Convolution Processing Unit Featuring Adaptive Precision using Dynamic Reconfiguration

David Cain[2], Omar Eldash[1], Kasem Khalil[1], and Magdy Bayoumi[1,2]
[1]The Center for Advanced Computer Studies
[2]Department of Electrical and Computer Engineering
University of Louisiana at Lafayette, Louisiana, USA
Emails: {c00043561, oke1206, kmk8148, mab0778}@louisiana.edu

***Abstract*—As the demand of low latency machine learning IoT devices has rapidly expanded in the past decade for commercial and industrial applications, cloud computing models have relied on high-speed connectivity which many devices do not have access to. Cloud based solutions are also becoming less attractive for inference solutions with arising privacy and security concerns. Transitioning computation towards edge devices is becoming essential and finding ways to make such computer vision workloads operational on constrained IPs is a key element in this work. As edge devices are limited in processing resources, relying on reconfigurable hardware such as FPGAs will offer an embedded platform to dynamically shape the IPs and cores as needed for each specific application working on it. Convolutional Neural Networks and alike computer vision techniques rely heavily on computation and matrix based operations, this paper proposes a dedicated convolution processing element. The proposed a convolution processing element is implemented on a Pynq-Z2 FPGA that supports Dynamic Partial Reconfiguration between integer point convolution operations of 8, 16, and 32 bit precision. The use of Dynamic Partial Reconfiguration at a processing element level enables the design to be reconfigured 800× more quickly than traditional methods while minimally increasing system overhead.**

***Index Terms*—Field-Programmable Gate Array, Convolutional Neural Network, Reconfigurable Architecture**

I. Introduction

As the computer vision domain transitions into the embedded systems level, unique IoT edge devices are being designed with image and video data workloads in mind. Machine learning workloads are commonly offloaded to cloud computing servers to process data and then return the extracted details to the edge device [1]. This offloading method is required as machine learning quickly exhausts the constrained computing capabilities of the edge devices with the use of Deep Learning (DL) and especially Convolutional Neural Networks (CNNs). As IoT systems shift to decentralize these DL processes from cloud servers with racks of dedicated hardware to edge devices with much lower processing capabilities, such processes must be optimized for constrained computing while retaining high performance and accuracy [2]. Computer vision workloads have been trained and executed over a range of devices from powerful and commercial CPUs, GPUs, and specialized hardware accelerators [1]. In this direction, Field-Programmable Gate Arrays (FPGAs) are gaining a great momentum. FPGAs are reconfigurable hardware computing devices which are commonly used for rapid prototyping of hardware solutions and adding hardware flexibility. FPGAs have also been shown to have up to 22× increased power efficiency compared to a traditional CPU when processing images [3]. This flexibility, customization, and increasing power efficiencies is what makes FPGAs attractive for accelerating CNNs. Several works as in [4]–[10] have been published on hardware accelerating the CNN model by use of an FPGA.

FPGAs have supported Dynamic Partial Reconfiguration (DPR) for some time and it is a great feature for creating even more flexible and dynamic solutions. DPR allows for FPGAs to be partially reconfigured for either a change in behavior, functionality, or a trade-off in performance metrics [4]. DPR allows system processing blocks to be interchanged during runtime which aids by effectively reducing hardware overhead for configurable or parallel processing pipelines that facilitate the flexible requirements of a CNN model. Traditionally DPR is used to change a CNN pipeline entirely, which may result in system downtime with long configuration periods. The proposed design applies DPR at a processing element level to change convolution bit precision. As these CNN pipeline consists of many processing elements they may be strategically queued for reconfiguration which mitigates system downtime.

The paper is organized as follows: Section II covers a background to frame the context of Convolutional Processing Element (CPE) architecture and related contributions by introducing Deep Learning and CNN layers as well as reconfigurable hardware. Section III presents an overview of previous work in the areas of convolutions optimization and CNN acceleration generally. Section IV describes the architecture of the proposed Convolution Processing Element in detail. Section V discusses results of hardware implementation and comparison analysis with previous work on convolution designs, followed by a conclusion in Section VI.

II. Background Overview

In this section, main underlying concepts are discussed to lead to the contributions of this paper.

A. Convolutional Neural Networks

While this paper focuses effort on design of the convolutional layer for end-to-end CNN, it is helpful to be informed on other common layers found within the full pipeline structure. These

layers are as follows. Convolutional layer, is used for high-level feature extraction like the systematically known Sobel edge detection filter. These operations create a heavy computational strain on the total system and it has been shown that the convolutional layers within a CNN will comprise around 90% of computations for the entire CNN [5]. Pooling layer, is used for reducing the spatial size of the feature map. Similar to the convolutional layer, the pooling layer uses a kernel on the input map and applies a pooling function such as max pooling, average pooling, etc. Fully Connected layer, acts as the classifier stage and is commonly used at the end of the CNN pipeline as this layer has the most data access of all the layer types [6].

B. Reconfigurable Hardware and DPR

Reconfigurable computing devices change the behavior or functionality of a system at the hardware logic level. Behavioral changes are commonly for updating algorithms of compute processes while changing functionality allows designers to adapt the system's performance metrics like power efficiency, data throughput, etc. Since the introduction of the Xilinx FPGA in 1984, this class of computing devices has transitioned from a prototyping platform to being found in high-performance data centers as local accelerators.

In a normal mode of operation FPGAs load a complete bitstream file from some attached flash memory to self-configure a design. This bitstream directly varies in size with the design and as this design grows in hardware complexity, this will negatively impact the reconfiguration or boot period for the FPGA. In this case if a designer wanted to change a minor behavior or functionality of the FPGA, they would have to load the new bitstream into flash, shutdown the field device, and then boot to wait for the entire new bitstream configuration to load, regardless of how small the change is. This downtime is non-desirable and in some systems, unacceptable. In response to this issue, FPGA manufactures such as Xilinx Inc. and Altera Co. have implemented DPR in many of their FPGA families and have tightly integrated the tools for enabling the feature.

DPR enables the FPGA to be partially reprogrammed, removing the requirement to load the entire bitstream which reduces system downtime. There are many ways to achieve DPR and of the early implementations two methods were most common, Difference-based DPR and Module-based DPR [11]. Difference-based DPR is a method which tracks all changes of the reconfigurable design from the static design. As the changes from the static design grow, this leads to lengthy file requirements and load times when larger reconfiguration is needed. Module-based reconfiguration has been widely adopted and in this method, the designer is not required to even consider DPR until the static implementation of the design is complete. This method works by defining partition blocks of the static design as Reconfigurable Regions (RRs) and then create various Reconfigurable Blocks (RBs) to be interchangeable within the RRs [4]. Its worth mentioning that not all RBs must be defined at the time of device deployment and can be added later on. The only requirement is that any RB must fit within all RR that is has been designated to replace. Generally some added hardware such as decouplers is recommended to ensure data faults do not occur during DPR.

To complete reconfiguration, there are several options to access and modify the FPGA configuration registers. Commonly, an FPGA will use the serial port to load a bitstream file from attached flash memory but other methods are available for use. Some examples of available configuration methods are JTAG, Internal Configuration Access Port (ICAP), and Processor Configuration Access Port (PCAP). The ICAP and PCAP is hardware devices found within the static region for an FPGA so these are only available for partial reconfiguration. It is important note that the bandwidth of these ports can range drastically, this means the time spent reconfiguring the device is greatly affected by the port used. As an example from the Zynq-7000 FPGA used in testing, the serial port has a maximum bandwidth of 100Mb/s whereas the ICAP has a maximum bandwidth of 3.2Gb/s. By utilizing the DPR design flow, this enables a system designers to measurably reduce system downtime spent on device configuration.

III. Literature Survey

Hardware based convolution and CNN designs have been developed for many reasons, although a current focus of IoT development is to decentralize the processing from cloud based servers and move towards edge device processing. This means specialized CNN processors must be developed for the constrained environments of embedded edge devices. FPGAs are common target devices of literature as they have reduced development cycles along with the previously mentioned processing capabilities for the embedded platform. This section will discuss related literature that optimizes CNNs for the embedded platform along with FPGA based convolution architectures for embedded platforms.

A. CNNs Optimized for Embedded Platforms

Ma et. al [5] have worked on optimizing the convolution operation for lower inference latency and reducing off-chip memory access. This was achieved by heavily unrolling the convolution operation and reducing internal partial sum storage. Their method was used for several CNN models such as NiN, VGG-16, and RestNet-152, achieving up-to 715 GOPS. Du et. al [12] propose a deep CNN accelerator for IoT. Being an end-to-end CNN geared for IoT, energy efficiency was a primary goal and achieved a lightweight system at 434 GOPS/W. This was realized largely by minimizing unnecessary data movement with the processing architecture. Their design is very thorough and will be a good comparison for future work, but being implemented on ASIC, this will not be something to directly compare without translating to an FPGA device first.

[13] and [14] survey FPGA based CNNs and associated tools such as FINN and DNNWeaver which were developed to generate synthesizable HDL to map these DL algorithms to an FPGA for hardware acceleration. The FINN tool flow produces a highly paralleled processing structure for desirable throughput which streams data to all layers of the CNN. Conversely, DNNWeaver employs groups universal processing elements

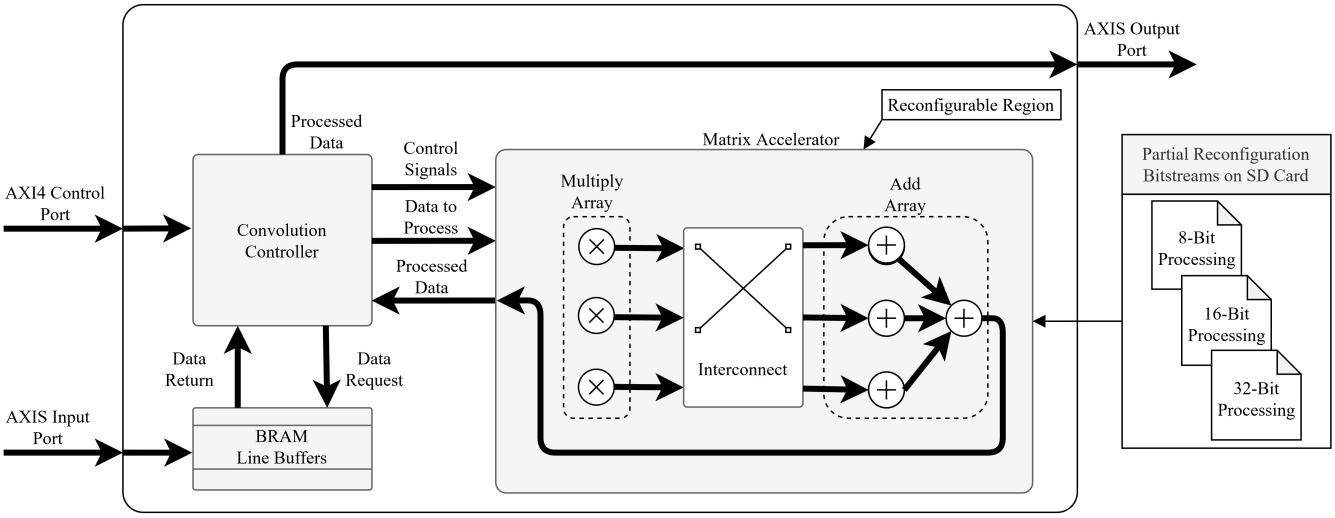

Fig. 1. Convolution Processing Element Architecture

and a state machine to assign data to one layer of the CNN at a time. This results in comparably lower hardware utilization than FINN, but also reduces the throughput capabilities.

B. Embedded FPGA Convolution Architectures

Wang et. al [7] present a Deep Learning Accelerator Unit (DLAU) which includes three internal processing units, the Tiled Matrix Multiplication Unit (TMMU), Part Sum Accumulation Unit (PSAU), and Activation Function Acceleration Unit (AFAU). Wang's DLAU is a convolution processor that scales the number of multipliers to equal the kernel size and increases the number of adders to complete all partial sum accumulates in a feed forward fashion. While this design is quite simple and flexible, the resulting hardware utilization puts the TMMU at the greatest of the three processing blocks. The DLAU achieves increased dynamic power efficiency compared to GPU based convolution processors. Cao et. al [8] propose a convolution accelerator with a systolic array structure. The systolic array first requires the convolution to be converted to matrix multiplication but is a processing structure that enables parallel processing and tight timing requirements for a high throughput relative to the FPGA Digital Signal Processors (DSP) utilization. The timing requirements arise because the individual PE must pass the data to the neighboring PE between arithmetic operations. To facilitate data reuse preventing excess off chip memory access, each of the systolic array PEs are delimited with an input and output FIFO. The output FIFO can feedback data to the input when necessary. Cao's proposed architecture was implemented on a Xilinx Zedboard FPGA and achieves a performance density 1.19× greater than the compared literature. Xu et. al [9] have proposed a fast convolution architecture. This architecture aims to accelerate the convolution operation by first applying a fast algorithm kernel which decreases the number of multiplications but increases the number of additions required for each convolution. After this fast algorithm was implemented, the paper goes further to highly parallelize the kernel usage by connecting 3 complex adder array structures on the output of the convolution kernel. These parallel adder arrays are connected with a crossbar to facilitate port selection. At first, Xu's proposed fast algorithm architecture has a throughput of 5.05 GOP/s but then with the fully parallel implementation the throughput increases to 15.14 GOP/s. While Xu's architecture has increased LookUp Table (LUT) usage compared to other work, the design has decreased DSP utilization which can be more common of a shortage when implementing larger systems.

C. Dynamic Reconfiguration for CNN

Youssef et. al [10] develop a full MNIST CNN that supports DPR to dynamically adjust the bit-precision of the CNN pipeline from 16 bit down to 5 bit while retaining 81.74% accuracy. This enables Youssef's CNN model to reduce the bit-precision for lower energy consumption when the system battery level drains without requiring the full pipeline to be disabled. The energy consumption of this model at 16 bits was shown to be 11.25 μJ per image and at 5 bits, 3.02 μJ per image, reducing the energy consumption rate by almost a factor of 4.

IV. Proposed Convolution Processing Element

The CPE is an important design component of a full CNN pipeline with dynamic partial reconfiguration support. The CPE architecture utilizes DPR to reconfigure the internal processing pipeline precision. The kernel size is fixed on synthesis to a size N, this generates N×N multipliers in parallel. This architecture also supports multi-channel data processing by increasing the number of parallel Matrix Accelerators (MA) to equal the channel count. This section discusses the internal blocks of the CPE, as shown in Figure. 1 for an internal block diagram of the architecture.

A. Convolution Controller

Each CPE hosts a controller with an internal state machine for handling the data processing and transferring steps. As the

CPE input feature map is not constant in spatial dimension, the controller can be adjusted during run-time via the AXI4 port to set corresponding spatial dimension registers and filter values.

Once configured, the CPE can accept a stream of data from any device using the AXIS protocol. AXIS data input enables the CPE to be used in many system types such as streaming or acceleration; in this paper our focus is on hardware acceleration for a software applications. The controller also hosts internal buffers for storing the current filter kernel and data to be processed by the matrix accelerator. The loading sequence for the data window buffer is shown in Figure. 2. The data being loaded is retrieved from the line buffers. The controller loads data to fill the window buffer but will not trigger for convolution until this buffer is full. Once the data buffer is filled, it is then fed to the matrix accelerator to calculate the convolution sum. Later, this convolution sum is then returned to the controller and transmitted on the AXIS output port.

B. Line Buffers

When reading data from main memory, the Direct Memory Access (DMA) controller will read an entire row of the input map before moving to the next row; this data flow is an issue as a convolution kernel with a width of N requires data points from N rows and N columns for each operation. To fix this data flow issue and also reduce redundant memory accessing, a line buffer is implemented for the CPE to hold N rows of the configured image width at a time. Each row of the line buffer is implemented as a dedicated Block Random Access Memory (BRAM) bank that is index addressable. To avoid shifting data between register rows at the end of a convolution row, an output multiplexer is used to correct the order being sent to the convolution controller. The buffer hardware utilization will vary with configurations but the the maximum input map spatial width will be have the greatest effect to this.

C. Matrix Accelerator and Compute Arrays

The matrix accelerator is where all the computations occur within the CPE and this block hosts a multiplier array and an adder array. The matrix accelerator does not have any control logic built-in so the controller is required to do this (by use of the mentioned state machine) to compute any convolution sum. DPR is used in the matrix accelerator block for changing the processing pipeline bit precision.

TABLE I
THREE CHANNEL HARDWARE UTILIZATION BREAKDOWN

	Component	LUTs	DSPs	BRAM
	Controller	5503	0	18
32-bit	Matrix Accelerator	271	27	0
	CPE	6316 (11.87%)	81 (36.82%)	18 (12.86%)
16-bit	Matrix Accelerator	365	9	0
	CPE	6610 (12.42%)	27 (12.27%)	18 (12.86%)
8-bit	Matrix Accelerator	721	0	0
	CPE	7741 (14.55%)	0 (0.00%)	18 (12.86%)

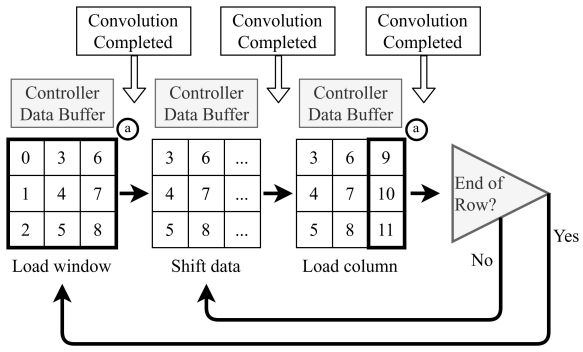

Fig. 2. Controller Data Buffer Loading Sequence
[a] Data being loaded into the Controller Data Buffer comes from the BRAM line buffers

TABLE II
HARDWARE UTILIZATION COMPARED TO OTHER DESIGNS

	[9] (Data reuse)	[15]	CPE	CPE	CPE
Data Type	N/A	N/A	Integer	Integer	Integer
Precision	16-bit	N/A	32-bit	16-bit	8-bit
Kernel Size	3×3	3×3	3×3	3×3	3×3
LUTs	684	38069	5291	5398	5785
FFs	672	18557	3762	3597	3661
BRAMs	N/A	72	6	6	6
DSPs	18	3	27	9	0

V. HARDWARE IMPLEMENTATION AND COMPARISONS

The high-level design shown in Figure. 3 was implemented on the Xilinx Pynq-Z2 using Vivado 2020.1; all design blocks shown have been developed with Verilog HDL. The Pynq-Z2 is an FPGA development board with a Zynq-7000 FPGA (xc7z020clg400-1) that was designed to support PYNQ, which is Xilinx's open-source framework to enable embedded developers to evaluate Xilinx FPGAs easily. The Zynq-7000 FPGA features a 650MHz dual-core Arm Cortex-A9 processor and the programmable logic is equivalent to an Artix-7 FPGA with 13,300 logic slices, each with four 6-input LUTs and 8 flip-flops. The dotted line represents the boundary between the Processing System (PS) and Programmable Logic (PL). The PS is a Xilinx IP that integrates dedicated hardware on the FPGA such as a dual core ARM Cotex CPU, DDR2/3 memory interfaces, etc.

Notice that the ARM Processor, DDR3 Memory, and DMA Controller would normally be present in an embedded system the CPE is designed to be used for, so these will not be included when analyzing hardware utilization and comparison with other FPGA convolution accelerators. To enable DPR for the system, the Xilinx DFX controller IP and ICAP must be added to the design. As the ICAP is hard-built into the FPGA and not used for any other purpose, this will not be considered as system overhead. Conversely, the DFX controller does add a small hardware utilization requirement to the system. This utilization is a fixed amount at 937 LUTs, regardless of the way DPR is implemented. This comprises 5% of the designs LUT

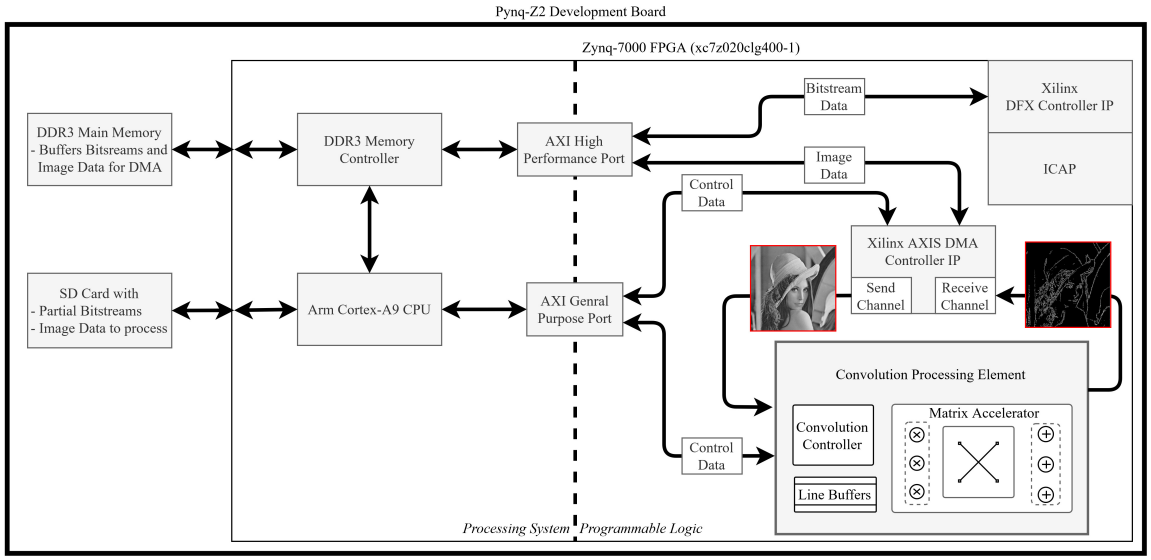

Fig. 3. Implemented Architecture for CPE Testing

usage and 1% of what is available on the device. The processing channel count of the CPE is fixed on synthesis, so single and three channel configurations were generated for testing. Implementation on FPGA has been planned for an efficient use of resources to be able to deploy it on more constrained platforms which is convenient for IoT edge devices. Although FPGAs are know to be power hungry, few devices are targeted for low power domain. Terasic offers a low power FPGA but this is implementation is prototyping on the available kits in the lab.

Once running, the architecture uses DPR to change between the processing resolutions of 8, 16, and 32 bit. The hardware for reconfigurable regions must be mapped manually to the FPGA floorplan. This is done within Vivado 2020.1 by creating pBlocks on the floorplan of the FPGA, and then assigning the associated reconfigurable modules. The Dynamic Function eXchange (DFX) wizard within Vivado can be used to create several preset DPR configurations for all RRs within the design, but each may also use DPR independently. The CPE hardware utilization at 8, 16, and 32 bit precision is shown in Table I, percentages represent proportion of available hardware. The DSP utilization peaks at 27 per MA and this is around 13% of the total available DSPs on the Pynq Z2 FPGA. This enables the processing pipeline to be extended for more feature extraction later on. At all precisions, the controller LUT and FF utilization is at least 6× greater than that of the matrix accelerator due to the complex logic structure required.

Lastly for hardware utilization analysis, the single channel CPE was compared with other FPGA based convolution architectures, shown in Table II. As there are several design approaches for employing DPR, the application at the processing element level is currently not well investigated. As a result, design comparison is not a simple task with other works but hardware utilization is generally a good metric. Xu et. al [9] uses half the LUTs and [15] LUT utilization is roughly 19× greater than CPE as the design mimics reconfiguration by storing many filters on the FPGA and swaps the running kernel by changing the active data path. This is a strong example of when DPR would benefit a design as much of the hardware is idle during runtime. These other kernels could be stored on separate memory and loaded as needed. The design also buffers the full input image before returning it to the host device which requires 72 BRAM blocks as temporary storage.

Initial testing for the CPE was done utilizing the PYNQ Jupyter Notebook Python environment. A test array of images were pre-loaded onto an SD card to be processed by the CPE. These images were organized by fixed pixel widths of 120, 240, 480, 720, and 1080 but all images vary in height. The runtime for each bit width/bit precision combination was averaged and is shown in Figure. 4 for single channel image processing. Bit precision largely does not affect the throughput capabilities of the system as multiply and add operations are done in parallel. This means that other metrics like power consumption or CNN accuracy would be important to compare with in the future.

While using the PYNQ environment DPR was accomplished using a JTAG port with a bandwidth of 66Mbps, but the Zynq-7000 has a dedicated ICAP with a bandwidth of 3.2Gbps to be used for DPR only. This increased bandwidth can reduce the reconfiguration time of the FPGA and Xilinx provides (1) for estimating this period.

$$\text{Reconfiguration Time} = \text{Bitstream size}/\text{Port Bandwidth} \quad (1)$$

As the MA has a partial reconfiguration file size of 220,872 bytes, this equation shows the following when using JTAG for DPR, $1,766,976\text{b}/66,000,000\text{bps} = 26.772\text{ms}$, similarly it can be shown that using ICAP for DPR would take 0.552ms. To reconfigure hardware precision with traditional methods would require a full FPGA reconfiguration via JTAG, as our

fully implemented design has a file size of 4,045,564 bytes the reconfiguration period would be 490.371ms. This means that the ICAP decreased our DPR period by a factor of 48× and when compared to traditional methods a factor of 800×. To enable the ICAP, testing was moved to the C based Vitis environment which Xilinx directly develops driver code for.

Using Vitis, an input map of size 64×64 was processed by the 32 bit precision CPE in single and three channel configurations. Analysis of these processing run times are shown in Table III. To process three channel input data with the single channel CPE configuration, each channel was sequentially queued to be read by the DMA controller.

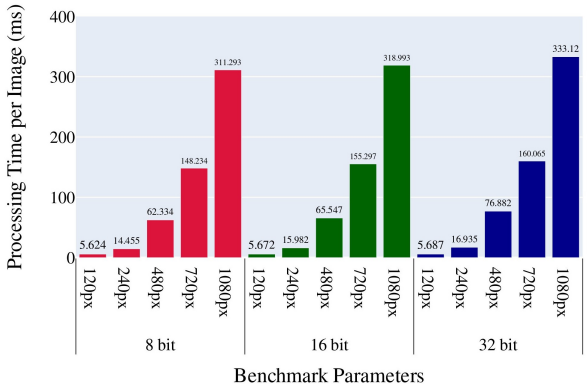

Fig. 4. CPE Average Processing Times for Single Channel Images

TABLE III
CPE Runtime Analysis on 64x64 Input Map

CPE Channels	Input Channels	Convolutions per Frame	Cycles per Frame	Process FPS
1	1	3969	31190	3206
1	3	11907	93570	1068
3	3	11907	47070	2124

VI. Conclusions

In this paper, Convolution Processing Unit is implemented aiming at utilizing DPR for the utilization of different precision configurations. DPR is showing a great benefit by saving time and resources as needed by using an entirely different processing element which is dedicated at computing specific precision instead of implementing an all-in-one solution for design flexibility. DPR's timing is crucial for this process as the smaller the RR, the faster it is able to be reconfigured. In our case, the RR is focused at a small scale of the processing element which can be theoretically be reconfigured in 0.552ms using ICAP compared to the entire design at 490.371ms using JTAG. This shows that DPR enables the device to be reconfigured for new data type processing 800× faster. The potential for dynamically reconfiguring smaller elements is much more significant that switching an entire structure in terms of runtime and throughput expectations. In this scenario, it also allows to incorporate more parallelized elements if needed to accelerate further computation when using smaller precisions.

References

[1] M. P. Véstias, "A survey of convolutional neural networks on edge with reconfigurable computing," *Algorithms*, vol. 12, no. 8, 2019.

[2] K. Guo, L. Sui, J. Qiu, *et al.*, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, 2018.

[3] M. Qasaimeh, J. Zambreno, P. H. Jones, *et al.*, "Analyzing the energy-efficiency of vision kernels on embedded cpu, gpu and fpga platforms," in *International Symposium on Field-Programmable Custom Computing Machines*, 2019, pp. 336–336.

[4] O. Eldash, A. Frost, K. Khalil, *et al.*, "Dynamically reconfigurable deep learning for efficient video processing in smart iot systems," in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, 2020, pp. 1–6.

[5] Y. Ma, Y. Cao, S. Vrudhula, *et al.*, "Optimizing the convolution operation to accelerate deep neural networks on fpga," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 7, pp. 1354–1367, 2018.

[6] R. Ding, G. Su, G. Bai, *et al.*, "A fpga-based accelerator of convolutional neural network for face feature extraction," in *2019 IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC)*, 2019, pp. 1–3.

[7] C. Wang, L. Gong, Q. Yu, *et al.*, "Dlau: A scalable deep learning accelerator unit on fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 3, pp. 513–517, 2017.

[8] Y. Cao, X. Wei, T. Qiao, *et al.*, "Fpga-based accelerator for convolution operations," in *International Conference on Signal, Information and Data Processing*, 2019, pp. 1–5.

[9] W. Xu, Z. Wang, X. You, *et al.*, "Efficient fast convolution architectures for convolutional neural network," in *International Conference on ASIC*, 2017, pp. 904–907.

[10] E. Youssef, H. A. Elsemary, M. A. El-Moursy, *et al.*, "Energy adaptive convolution neural network using dynamic partial reconfiguration," in *International Midwest Symposium on Circuits and Systems*, 2020, pp. 325–328.

[11] W. Lie and W. Feng-yan, "Dynamic partial reconfiguration in fpgas," in *Symposium on Intelligent Information Technology Application*, vol. 2, 2009, pp. 445–448.

[12] L. Du, Y. Du, Y. Li, *et al.*, "A reconfigurable streaming deep convolutional neural network accelerator for internet of things," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 1, pp. 198–208, 2018.

[13] K. Guo, S. Zeng, J. Yu, *et al.*, "A survey of FPGA based neural network accelerator," *CoRR*, 2017.

[14] S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Toolflows for mapping convolutional neural networks on fpgas: A survey and future directions," *ACM Comput. Surv.*, vol. 51, no. 3, 2018.

[15] A. Rupani, P. Whig, G. Sujediya, *et al.*, "Hardware implementation of iot-based image processing filters," in. Jul. 2018, pp. 681–691.