861, 2017.

[6] S. Bouguezzi, H. Faiedh, and C. Souani, "Slim MobileNet: An enhanced deep convolutional neural network," in *Proc. the 2021 18th International Multi-Conference on Systems, Signals & Devices (SSD)*, Monastir, Tunisia, pp. 12–16, 2021.

[7] Avnet. Ultra96-v2 board. Arm-based, Xilinx Zynq UltraScale+ MPSoC development board based on the Linaro 96Boards Consumer Edition specification. [Online]. Available: https://www.avnet.com/wps/portal/us/products/new-product-introductions/npi/aes-ultra96-v2/

[8] C. Pham-Quoc, X. Q. Nguyen, and T. N. Thinh, "Towards an FPGA-targeted hardware/software co-design framework for CNN-based edge computing," *Mobile Network and Application*, vol. 27, pp. 2024–2035, 2022.

[9] K. Guo, S. Han, S. Yao, Y. Wang, Y. Xie, and H. Yang, "Software-hardware codesign for efficient neural network acceleration," *IEEE Micro*, vol. 37, no. 2, pp. 18–25, 2017.

[10] H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou, and L. Wang, "A high performance FPGA-based accelerator for large-scale convolutional neural networks," in *Proc. the International Conference on Field Programmable Logic and Applications (FPL)*, 2016, pp. 1–9.

[11] X. Lin, S. Yin, F. Tu, L. Liu, X. Li, and S. Wei, "LCP: A layer clusters paralleling mapping method for accelerating inception and residual networks on FPGA," in *Proc. the 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[12] Z. Liu, Y. Dou, J. Jiang, and J. Xu, "Automatic code generation of convolutional neural networks in FPGA implementation," in *Proc. the International Conference on Field-Programmable Technology (FPT)*, 2016, pp. 61–68.

[13] Y. Ma, Y. Cao, S. Vrudhula, and J. S. Seo, "Optimizing loop operation and dataflow in fpga acceleration of deep convolutional neural networks," in *Proc. the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA'17*, 2017, pp. 45–54.

[14] L. Yang, Z. He, and D. Fan, "A fully on-chip binarized convolutional neural network fpga implementation with accurate inference," in *Proc. the International Symposium on Low Power Electronics and Design, ISLPED'18*. ACM, New York, NY, USA, 2018.

[15] X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W. M. Hwu, and D. Chen, "Dnnbuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAS," in *Proc. the International Conference on Computer-Aided Design, ICCAD'18*. ACM, New York, NY, USA, 2018.

[16] A. Podili, C. Zhang, and V. Prasanna, "Fast and efficient implementation of convolutional neural networks on FPGA," in *Proc. IEEE 28th International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, 2017, pp. 11–18.

[17] M. Motamedi, P. Gysel, V. Akella, and S. Ghiasi, "Design space exploration of fpga-based deep convolutional neural networks," in *Proc. 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2016, pp. 575–580.

[18] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, "Binary neural networks: A survey," arXiv:2004.03333, 2020,

[19] Y. Yang, Q. Huang, B. Wu, T. Zhang, L. Ma, G. Gambardella, M. Blott, L. Lavagno, K. Vissers, J. Wawrzynek, and K. Keutzer, "Synetgy: Algorithm-hardware co-design for convnet accelerators on embedded FPGAS," in *Proc. the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA'19*, ACM, New York, NY, USA, 2019, pp. 23–32.

[20] M. Ghasemzadeh, M. Samragh, and F. Koushanfar, "Rebnet: Residual binarized neural network," in *Proc. IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018, pp. 57–64.

[21] D. J. M. Moss, E. Nurvitadhi, J. Sim, A. Mishra, D. Marr, S. Subhaschandra, and P. H. W. Leong, "High performance binary neural networks on the XEON+FPGA™ platform," in *Proc. the 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–4.

[22] H. Nakahara, T. Fujii, and S. Sato, "A fully connected layer elimination for a binarizec convolutional neural network on an FPGA," in *Proc. the 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–4.

[23] 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. the International Conference on Field-Programmable Technology (FPT)*, 2016, pp. 77–84.

[24] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proc. the ACM/ SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA'17*, ACM, New York, NY, USA, 2017, pp. 65–74.

[25] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "FP-BNN: Binarized neural network on FPGA," *Neurocomputing*, vol. 275, pp. 1072–1086, 2018.

[26] L. Jiao, C. Luo, W. Cao, X. Zhou, and L. Wang, "Accelerating low bit-width convolutional neural networks with embedded FPGA," in *Proc. the 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–4.

[27] A. Prost-Boucle, A. Bourge, F. Pétrot, H. Alemdar, N. Caldwell, and V. Leroy, "Scalable high-performance architecture for convolutional ternary neural networks on FPGA," in *Proc. the 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–7.

[28] S. Cao, C. Zhang, Z. Yao, W. Xiao, L. Nie, D. Zhan, Y. Liu, M. Wu, and L. Zhang, "Efficient and effective sparse LSTM on FPGA with bank-balanced sparsity," in *Proc. the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA'19*, Association for Computing Machinery, 2019, pp. 63–72.

[29] S. Kala, B. R. Jose, J. Mathew, and S. Nalesh, "High-performance CNN accelerator on FPGA using unified WINOGRAD-GEMM architecture," *IEEE Trans Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 12, pp. 2816–2828, 2019

[30] J. Wang, Q. Lou, X. Zhang, C. Zhu, Y. Lin, and D. Chen, "Design flow of accelerating hybrid extremely low bit-width neural network in embedded FPGA," in *Proc. the 28th International Conference on Field Programmable Logic and Applications (FPL)*, 2018, pp. 163–166.

[31] P.-Q. Cuong, J. Heisswolf, S. Werner, Z. Al-Ars, J. Becker, and K. Bertels, "Hybrid interconnect design for heterogeneous hardware accelerators," in *Proc. 2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2013.

[32] J. Wang, J. Lin, and Z. Wang, "Efficient hardware architectures for deep convolutional neural network," *IEEE Trans. Circ. Syst. I: Regular Papers*, vol. 65, no. 6, pp. 1941–1953, 2018.

[33] L. Lu, Y. Liang, Q. Xiao, and S. Yan, "Evaluating fast algorithms for convolutional neural networks on FPGAs," in *Proc. the 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017, pp. 101–108.

[34] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, J. Cong, "FP-DNN: An automated framework for mapping deep neural networks onto FPGAS with RTL-HLS hybrid templates," in *Proc. the IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017, pp. 152–159.

[35] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, "Going deeper with embedded fpga platform for convolutional neural network," in *Proc. the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA'16*, ACM, New York, NY, USA, 2016, pp. 26–35.

[36] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs," in *Proc. the 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[37] Y. Shen, M. Ferdman, and P. Milder, "Escher: A CNN accelerator with flexible buffering to minimize off-chip transfer," in *Proc. the IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017, pp. 93–100.

[38] 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 Trans Comput-Aided Des Integr. Circuits Syst.*, vol. 37, no. 1, pp. 35–47, 2017.

[39] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, *et al*., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. 33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, 2019.

[40] J. Su, J. Faraone, J. Liu, Y. Zhao, D. B. Thomas, P. H. Leong, and P. Y. Cheung, "Redundancy-reduced MobileNet acceleration on reconfigurable logic for ImageNet classification," in *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer: Cham, Switzerland, 2018.

[41] Y. Zhao, X. Gao, X. Guo, J. Liu, E. Wang, R. Mullins, P. Y. Cheung, G. Constantinides, and C. Z. Xu, "Automatic generation of multi-precision multi-arithmetic CNN accelerators for FPGAs," in *Proc. the 2019 International Conference on Field-Programmable Technology, ICFPT 2019*, Tianjin, China, 9–13 December 2019.

[42] B. Liu, D. Zou, L. Feng, S. Feng, P. Fu, and J, Li, "An FPGA-Based CNN accelerator integrating depth-wise separable convolution," *Electronics*, vol. 8, p. 281, 2019.

[43] I. Pérez and M. A. Figueroa, "Heterogeneous hardware accelerator for image classification in embedded systems," *Sensors*, vol. 21, 2637, 2021.

[44] S. Bouguezzi, H. B. Fredj, T. Belabed, C. Valderrama, H. Faiedh, and C. Souani, "An efficient FPGA-based convolutional neural network for classification: Ad-MobileNet," *Electronics*, vol. 10, no. 18, 2272, 2021.

Copyright © 2023 by the authors. This is an open access article distributed under the Creative Commons Attribution License (CC BY-NC-ND 4.0), which permits use, distribution and reproduction in any medium, provided that the article is properly cited, the use is non-commercial and no modifications or adaptations are made.