2021 31st International Conference on Field-Programmable Logic and Applications (FPL) | 978-1-6654-3759-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/FPL53798.2021.00061

Increasing Flexibility of FPGA-based CNN Accelerators with Dynamic Partial Reconfiguration

Hasan Irmak
University of Twente
Enschede, The Netherlands
h.irmak@utwente.nl

Daniel Ziener
Technische Universität Ilmenau
Ilmenau, Germany
daniel.ziener@tu-ilmenau.de

Nikolaos Alachiotis
University of Twente
Enschede, The Netherlands
n.alachiotis@utwente.nl

***Abstract*—Convolutional Neural Networks (CNN) are widely used for image classification and have achieved significantly accurate performance in the last decade. However, they require computationally intensive operations for embedded applications. In recent years, FPGA-based CNN accelerators have been proposed to improve energy efficiency and throughput. While dynamic partial reconfiguration (DPR) is increasingly used in CNN accelerators, the performance of dynamically reconfigurable accelerators is usually lower than the performance of pure static FPGA designs. This work presents a dynamically reconfigurable CNN accelerator architecture that does not sacrifice throughput performance or classification accuracy. The proposed accelerator is composed of reconfigurable macroblocks and dynamically utilizes the device resources according to model parameters. Moreover, we devise a novel approach, to the best of our knowledge, to hide the computations of the pooling layers inside the convolutional layers, thereby further improving throughput. Using the proposed architecture and DPR, different CNN architectures can be realized on the same FPGA with optimized throughput and accuracy. The proposed architecture is evaluated by implementing two different LeNet CNN models trained by different datasets and classifying different classes. Experimental results show that the implemented design achieves higher throughput than current LeNet FPGA accelerators.**

***Index Terms*—FPGA, CNN, Partial Reconfiguration**

I. Introduction

Convolutional Neural Networks (CNNs) have been extensively used in computer vision applications such as object recognition, autonomous driving, and semantic segmentation [1], [2]. Their state-of-the-art performance in classification accuracy makes CNNs very attractive. Due to the huge computational load of CNNs, it is difficult to implement energy efficient designs on CPU and GPU platforms [3]. In real-time CNN implementations, *Field Programmable Gate Arrays* (FPGA) and *Application Specific Integrated Circuit* (ASIC) are very good alternatives for accelerating CNNs. Since ASICs are custom designed integrated circuits and can only be optimized for accelerating particular CNNs, they lack scalability and flexibility. FPGAs can be used in CNN accelerator designs effectively, since they are more energy efficient than CPUs and GPUs, and more flexible than ASICs [4].

The *Dynamic Partial Reconfiguration* (DPR) capability of modern FPGAs allows to dynamically modify the hardware structure of the implemented circuit by loading partial bit files while the remaining logic continues to operate [5]. DPR provides a very high level of flexibility to realize adaptive hardware designs. In addition, sharing resources between mutually exclusive computations reduces the resource utilization. This leads to the dynamic FPGA design consuming less power than the purely static design. Although the major FPGA vendors have been supporting DPR for two decades already, it has remained mostly underutilized due to the limited use cases and shortage of benefits over static designs [6]. CNN accelerators represent a suitable application for exploiting DPR, since different CNN architectures can be implemented with partial modifications.

In CNN accelerator applications, hardware optimizations play a key role in performance. To design a high-performance accelerator, computations and memory operations should be considered together. Optimizing the bit sizes can reduce computations and memory requirements, and help to exploit the trade-off between accuracy and latency [7]. Moreover, pipelining and parallelism strategies need to be consistent with the hardware architecture, bit sizes and available resources. As a result, using DPR and hardware optimizations in the FPGA design, flexible and efficient CNN accelerators can be realized. To this end, the contribution of this work is two-fold:

- We devise a flexible hardware design and computer architecture that allows to easily modify the CNN accelerator (i.e., insert/delete/update the structure and precision of each layer) at run time using dynamic partial reconfiguration. The proposed architecture can adapt to different networks and, therefore, also to different applications using DPR without any degradation in accuracy or throughput.
- We introduce a novel method to perform feature extraction that allows to hide the computations of a pooling layer behind the computations of a convolutional layer. This allows to further exploit pipelining and other optimizations for better performance.

II. Background & Literature Review

CNNs belong to a class of deep neural networks that can be used in image processing and object recognition as they can extract and classify features in an image. It consists of convolutional, pooling, and *fully connected* (FC) layers. The convolutional layers implement convolutions to extract the features in an image, while the pooling layers reduce the data size progressively, thereby decreasing the number of parameters and the sensitivity to the location of the features. Thereafter, the feature maps are flattened and fed to FC layers for classification. An FC layer is a feed forward neural network that consists of one or more hidden layers. After the hidden layers, an output layer produces a probability per class/object.

In recent years, CNN architectures have been increasingly more accurate and complex [8]–[10]. Designing more robust and accurate CNN architectures results in higher computational complexity and power consumption. FPGAs can over-

978-1-6654-3759-2/21/$31.00 ©2021 IEEE
DOI 10.1109/FPL53798.2021.00061

come these limitations due to their custom parallel processing capabilities and flexibility. Various CNN implementations on FPGA have been reported in literature [11]–[13], focusing on different aspects, e.g., the optimization of only the convolutional layers [14], [15] or the overall accelerator throughput [16]. There are also SW/HW co-design solutions that exploit the aggregate power of both an embedded processor and the programmable logic [17], [18]. Some [19], [20] present resource-intensive designs and achieve high throughput disregarding power consumption, while others implement *binary neural networks* (BNNs) that achieve high power efficiency at the cost of reduced accuracy using binary weights/biases [21].

Various studies [22]–[24] investigate the DPR feature of FPGAs, and design resource-efficient and low-power CNN accelerators. A hardware design with DPR is divided into two parts, the static part and the dynamic part. The static part is configured at power up and contains the unmodified part of the design. Dynamic parts are the reconfigurable regions having a set of physical resources on the FPGA. These collections of resources inside one or more rectangular regions are called *Pblocks* and can be reconfigured with different partial bitstreams without changing the static part, which continues to operate. For DPR, *Internal Configuration Access Port* (ICAP) is offered by Xilinx FPGAs and *Processor Configuration Access Port* (PCAP) is also provided to reconfigure FPGA from processor side in Zynq FPGAs [5].

Youssef et al. [7] use DPR to adjust the power consumption according to the power level of the battery. If the power is low, a lower accuracy design (i.e., smaller bit sizes) is loaded. Other works use DPR to change the kernel size dynamically and execute convolutions with different kernel sizes [25], [26]. There are also CNN accelerators utilizing DPR to reconfigure the processing engines in the hardware, which operate in a cascaded manner [27], [28]. Moreover, in some works, each layer of the CNN accelerator is designed as a separate partial bit file and every layer is processed after the related partial bit file is loaded [29], [30]. The resource usage is considerably minimized, but the DPR overhead significantly decreases the throughput of the CNN accelerator. Using our proposed hardware architecture, different CNN accelerators can be realized using DPR without sacrificing throughput or accuracy of each CNN accelerator, because DPR is employed for switching between the different CNN accelerators instead of switching between layers within the same CNN.

III. System Design

A. Accelerator Architecture

In our proposed accelerator, a basic processing element is referred to as a *MacroBlock*. A *MacroBlock* is a generic construct with three different interfaces: a data interface (i.e., AXI stream master/slave) is used for high-speed data transfer, a memory interface (i.e., AXI4) is used for updating the weights and biases, and a GPIO interface is employed for I/O operations. The *MacroBlock* is a partially reconfigurable region, thus it has an additional reconfiguration interface and it can be used to implement any of the CNN layers.

The proposed CNN accelerator architecture, shown in Figure 1, is a cascaded connection of a number of *MacroBlocks*, depending on the CNN depth. Every *MacroBlock* can be interconnected to any other *MacroBlock* or I/O port of the

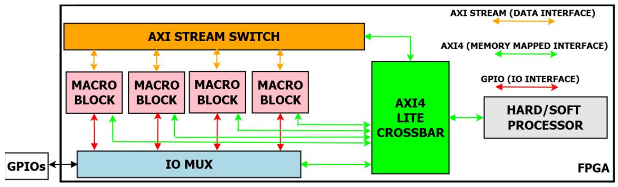

Fig. 1. Proposed hardware architecture with 4 macro blocks that can be dynamically interconnected.

FPGA using internal bus switches, and these interconnections can be configured at runtime. This allows to dynamically change the CNN network size by adding or removing layers, as well as by replacing the structure of a layer with another structure to realize different types of CNN networks. In the updated structure, the weights and biases can be updated using the memory mapped interface of the *MacroBlocks*.

Furthermore, the stream switch in the proposed architecture allows *MacroBlocks* to be clocked at different frequencies. When the sender's data transfer speed is higher than the receiver's, the receiver (i.e., slave) can block the sender (i.e., master) to decrease the data transfer rate using the back pressure property of the AXI stream interface [31]. Furthermore, since each *MacroBlock* can be run at a different frequency, clock frequencies can be optimized according to the frequency margin of the related *MacroBlock*. Thus, if a *MacroBlock* needs more clock cycles for execution, it is possible to run it at higher frequencies to preserve the overall throughput.

B. Design Optimizations

a) Pipeline: A typical CNN architecture consists of sequential layers where the output of one layer is input to the next. Thus, several layers are computed concurrently in a pipelined datapath. The pipeline performance is limited by the *MacroBlock* with the highest initiation interval (the number of cycles that must elapse between issuing two operations). For the maximum throughput, the available hardware resources are allocated to adjust the execution times approximately equal for each layer. In other words, for each layer, the degree of the parallelism is optimized for the maximization of the overall throughput, not to maximize the throughput of a specific layer.

b) Quantization: An important factor in a CNN accelerator is the precision of bit sizes for activations, weights, and biases. Calculations that implement floating-point arithmetic consume a considerable amount of hardware resources, yet the performance difference between fixed-point and floating-point arithmetic implementations in terms of accuracy can be negligible. This can be achieved only if optimum bit sizes are used in the fixed-point design. Thus, the network is first developed using *Python Tensorflow* and the optimum bit sizes are determined based on the accuracy drop, as compared with the accuracy achieved with single-precision floating-point arithmetic. We use 8-bit weights, 16-bit activations, and 32-bit biases, with an accuracy drop of less than a 0.01% in comparison with the single-precision floating-point implementation.

c) Merging of the pooling and convolution layers: The most time-consuming operation in a CNN is the convolution. To this end, we devise a novel convolution operation that hides the computations of pooling layers within the convolution layer, thereby eliminating the delay of the pooling layers. This optimization is directly applicable to the most popular CNNs,

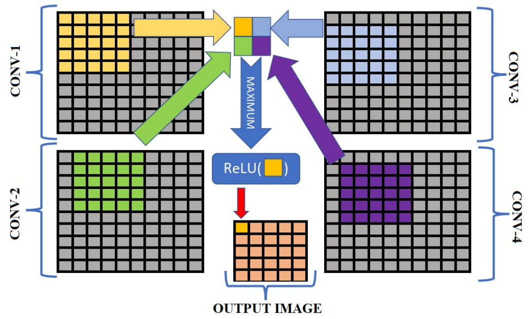

Fig. 2. Block diagram of the proposed convolution

such as LeNet, AlexNet, VGGNet, because all of them apply pooling to the output of the convolutional layers to decrease sensitivity to the location of the features.

The output pixel of a convolution layer with *a()* input and *f()* activation function followed by a maximum pooling layer with 2×2 kernel, and downsampling by 2 is given by Equation 1.

$$output(\frac{i}{2}, \frac{j}{2}) = max(f(Conv(a(i,j)), f(Conv(a(i,j+1)), f(Conv(a(i+1,j))), f(Conv(a(i+1,j+1)))) \quad (1)$$

In Equation 1, if *f()* is a monotonically non-decreasing function, then it can be applied after the pooling layer. The most popular activation functions, such as ReLU, tanh, sigmoid are all monotonically non-decreasing functions [32]. Thus, the order of applying the activation operation can be replaced with the pooling and can be written as in Equation 2.

$$output(\frac{i}{2}, \frac{j}{2}) = f(max(Conv(a(i,j)), Conv(a(i,j+1)), Conv(a(i+1,j)), Conv(a(i+1,j+1)))) \quad (2)$$

Using Equation 2 has two advantages: a) the activation is applied to a smaller number of samples, and b) the pooling data is already in the buffers, thus no additional read/write operations are required for the pooling operation. This way the execution time of the pooling layer is practically eliminated. The corresponding hardware architecture for Equation 2 is shown in Figure 2.

Optimization of an FPGA design requires a multi-dimensional approach considering the architecture with available resources in the hardware. In Figure 2, if there is enough number of resources on the FPGA, all processing can be done in parallel. In other words, using 100 DSPs (i.e., 25 DSPs per convolution), the Equation 2 can be processed in one clock cycle using the proposed hardware architecture in Figure 2. Based on the number of resources assigned to any convolution layer, the number of clock cycles for this operation can be increased. For instance, if you dedicate 50 DSPs for any convolution block, the execution time will become 2 clock cycles for the computation in Figure 2. In the design of convolutional layers, to complete the proposed convolution in one clock cycle, it is required to read weights and biases in a single clock cycle. Therefore, all the internal memories (i.e., BRAMs) storing weights and biases are 256 bits wide to read them in a single transaction. In each address of the BRAM, the weights and biases are stored in a concatenated manner.

d) Optimizations in fully connected and output layers: The pseudo code for the conventional computation of an FC layer is shown in Algorithm 1. As explained in the beginning of this section, to preserve the overall throughput, the execution time of FC layers should be lower than maximum execution time of any convolutional layer. Thus, based on the time budget and available resources, the multiplications can be parallelized as much as possible. This can be achieved by unrolling the inner for loop. The amount of parallelism is limited by the available resources for the specific FC layer. Moreover, the summation with the bias term can be omitted by initializing the accumulator with the bias value. Algorithm 2 describes the proposed optimization that reduces the execution times of the FC layers.

Algorithm 1 Conventional Computation of FC Layers

```
procedure FCLAYER(X, W, bias)
    for i from 1 to outputNodeSize do
        accumulator = 0
        for j from 1 to inputNodeSize do
            accumulator = accumulator + W[i][j]X[j]
        end for
        Y[i] = accumulator + bias[i]
    end for
end procedure
```

Algorithm 2 Proposed Computation of FC Layers

```
procedure FCLAYER(X, W, bias)
    for i from 1 to outputNodeSize do
        UNROLL_LOOP
        accumulator = bias[i]
        for j from 1 to inputNodeSize do
            accumulator = accumulator + W[i][j]X[j]
        end for
        Y[i] = accumulator
    end for
end procedure
```

The output layer of a conventional CNN relies on a softmax function to find the probability distribution over the predicted output classes [33] in order to select the class with the highest probability as the classification result. The softmax function, however, consists of complex exponentials and divisions which are highly compute- and resource-demanding operations. While our Python CNN implementation employs the softmax function in the output layer, our FPGA design does not implement the softmax function and directly selects the highest-value class, thereby producing the same classification result as if the softmax function was used. In addition, since the values of the output layer are calculated in a sequential manner, no additional clock cycles are required for finding the maximum. In other words, it is hidden in the computation of the output layer.

C. Dynamic Partial Reconfiguration

In addition to the aforementioned optimizations, DPR is used to reduce resource utilization and power consumption for the implementation of more than one CNN accelerators. The underlying idea is that, instead of performing different accelerators together, implementing them separately increases the accuracy greatly. It is clear that training the CNNs separately, instead of training together, gives more accurate classifications, since feature detection is much easier in the former case.

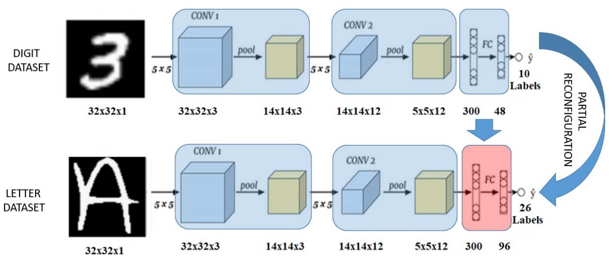

Fig. 3. Implementation of different CNN accelerators

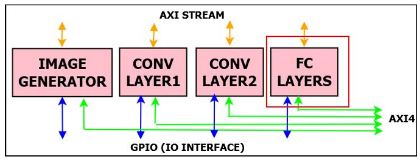

Fig. 4. *MacroBlocks* used in the CNN accelerator. The red square indicates the partially reconfigurable *MacroBlock* that can be dynamically exchanged to implement FC layers for digit/letter classification.

In the proposed architecture, each *MacroBlock* can be a partially reconfigurable region (*Pblock*) to realize different CNN accelerators. To reconfigure a *MacroBlock*, the PCAP interface is used. The *Pblocks* are determined according to the resource utilization of the *MacroBlocks*. To avoid routing congestion, only the required *MacroBlocks* are mapped/assigned to *Pblocks*. In other words, when there is no need for DPR of a particular *MacroBlock*, the related logic is placed to any part of the FPGA except for the device area of the *Pblocks*. DPR introduces significant flexibility to the system architecture. It allows realizing more than one CNN architectures using the same FPGA implementation by only updating a small fraction of the FPGA resources with a dedicated partial bitstream. The programmable *MacroBlock* interconnection allows to implement other CNN architectures with a smaller or larger number of layers. Hence, a totally different CNN architecture can be realized with the proposed hardware architecture. Moreover, a different CNN architecture only requires the implementation of new *MacroBlocks*, while the placement and routing of the static design do not need to change, thereby lowering implementation times for new CNN architectures. If there is high variability in layer sizes among different CNNs, multiple DPR regions with different sizes can also be used.

D. Case Study

For the proof of concept, two different LeNet CNN architectures have been developed for letter classification and digit classification. LeNet was the first CNN architecture and promoted the development of deep learning [34]. To switch the required accelerator, the weights and biases of the convolution layers are updated and the FC layers are dynamically partially reconfigured with the corresponding *MacroBlocks*. The block diagram of an example design is shown in Figure 3.

In the example design, the CNN input is a 32 x 32 grayscale image and the output is the classification result. Both CNN accelerators consist of 2 convolutional layers, 2 max pooling layers, a hidden FC layer and an output layer. The size of the hidden layer and output layer are different for the accelerators. Except for the last layer, ReLU activation function is used in the convolutional and hidden layers. In convolutional layers, the convolutional kernel and pooling kernel are selected as 5 x 5 and 2 x 2 for its better performance as compared to other size kernels. In the pooling layers, down sampling factor is selected as 2. After the convolutional layers, data is flattened and fed to the FC layers. In letter recognizer CNN accelerator, 48 nodes and 10 nodes are used for hidden and output layer, respectively. For the letter recognizer, 96 nodes and 26 nodes are used for hidden and output layer, respectively. The CNNs are trained and tested using MNIST handwritten digit dataset and EMNIST handwritten letter dataset separately [35], [36].

IV. IMPLEMENTATION & EVALUATION

The CNNs in Figure 3 are implemented using the hardware architecture depicted in Figure 1 and *MacroBlocks* given in Figure 4. The hardware design is implemented using Xilinx Vivado 2020.2. The *MacroBlocks* are designed in Vitis HLS, which relies on a *high-level synthesis* (HLS) approach to transform a C, C++, or System C code into a register transfer level implementation for Xilinx FPGAs. Using *pragmas* in the software code, different levels of parallelization and, therefore, different hardware structures can be generated [37]. The Vitis HLS allows a designer to develop and verify the designs faster than the traditional hardware description languages. For the DPR, Vivado's partial reconfiguration flow is used.

The design is first verified on the simulation environment of Vivado. Then, it is implemented and tested on a Zedboard FPGA board [38]. The board is equipped with a Xilinx Zynq 7020 FPGA. The input images are loaded from serial port of the Zedboard and the result is printed on the LEDs. In addition, the internal data transfers between *MacroBlocks* are tested on the system using Vivado Hardware Manager. The proposed architecture operates at 100 MHz clock frequency.

In the hardware architecture, since the only difference between the digit and letter CNN accelerators is in the FC layers, only the *MacroBlock* that implements the FC layers is reconfigurable as denoted by the red square in Figure 4. For another application, other *MacroBlocks* could be reconfigurable as well. Note that weights and biases also differ between the two CNN accelerators but they are updated through the memory-mapped interface of the *MacroBlocks*. The layout of the FPGA implementation of the CNN accelerator and the reconfigurable *Pblock* for the *MacroBlock* of the FC layer is shown in Figure 5. The size and location of the *Pblock* is determined according to the resource utilization of the related *MacroBlock* of the CNN accelerators.

Because multiplications dominate computation, the available DSP slices on the target device are fully utilized in the proposed CNN accelerator to maximize performance. These resources are shared between *MacroBlocks* to be able to make the execution times of them nearly equal. The resource usage and processing times of the *MacroBlocks* are given in Table I. As can be observed in the table, the execution time of the FC layer in the letter CNN accelerator is higher than the other *MacroBlocks* since the system is optimized to maximize the throughput of the digit CNN accelerator.

Table II provides implementation results for three different accelerators: Hybrid CNN, letter CNN, and digit CNN. The accelerators employ all the DSP slices on the device (i.e., 220 DSPs on the Zedboard) to maximize throughput. The digit

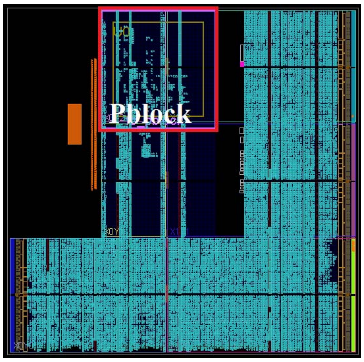

Fig. 5. Layout of the implemented CNN accelerator. The red square *Pblock* corresponds to the partial reconfigurable area.

TABLE I
RESOURCE USAGE AND PROCESSING TIME OF DIFFERENT LAYERS (L)

	BRAM	DSP	LUT	Clock Cycle	Processing Time
Conv L 1 + Max Pool L 1	5	50	7372	2022	20.22 μs
Conv L 2 + Max Pool L 2	22	150	14761	2069	20.69 μs
FC Layer (Digit)	9	20	639	2128	21.28 μs
FC Layer (Letter)	20	20	601	4401	44.01 μs

TABLE II
COMPARISON OF PROPOSED CNN ACCELERATOR WITH HYBRID CNN ACCELERATOR IN TERMS OF RESOURCE USAGE AND ACCURACY

	BRAM	DSP	LUT	Accuracy	Execution Time
Hybrid CNN (Static)	55	220	26897	87.53	87.55 μs
Letter CNN (DPR)	55	220	26807	92.45	84.92 μs
Digit CNN (DPR)	44	220	26845	98.88	62.19 μs

TABLE III
PERFORMANCE COMPARISON OF DIFFERENT LENET CNN ACCELERATORS

	BRAM	DSP	LUT	Accuracy	Processing Time
Shi [39]	54	204	25276	99.11	170 μs
Youssef [7]	9	N/A	19141	98.12	270 μs
Li [28]	619	916	9071	98.16	490 μs
This work	44	220	26845	98.88	62 μs

CNN and the letter CNN use the same static implementation but require a different *MacroBlock* for the FC layers. The hybrid CNN deploys a mixed CNN model that has been trained with both letter and digit datasets without using DPR. It has the same number of nodes with the letter CNN in the hidden layer (i.e., 96 nodes), while the output layer has 36 nodes (i.e., 26 for letters and 10 for digits). The same optimizations (Section III) have been applied to all three CNN accelerators, and the resource utilization of the hybrid CNN is similar to the letter CNN. However, the main differences between the hybrid CNN and the others are in terms of accuracy and throughput. Accuracy drops by 5% and 10% in comparison with the letter and the digit CNNs, respectively. Similarly, throughput also decreases in the hybrid CNN in comparison with dedicated digit and letter CNN accelerators using DPR. Therefore, DPR allows to explore the trade-offs between resource usage, throughput, and accuracy.

To conduct a fair performance evaluation of the proposed CNN accelerator, we compare performance with 3 state-of-the-art CNN accelerators that use the same LeNet architecture (i.e., 2 convolutional layers, 2 pooling layers, 1 hidden layer), trained with the MNIST dataset. The first CNN accelerator is a static design [39] and the ZCU102 evaluation board is used for the implementation. For each convolutional layer and pooling layer, separate accelerators are designed. The second is a DPR design and according to the energy level of the power source, the processing system uploads the required partial bitstream at run time using ICAP [7]. The last work is using cascaded connections of processing engines designed to compute the convolution [28]. Pipelining and tiling techniques are used to improve the performance of that design. The performance comparison with these studies is given in Table III. As can be seen from Table III, our proposed accelerator has the shortest processing time to complete the classification of an image, due to the hardware optimizations discussed in Section III that allow for a considerable throughput improvement in comparison with the previous accelerators. Moreover, accuracy is similar to the other designs. The LUT usage is slightly higher than the other designs because of the implementation of the flexible hardware architecture in Figure 1.

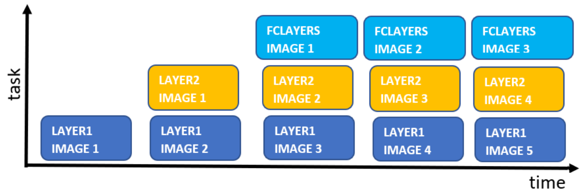

Fig. 6. Pipelined processing of the proposed CNN accelerator.

In the proposed architecture, all data is processed in a pipelined manner. Although the total processing time is 62 μs for the letter CNN accelerator; the accelerator can be fed with a higher frame rate. Every *MacroBlock* can process new data after finishing its task, i.e., there is no need to wait until the end of the overall processing of one image to proceed with the next. As shown in Figure 6, the overall frame rate depends on the processing time of the *MacroBlock* with the largest delay. Therefore, using the pipelining in Figure 6, the proposed accelerator achieves frame rates of up to 45K images/sec in digit classification which is 7x higher than the state-of-the-art LeNet CNN accelerators given in Table III. Lastly, in the proposed architecture, the switch time between CNN accelerators (i.e., DPR time) is 9.1 ms since the PCAP throughput is 145 MB/s and our partial bit file size is 1.32 MB [40].

V. CONCLUSION

In this work, a high performance and flexible CNN accelerator architecture is proposed and implemented on a FPGA platform. By optimizing the operations in the CNN accelerator, throughput is improved without any degradation of accuracy. In addition, we use DPR to realize different CNN accelerators without changing the entire FPGA implementation. The method is evaluated by implementing a single static design for two different LeNet CNN accelerators on a Zedboard, and employing DPR to switch between them. Experimental results showed that the proposed LeNet CNN accelerator achieves higher throughput with moderate DSP and BRAM usage in comparison with previous LeNet implementations on FPGAs.

ACKNOWLEDGMENT

This research was supported by The Scientific and Technological Research Council of Turkey (TUBITAK).

References

[1] G. Prabhakar, B. Kailath, S. Natarajan, and R. Kumar, "Obstacle detection and classification using deep learning for tracking in high-speed autonomous driving," in *2017 IEEE region 10 symposium (TENSYMP)*, pp. 1–6, IEEE, 2017.

[2] J. Perry and A. Fernandez, "Minenet: A dilated cnn for semantic segmentation of eye features," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pp. 0–0, 2019.

[3] M. Qasaimeh, K. Denolf, J. Lo, K. Vissers, J. Zambreno, and P. H. Jones, "Comparing energy efficiency of cpu, gpu and fpga implementations for vision kernels," in *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*, pp. 1–8, IEEE, 2019.

[4] "Comparing hardware for artificial intelligence: Fpgas vs. gpus vs. asics." http://lreese.dotsenkoweb.com/2019/03/30/comparing-hardware-for-artificial-intelligence-fpgas-vs-gpus-vs-asics/#. Accessed: 2021-03-25.

[5] X. UG702, "Partial reconfiguration user guide," *Xilinx Pg*, vol. 96, 2017.

[6] M. Nguyen, R. Tamburo, S. Narasimhan, and J. C. Hoe, "Quantifying the benefits of dynamic partial reconfiguration for embedded vision applications," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 129–135, IEEE, 2019.

[7] E. Youssef, H. A. Elsemary, M. A. El-Moursy, A. Khattab, and H. Mostafa, "Energy adaptive convolution neural network using dynamic partial reconfiguration," in *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 325–328, IEEE, 2020.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[11] A. Shawahna, S. M. Sait, and A. El-Maleh, "Fpga-based accelerators of deep learning networks for learning and classification: A review," *IEEE Access*, vol. 7, pp. 7823–7859, 2019.

[12] K. Abdelouahab, M. Pelcat, J. Serot, and F. Berry, "Accelerating cnn inference on fpgas: A survey," *arXiv preprint arXiv:1806.01683*, 2018.

[13] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A survey of fpga-based neural network accelerator," *arXiv preprint arXiv:1712.08934*, 2017.

[14] Z. Liu, Y. Dou, J. Jiang, J. Xu, S. Li, Y. Zhou, and Y. Xu, "Throughput-optimized fpga accelerator for deep convolutional neural networks," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 10, no. 3, pp. 1–23, 2017.

[15] D. Wu, Y. Zhang, X. Jia, L. Tian, T. Li, L. Sui, D. Xie, and Y. Shan, "A high-performance cnn processor based on fpga for mobilenets," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 136–143, 2019.

[16] A. Kyriakos, V. Kitsakis, A. Louropoulos, E.-A. Papatheofanous, I. Patronas, and D. Reisis, "High performance accelerator for cnn applications," in *2019 29th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pp. 135–140, IEEE, 2019.

[17] D. Rongshi and T. Yongming, "Accelerator implementation of lenet-5 convolution neural network based on fpga with hls," in *2019 3rd International Conference on Circuits, System and Simulation (ICCSS)*, pp. 64–67, IEEE, 2019.

[18] E. González, W. D. Villamizar Luna, and C. A. Fajardo Ariza, "A hardware accelerator for the inference of a convolutional neural network," *Ciencia E Ingenieria Neogranadina*, vol. 30, no. 1, pp. 107–116, 2020.

[19] L. Xie, X. Fan, W. Cao, and L. Wang, "High throughput cnn accelerator design based on fpga," in *2018 International Conference on Field-Programmable Technology (FPT)*, pp. 274–277, IEEE, 2018.

[20] L. Gong, C. Wang, X. Li, H. Chen, and X. Zhou, "Maloc: A fully pipelined fpga accelerator for convolutional neural networks with all layers mapped on chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2601–2612, 2018.

[21] T. Simons and D.-J. Lee, "A review of binarized neural networks," *Electronics*, vol. 8, no. 6, p. 661, 2019.

[22] P. Skrimponis, E. Pissadakis, N. Alachiotis, and D. N. Pnevmatikatos, "Accelerating binarized convolutional neural networks with dynamic partial reconfiguration on disaggregated fpgas.," in *PARCO*, pp. 691–700, 2019.

[23] C. Xue, S. Cao, R. Jiang, and H. Yang, "A reconfigurable pipelined architecture for convolutional neural network acceleration," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2018.

[24] A. Ansari, K. Gunnam, and T. Ogunfunmi, "An efficient reconfigurable hardware accelerator for convolutional neural networks," in *2017 51st Asilomar Conference on Signals, Systems, and Computers*, pp. 1337–1341, IEEE, 2017.

[25] P. Meloni, G. Deriu, F. Conti, I. Loi, L. Raffo, and L. Benini, "A high-efficiency runtime reconfigurable ip for cnn acceleration on a mid-range all-programmable soc," in *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pp. 1–8, IEEE, 2016.

[26] C. Yang, Y. Wang, H. Zhang, X. Wang, and L. Geng, "A reconfigurable cnn accelerator using tile-by-tile computing and dynamic adaptive data truncation," in *2019 IEEE International Conference on Integrated Circuits, Technologies and Applications (ICTA)*, pp. 73–74, IEEE, 2019.

[27] B. B. Seyoum, M. Pagani, A. Biondi, S. Balleri, and G. Buttazzo, "Spatio-temporal optimization of deep neural networks for reconfigurable fpga socs," *IEEE Transactions on Computers*, 2020.

[28] Z. Li, L. Wang, S. Guo, Y. Deng, Q. Dou, H. Zhou, and W. Lu, "Laius: An 8-bit fixed-point cnn hardware inference engine," in *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*, pp. 143–150, IEEE, 2017.

[29] M. Farhadi, M. Ghasemi, and Y. Yang, "A novel design of adaptive and hierarchical convolutional neural networks using partial reconfiguration on fpga," in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–7, IEEE, 2019.

[30] F. Kästner, B. Janßen, F. Kautz, M. Hübner, and G. Corradi, "Hardware/software codesign for convolutional neural networks exploiting dynamic partial reconfiguration on pynq," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 154–161, IEEE, 2018.

[31] A. Xilinx, "Reference guide, ug761 (v13. 1)," *URL http://www. xilinx. com/support/documentation/ip documentation/ug761 axi reference guide. pdf*, 2011.

[32] Y. Wang, Y. Li, Y. Song, and X. Rong, "The influence of the activation function in a convolution neural network model of facial expression recognition," *Applied Sciences*, vol. 10, no. 5, p. 1897, 2020.

[33] R. Hu, B. Tian, S. Yin, and S. Wei, "Efficient hardware architecture of softmax layer in deep neural network," in *2018 IEEE 23rd International Conference on Digital Signal Processing (DSP)*, pp. 1–5, IEEE, 2018.

[34] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[35] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[36] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, "Emnist: Extending mnist to handwritten letters," in *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2921–2926, IEEE, 2017.

[37] "Vitis high-level synthesis." https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html. Accessed: 2021-03-25.

[38] "Zedboard." https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/zedboard/. Accessed: 2021-03-25.

[39] Y. Shi, T. Gan, and S. Jiang, "Design of parallel acceleration method of convolutional neural network based on fpga," in *2020 IEEE 5th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)*, pp. 133–137, IEEE, 2020.

[40] Xilinx, "Zynq-7000 all programmable soc: Technical reference manual. ug585, v1. 8.1," 2014.