information

Article

Efficient Edge-AI Application Deployment for FPGAs †

Stavros Kalapothas, Georgios Flamis and Paris Kitsos *

Electronic Circuits, Systems and Applications (ECSA) Laboratory, Electrical and Computer Engineering Department, University of Peloponnese, 26334 Patras, Greece; s.kalapothas@go.uop.gr (S.K.); g.flamis@go.uop.gr (G.F.)

* Correspondence: kitsos@uop.gr

† This paper is an extended version of our paper published in SEEDA-CECNSM 2021.

Abstract: Field Programmable Gate Array (FPGA) accelerators have been widely adopted for artificial intelligence (AI) applications on edge devices (Edge-AI) utilizing Deep Neural Networks (DNN) architectures. FPGAs have gained their reputation due to the greater energy efficiency and high parallelism than microcontrollers (MCU) and graphical processing units (GPU), while they are easier to develop and more reconfigurable than the Application Specific Integrated Circuit (ASIC). The development and building of AI applications on resource constraint devices such as FPGAs remains a challenge, however, due to the co-design approach, which requires a valuable expertise in low-level hardware design and in software development. This paper explores the efficacy and the dynamic deployment of hardware accelerated applications on the Kria KV260 development platform based on the Xilinx Kria K26 system-on-module (SoM), which includes a Zynq multiprocessor system-on-chip (MPSoC). The platform supports the Python-based PYNQ framework and maintains a high level of versatility with the support of custom bitstreams (overlays). The demonstration proved the reconfigurabibilty and the overall ease of implementation with low-footprint machine learning (ML) algorithms.

Keywords: artificial intelligence; deep learning; FPGA; PYNQ; MPSoC; DNN; CNN; Kria; KV260; edge-AI

check for updates

Citation: Kalapothas, S.; Flamis, G.; Kitsos, P. Efficient Edge-AI Application Deployment for FPGAs. *Information* **2022**, *13*, 279. https://doi.org/10.3390/info13060279

Academic Editor: Markos G. Tsipouras, Alexandros T. Tzallas, Nikolaos Giannakeas and Katerina D. Tzimourta

Received: 15 March 2022
Accepted: 25 May 2022
Published: 28 May 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In recent years, the demand for intelligent applications is continuously on an upward trend in both the research community and the business information and communication technology (ICT) market. The application domain, where artificial intelligence (AI) and especially machine learning (ML) algorithms are being deployed, is spread to various sectors, such as healthcare, industry, education and safety, offering smart solutions that are immersed in our daily life. The Internet of Things (IoT) and Advanced Driver Assistance Systems (ADAS), with a plethora of sensors and actuators in the field, require massive data processing at the edge. Therefore, the computation intensive tasks that derive from ML algorithms typically imply efficient software and hardware architectures.

The extensive use of deep learning (DL) algorithms, such as the Convolutional Neural Network (CNN) and Deep Neural Network (DNN) and the inherent model complexity, including billions of 32-bit floating-point (FP32)-based multiplication-and-accumulation (MAC) operations, formulated the exploitation of hardware accelerators [1]. In particular, the field programmable gate array (FPGA)-based hardware accelerators, offer a significant advantage for Edge-AI applications, in terms of low-latency and power efficiency, whereas the graphical processing unit (GPU)-based hardware architectures introduce more power consumption, and application-specific integrated circuit (ASIC)-based accelerators offer limited, or no reconfiguration capabilities [2]. In general, modern FPGAs incorporate high performance digital signal processing (DSP) modules and fast block random access memory (BRAM) that introduce accelerated convolution operations. Furthermore, the

parallel processing elements and low-latency communication paths of FPGAs are exploited for higher performance gain at the massive operations in CNN models.

However, the integration of CNN and DNN models onto FPGAs poses numerous design challenges mainly related to the hardware resource optimization and which is transcribed to a drastic reduction of the computing and memory elements. The approach that is currently trending for minimizing the hardware utilization is to introduce model compression techniques, such as weight quantization with reduced precision, from FP32 to 8-bit integer (INT8) representations and network pruning [3]. In addition, the deployment of pre-trained DNN models onto FPGAs is not a trivial task and requires knowledge of the underlying hardware architecture at the circuit level, the design tools and the DNN model integration workflows.

In the context of tooling and frameworks, FPGA manufacturing companies have introduced their own vendor-specific and proprietary hardware synthesis tools, such as the Xilinx Vivado Design Suite [4], Intel Quartus [5] and Lattice Diamond [6]. In the last few years, the advances in the open source frameworks have been substantial. In particular, Yosys [7] is an open source framework for register transfer level (RTL) synthesis, which is currently supported to only the Xilinx 7-series and Lattice iCE40, ECP5 and FPGAs. PYNQ [8], is an additional open source and very modifiable framework with a Python-based programming interface for high level synthesis and rapid CNN/DNN prototyping on FPGAs with a Zynq system-on-chip (SoC). Moreover, for AI inference workflows, Xilinx provides Vitis AI [9], Intel the Open Vino [10] and Lattice the SensAI [11]. In previous research work, there are significant contributions in FPGA frameworks for AI inference such as DNNWEAVER [12], CAFFEINE [13], FINN [14], FP-DNN [15] and Google CFU [16]. Similarly, in DNN research many open source frameworks have been released to facilitate network modeling, development and experimentation, i.e., TensorFlow [17], PyTorch [18], Caffe [19] and a handful of others.

In this work, a DNN deployment workflow for AI inference with the adoption of the PYNQ framework [20], is presented. The newly developed Xilinx Kria system-on-module (SoM) [21] is exploited as the target FPGA, which includes a high-performance and production ready computing solution based on the Zynq Ultrascale+ multiprocessor system-on-chip (MPSoC) [22] that embeds port connectivity features for edge applications. The scope is intentionally narrowed down onto the inference stage and the deployment of the pre-trained models to the Kria SoM. The presented workflow is based on a previous work [23], where model training followed by model compression, using a GPU with a Compute Unified Device Architecture (CUDA) [24] framework and the Deep Neural Network Development Kit (DNNDK) [25], has been shown, respectively. The deployment target was a low-cost ZedBoard FPGA, running AI inference and collecting performance metrics. These metrics are amended with the ones collected from the Kria SoM, where the evolution of the workflow is demonstrated. This study showcased the flexibility of the platform that incorporates FPGA fabric and the ARM-based embedded CPU in order to introduce an adaptive SoC that expedites the development of hardware-accelerated solutions for ML/DL algorithms.

The remainder of the paper is structured as follows. In Section 2, a hardware implementation workflow using the Zynq MPSoC and the PYNQ framework is presented. In Section 3, the results are evaluated and a comparison with other implementations is shown. Further discussion and the final remarks are enclosed in Section 4.

2. Experimental Setup

The development setup consists of a Xilinx Kria KV260 Vision Starter Kit hardware platform where all the benchmarks were conducted. An Intel i7-10510U-based system for support and debugging operations included the joint test action group (JTAG) interface, serial and network connectivity. The platform runs a custom Ubuntu 20.04 LTS device image, certified by Xilinx and released by Canonical [26]. The development was based on the PYNQ framework using various hardware overlays. The deep learning processing

unit (DPU)-PYNQ overlay [27], which contains a Vitis AI v1.4.0 DPU, was mainly tested. Vivado 2020.2 was also used for the customised overlays' exploration on the Intel system. Pre-trained ML models were imported from Model Zoo [28] and performed the testing using TensorFlow 2 [17], Python 3.8 [29] and Jupyter Notebooks [30] on the Kria.

2.1. Dpu Architecture

The DPU is a parameterizable computing engine that enables CNN/DNN models' inference. In 2018, Xilinx acquired DeePhi, which designed the Neural Network (NN)-based accelerators and demonstrated in practise that up to 20% low-bit quantization and more than 40% network pruning can have less than a 1% loss accuracy drop [31]. Thus, network sparsity has introduced a positive impact in overall network performance, in terms of latency and throughput. The Kria hardware platform enables support for a B4096F DPU architecture with a high-parallelism of 8 pixel, 16 input channels and 16 output channels. The DPU has inherent support for CNN and DNN models, and its parameters can be configurable in terms of the number of DPU cores, pooling layers, activation functions, lookup tables (LUT), BRAM and DSP slices, as required.

2.2. Initial Setup

The process of preparing the Ubuntu OS image and connecting the peripherals to the board is available on a Getting Started page in Xilinx's github public repository [32]. In order to install PYNQ, the dedicated Kria-PYNQ git repository was cloned and followed a straight-forward installation process using a script. Furthermore, with Jupyter included in the target system, hardware overlays on the programmable logic (PL) can be loaded and interactions with the processing system (PS) can be developed, in an intuitive way. The Kria SoM, as already mentioned, features a Zynq UltraScale+ MPSoC that includes a quad-core ARM Cortex-A53 application processing unit (APU), a dual-core ARM Cortex-R5F real-time processing unit (RPU) and an ARM Mali-400 graphics processing unit (GPU). As a first 'hello-world' example, an image resize overlay in PL using the Nearest-neighbor interpolation, is tested. Alternative interpolation techniques, such as Bilinear and Area interpolation, are also supported in the PL, wherein all functions can be initiated from the PS using the Vitis AI API. In the aforementioned example, the pre-processing is performed at the PL and the results are presented in the PS, since there is data communication between both, through the AXI interconnection interfaces. In Figure 1, a high-level model of the system is illustrated.

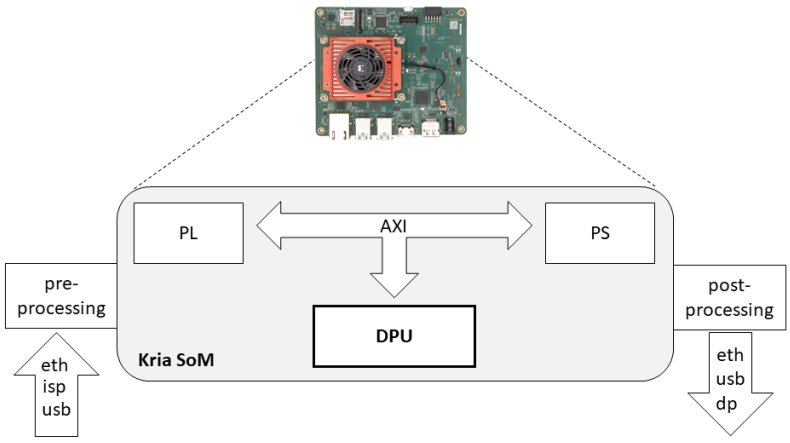

Figure 1. Kria SoM Model.

2.3. Workflow DNN Inference

The process workflow of a deep learning project includes several steps, such as data collection, data pre-processing, model selection, model training and model inference. Theoretically, after the training process is completed and the model is 'freezed', it can be deployed to a target system for inference. However, in typical real-world scenarios, model

compression techniques can optimize the trained models with no significant accuracy and performance degradation. Model training and model compression features are supported natively in the workflow. Model compression is of the utmost importance when custom DNN models need to be deployed on FPGAs and run inference on the edge, where power efficiency is considered a key requirement.

In this setup, AI inference is demonstrated by having deployed on the Kria a custom pre-trained model based on the MNIST dataset [33] and quantized using the Vitis AI (legacy DNNDK) framework. The pre-trained model is embedded in the DPU and is loaded with PYNQ using a hardware overlay. The whole MNIST classifier demo can be developed with a few lines of code in Python. Some of the basic operations and methods are imported from DPU-PYNQ, to manipulate hardware overlays and load AI models in the DPU.

Custom hardware overlays (bitstreams) are built with Vitis in a workflow that is documented in detail and in conjunction with the custom model training and compression process, in the previous work [23]. Inside the Jupyter Notebook, the python code for the classification task outputs the prediction on top of every digit picture. In Figure 2 below, a snapshot from a small test dataset, having been classified, is shown.

Figure 2. MNIST test dataset predictions.

Next, the RESNET50 [34], based on Caffe framework and InceptionV1 [35] based on TensorFlow, were explored. Both models are available in the Model Zoo, with embedded DPU acceleration for basic image classification task execution. The pre-built DPU core is configured with support for the average pooling layers and rectified linear unit (ReLU6) or a leaky rectified linear unit (LReLU) [36] activation functions. The code includes pre-processing functions for image manipulation using OpenCV [37] libraries. NumPy [38] is also supported in PYNQ, as a powerful Python library that facilitates operations on array objects, statistics and a plethora of mathematical functions that are considered fundamental in deep learning. In addition, natively, C++ code can be imported and used seamlessly with the Python code, which is a feature enabled with Pybind11 [39] and supported in PYNQ. An output from an object classification task, including the prediction accuracy, is shown in Figure 3, below.

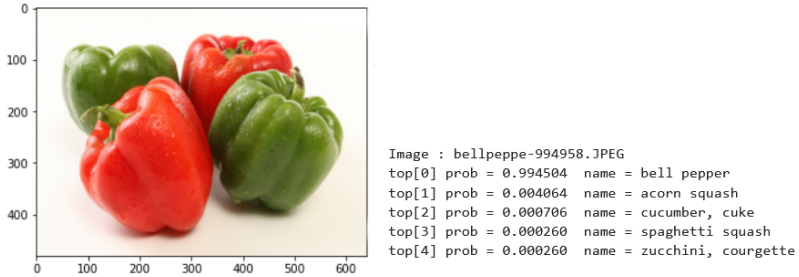

Figure 3. Object classification task.

Furthermore, Vitis AI library applications are available through the Ubuntu Snap store. The sample applications are based on a B3136 DPU overlay, running at 300 MHz in a $8 \times 14 \times 14$ configuration. To enable a B4096 DPU accelerator, which supports a $8 \times 16 \times 16$ configuration, a custom overlay needs to be generated together with the ML model to be referenced in the application code. The application code must be compressed to a single application package using the single Platform Assets Container (PAC) [40] and consequently can be copied to the target platform and then loaded using the command line utility 'xmutil'.

Moreover, the performance of a keyword spotting (KWS) application based on the depthwise separable convolutional neural network (DS-CNN) [41] architecture using the Google speech commands dataset [42], has been tested and its performance metrics examined. As a last step, a plate detection application based on the DenseBox [43] network architecture and the Caffe framework, was tested with a custom car license plate dataset, publicly available in Kaggle [44]. In Figure 4, the inference is performed on the sample images included in the dataset.

Figure 4. DenseBox car plate detection.

2.4. Advanced AI Applications

To exercise further the capabilities of the Kria SoM in AI inference, an extra set of pre-trained models for image segmentation and object detection were examined. These applications were also available in the Vitis AI Model Zoo repository and are lately at the forefront of research, for instance, the advanced AI scenarios of ADAS and Autonomous Driving (AD). More in particular, traffic detection, lane detection and segmentation algorithms have been tested in real-world road driving conditions.

2.4.1. MultiTask

The MultiTask model [45] is based on Caffe framework and executes two separate subtasks: semantic segmentation and single shot detector (SSD) on the BDD100K dataset [46]. The application domain is a semantic segmentation in a road scenery, including streets, highways and residential buildings and vehicle detection and tracking. A video subset with 288 $\times$ 512 pixel resolution was extracted from the BDD100K dataset and was used as an input to the DNN.

2.4.2. Lane Detection

The vpgnet_pruned_0_99 is a pruned model based on VPGNet [47], which is included in the Vitis AI Model Zoo. The model network structure includes eight convolutional and three pooling layers and performs four tasks in the parallel: grid regression, object detection, multi-label classification and vanishing point prediction. For the ADAS application scenario, the model was tested for lanes and road markings detection and classification, and the vanishing point prediction task. In the experiments, the Caltech Lanes Dataset [48] with 1225 individual video frames in 640 $\times$ 480 pixels resolution, were being tested.

2.4.3. YOLOv3 for ADAS

A pruned version of the YOLOv3 [49] model based on the DarkNet [50] framework consisting of 53 convolutional layers, which is included under the name yolov3_adas_pruned_0_9 in the Model Zoo, was also added in the experiments. The Cityscapes dataset [51], consisting of urban street scenes in 5000 video frames of 256 $\times$ 512 pixels resolution, was fed in the DNN to test the inference in the ADAS application scenarios.

2.4.4. Object Detection with VGG-19

Similarly, the VGG-19 [52] model was included in the benchmarks. There are 19 layers, with 16 convolutional layers and 3 fully connected (FC) layers, embedded in the network architecture. In the last FC layer, 1000 outputs, which correspond to the 1000 object

categories of classification, are supported. The pre-trained model vgg_19_tf is based on TensorFlow framework and has been trained on the ImageNet dataset [53] with 1000 object classes and 100,000 test images, with an input resolution of 224 × 224 pixels.

2.4.5. Benchmark Execution

The respective models, as well as the application binaries, were downloaded and ran locally in the target board using the Kria KV260 Vision AI Starter Kit Benchmark utility [54]. The video files from the aforementioned datasets were loaded onto the sd card and then fed into the application, processed through the single core DPU and then output directly to a monitor, or to a remote shell, via the network. The application ran in single and multiple threads to calculate the maximum total throughput. In the majority of the algorithms tested, a real-time performance of 30 frame-per-seconds (FPS) and above was observed. In Figure 5, a collage of multiple snapshots of the different AI applications while they ran on the board is visualized.

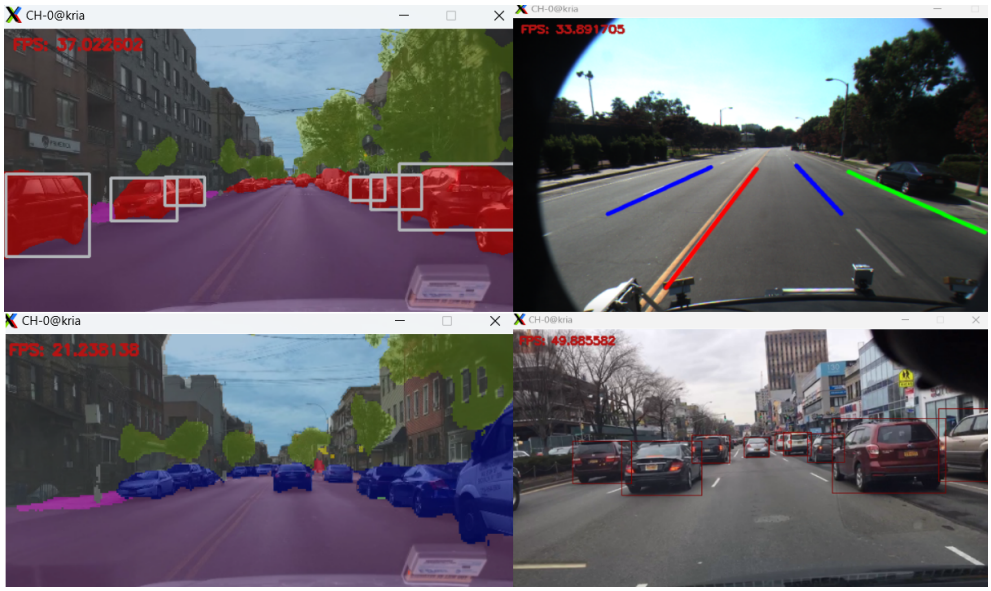

Figure 5. Lane detection and semantic segmentation.

3. Results

The performance evaluation was conducted on the KV260 development kit, consisting of a carrier board and the Kria SoM. Most of the application code tested was run directly from Jupyter Notebooks. In parallel, another set of application demos were tested and ran from the Linux command line interface. In the first scenario, a hardware overlay that contained the DPU was loaded using Jupyter and for the latter, the NLP-SmartVision [55] application, which contained also a DPU, was loaded from the Ubuntu Snap store. For the inference tests, standalone images and video files were used. In addition, some publicly available datasets were used for performance testing. To explore even further some real-world scenarios, and specifically for the video input, a 4K USB3 camera, as well as a camera connected to the mobile industry processor interface (MIPI), were also used. The output stage of the DL algorithms were produced mainly to a monitor connected via HDMI port, plus over a network connection via the ethernet interface.

The performance metrics collected are ranked based on the FPS which is a rather simple, but yet effective, computation to measure the images, or video frames processed per second, by the deep learning algorithm. The FPS Equation (1) is depicted below.

$$FPS = \frac{total\_images\ (or\ video\ frames)}{frame\_execution\_time\_start - frame\_execution\_time\_end} \quad (1)$$

In the context of the DPU performance, the DPU mean processing time and the end-to-end mean (E2E_MEAN) processing time were measured. The formula is presented in Equation (2). Therefore, the E2E_MEAN differs from the DPU_mean, as the first adds up to the pre-processing and post-processing time. In example, the pre-processing may include the time it takes for the image to be read from a camera connected to a USB interface, or a video to be read from an sd card and the execution time of processing the filters. Whereas, post-processing may include drawing patterns and other image partitioning techniques.

$$E2E\_MEAN = pre\_processing\_mean + DPU\_mean + post\_processing\_mean \quad (2)$$

In Table 1, the collected FPS from the various DL algorithms' implementations of computer vision applications is shown. Each AI model inference was tested separately, utilizing a single DPU core, with one thread, as well as with two thread executions per cpu, to cover low latency and high throughput scenarios, respectively. The measurements were taken mainly on the Kria board; however, a subset of the pre-trained models were also tested on the ZedBoard and measured the corresponding FPS. Indicatively, the inference performance has shown a 3× increase on the Kria against the performance achieved on the ZedBoard, when running inference with the RESNET50 model, and more than 7.5× increase in performance when running inference with the InceptionV1 model.

Table 1. Model inference performance on Xilinx K26 B3136 DPU & ZedBoard B1152 DPU.

Model	Kria K26 SoM FPS (1 thread)	Kria K26 SoM FPS (2 thread)	ZedBoard FPS (1 thread)	ZedBoard FPS (2 thread)
LeNet-Custom (MNIST)	2615.54	-	1237.62	-
RESNET50 (CIFAR10)	61.02	63.28	19.82	-
InceptionV1_tf	136.13	151.31	17.48	19.22
DenseBox_320_320	589.37	682.66	-	-
DenseBox_640_360	282.85	326.41	-	53.39
multi_task [1]	34.91	37.49	-	-
ssd_mobilenet_v2	46.19	49.93	-	-
vgg_19_tf	20.22	20.26	-	-
vpgnet_pruned_0_99	141.72	155.94	-	-
yolov3_adas_pruned_0_9	89.52	93.03	-	-

[1] multi_task supports both object detection and segmentation.

On a side note, the development with the ZedBoard has taken an enormous amount of work, especially to generate a configuration with the most recent version of DPU. However, all attempts proved to be unsuccessful, due to incompatibility issues between the tools' version and the host's operating system. In the example, with installations on Ubuntu 20.04 & 18.04, there had been always a missing key factor that had prohibited compilation. Awkwardly, the attempts to use an older DPU had not worked as expected, as the specific IP was no longer available at the AMD/Xilinx repository. Hence, the work was limited to use only the pre-compiled code configurations. Thus, the two threads' operation was not supported for some of the configurations in the DNNDK examples for the ZedBoard. Due to the aforementioned constraints with the tool operation, the results collected for the ZedBoard were narrowed.

Regardless of the plethora of computer vision applications and related deep learning models, a considerable number of models also exist in different types of AI applications, such as voice recognition and keyword detection. The KWS deep learning DS-CNN model inference test on the Kria detected 2398 keywords correctly out of 2567 in total, from a dataset with 30 unique keywords included in pre-recorded audio files and, therefore, achieved a 93.41% accuracy. This is a top-15 SOTA performance and ∼10% higher than the performance achieved by a DNN with similar network characteristics [41].

The achieved accuracy of the custom LeNet-5 model against the MNIST dataset with the current implementation on the Kria SoM is 98.71%. This level of accuracy is aligned to the 98.54% accuracy compared to the model ran on XC7Z020 with a B1152 DPU clocked at 90MHz, as presented in the previous work. However, the effort to deploy the model on XC7Z020 was based on DNNDK, which recently has been deprecated, but Vitis AI backward compatibility is maintained for legacy applications based on the old framework. In the legacy flow, an object ELF file is compiled after the development and uploaded in the target board manually. Nonetheless, with the use of the PYNQ composable overlays pipeline, the development, as well as the deployment of the PL and PS components, is more efficient via Jupyter, a web-based integrated development environment (IDE), which is adequate in rapid prototyping. An additional advantage is that the Kria platform has been supplemented by an ecosystem that leans towards adaptive workloads, underpinned by a desktop-like Linux OS and an application store with a wide range support of libraries and out-of-the box production-ready AI applications. These improvements are considered decisive for Edge-AI solution deployments. In Table 2, a summary of the PL resources used for each DPU implementation, compared to the total resources available in both development boards, is shown.

Table 2. PL Resources in Kria vs ZedBoard.

Resource	Kria K26 SoM Available	Kria K26 SoM Utilized with B3136 DPU	ZedBoard Available	ZedBoard Utilized with B1152 DPU
LUT	117,120	43,366	53,200	30,074
DSP	1248	548	220	194
LUTRAM	57,600	-	17,400	1738
BRAM	144	67	140	117.5
URAM	64	44	-	-

The resource utilization of the Kria and the ZedBoard development platforms, in terms of cpu, ram and power, is shown in Tables 3 and 4, respectively. A Linux tool called platformstats, which is provided as open-source by Xilinx [56], is executed locally on the target board to collect the statistics. More in detail, the Kria SoM is equipped with a current/voltage/power monitor based on INA260, a 16-bit precision analog-to-digital converter (ADC) by Texas Instruments [57]. The ADC readings are polled at fixed time intervals of 1 sec by the tool. For the power consumption analysis, different measurements were taken before and after the DPU, which is included in the NLP-SmartVision application, and is loaded. The DenseBox AI model is provided in this application to exploit object detection. Therefore, a separate power measurement while the inference application is running has also been collected and complemented the data in comparison. A notable 30.2% increase in current consumption and over 35% in cpu utilization is observed when the AI application is running, but there is no significant change between the idle and DPU only loaded states. It has been noticed that the electric current is not only consumed at the PL level, but it is also equally spread and consumed at the PS level when the inference application is running. The preceding behaviour is expected due to the fact in the specific application (object detection on a video file input and a network stream output) the pre/post-processing functions are included at the PS level.

On the ZedBoard, the InceptionV1 AI model is loaded onto the DPU. The power consumption (P) is measured with manual probing of the voltage (V) across the 10 milliohm shunt resistor (R) found on the J21 connector on the board, and the current (I) is calculated using the Ohm's Law (3), which then is multiplied by the 12V rail voltage (4). Therefore, a solid increase in power consumption (21.8%) and cpu utilization (28.1%) during inference is detected during the measurements. In Figure 6, the power efficiency in terms of performance per watt of both hardware accelerators platforms is shown.

Table 3. Resource utilization on Kria K26 SoM.

Kria K26 SoM		**HW Info**	
DPU Frequency		300 MHz	
CPU Frequency		1200 MHz	
RAM Total		3.93 GB	
ine **Status**	**Idle**	**DPU**	**DPU + Inference**
ine CPU Util	0.5%	0.8%	41.4%
RAM Used	529 MB	535 MB	589 MB
SoM Voltage	5048 mV	5048 mV	5048 mV
SoM Current	952 mA	956 mA	1425 mA
SoM Power	4806 mW	4826 mW	7340 mW

Table 4. Resource utilization on ZedBoard.

XC7Z020		**HW Info**	
DPU Frequency		90 MHz	
CPU Frequency		667 MHz	
RAM Total		512 MB	
Status	**Idle**	**DPU**	**DPU + Inference**
CPU Util	0.1%	0.2%	28.2%
RAM Used	29 MB	104 MB	159 MB
System Voltage	12 V	12 V	12 V
System Current	320 mA	320 mA	390 mA
System Power	3840 mW	3840 mW	4680 mW

$$I = V/R \tag{3}$$

$$P = V \cdot I \tag{4}$$

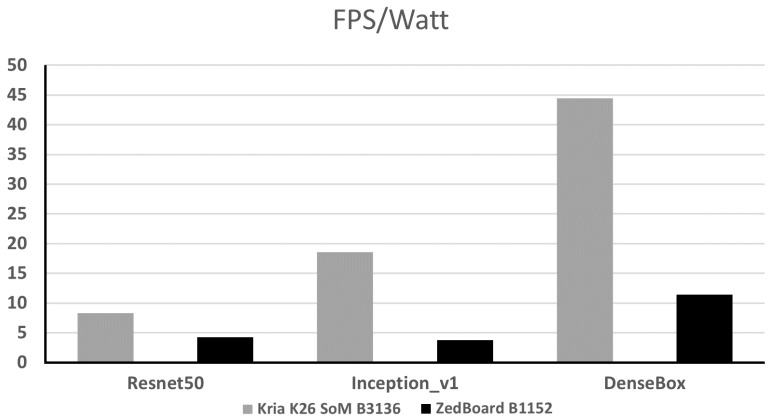

Figure 6. Performance per watt.

4. Discussion

In this work, a qualitative method to assess the capabilities of a reconfigurable platform with many computing vision-related features has been proposed. Our evaluation demonstrated a good combination of computing performance in a low power budget, and was fulfilled by Kria SoM. With respect to the power efficiency, the experimental results demonstrated a slight variation, which was expected and aligned with the workload increase, but in general the measurements from an FPGA-based SoM hardware were not unforeseen. Further, the experimental evaluation of multiple deep learning algorithms that process data streams demonstrated that all computational intensive workloads can be managed by the SoM with no issue. An extensive part of the study also included a comparative evaluation of computational performance and power efficiency between the

ZedBoard and the Kria SoM. Equally, the software ecosystem with PYNQ at its core and well known frameworks including TensorFlow, Caffe and PyTorch, provided a smooth software development and delivery experience. In fact, the team have worked also in the past with the legacy DNNDK framework on the less capable Xilinx Zedboard, which helped realize the merits of the software ecosystem around the new Kria SoM platform. The assortment of the AI applications the platform is targeted, spans in various domains such as, Healthcare, Security, IoT, Autonomous Driving and is supported by a variety of network architectures which, in most cases, are domain-specific. In the conducted experiments, an indicative collection of AI models were selected and these were rigorously demoed with the use of video recordings from the datasets, or with a real-time camera feed, as a source. Ultimately, the Kria board is qualified as a contender for Edge-AI applications in demanding tasks, such as semantic segmentation and object classification within real-time video feeds, where low-latency and high throughput are considered crucial.

To enhance the perception of the role of the frameworks for different hardware accelerator platforms, the team have tried to supplement the comparisons with performance data coming from the execution of inference on platforms with different architectures. Therefore, OpenVino with an Intel Neural Compute Stick [58] and TensorFlow Lite with a Qualcomm Snapdragon 870 SoC [59]-based mobile device and Android 11 OS, have been tested. However, the tests could not easily be replicated across the different frameworks, as different pruning techniques and different models were available in each respective Model Zoo repository. Thus, a different method definition that will take into consideration a common model training and different quantization, as well as pruning methods, which will be separately applied per model and per framework, is required.

In the future, the team is intended to focus the research efforts in ADAS applications and conduct more domain-specific experiments, respectively. The capabilities of the hardware accelerator platform provides enough room for exploration of multi-modal sensor fusion and data processing at the core of the SoM, not solely with vision data, but also with other types of sensor data, e.g., LiDAR, IR, Sonar, accelerometer, or other environmental sensor, towards building AI applications for decision support at the Edge.

Author Contributions: Conceptualization, S.K. and G.F.; methodology, S.K.; software, S.K.; validation, S.K., G.F. and P.K.; formal analysis, S.K.; investigation, S.K.; resources, S.K.; data curation, S.K.; writing—original draft preparation, S.K.; writing—review and editing, P.K.; visualization, S.K.; supervision, P.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not Applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Publicly available datasets were analyzed in this study. These data are not curated by our team and are available in: http://yann.lecun.com/exdb/mnist (accessed on 15 March 2022), in: http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz (accessed on 15 March 2022), in: https://www.kaggle.com/andrewmvd/car-plate-detection (accessed on 15 March 2022) and in: https://www.bdd100k.com/ (accessed on 15 March 2022).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Li, E.; Zeng, L.; Zhou, Z.; Chen, X. Edge AI: On-Demand Accelerating Deep Neural Network Inference via Edge Computing. *IEEE Trans. Wirel. Commun.* **2020**, *19*, 447–457. [CrossRef]
2. Flamis, G.; Kalapothas, S.; Kitsos, P. Best Practices for the Deployment of Edge Inference: The Conclusions to Start Designing. *Electronics* **2021**, *10*, 1912. [CrossRef]
3. Wu, R.; Guo, X.; Du, J.; Li, J. Accelerating Neural Network Inference on FPGA-Based Platforms—A Survey. *Electronics* **2021**, *10*, 1025. [CrossRef]
4. Xilinx. Vivado. Available online: https://www.xilinx.com/products/design-tools/vivado.html (accessed on 27 February 2022).
5. Intel Quartus. Available online: https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html (accessed on 27 February 2022).

6. Lattice Diamond. Available online: https://www.latticesemi.com/latticediamond (accessed on 27 February 2022).
7. Yosys. Open Sythesis Suite. Available online: https://github.com/YosysHQ/yosys (accessed on 27 February 2022).
8. Wang, E.; Davis, J.J.; Cheung, P.Y.K. A PYNQ-Based Framework for Rapid CNN Prototyping. In Proceedings of the 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Boulder, CO, USA, 29 April–1 May 2018; p. 223. [CrossRef]
9. Xilinx Vitis AI. Available online: https://github.com/Xilinx/Vitis-AI (accessed on 27 February 2022).
10. Intel Open Vino Toolkit. Available online: https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html (accessed on 27 February 2022).
11. Lattice SensAI. Available online: https://www.latticesemi.com/sensAI (accessed on 27 February 2022).
12. Sharma, H.; Park, J.; Amaro, E.; Thwaites, B.; Kotha, P.; Gupta, A.; Kim, J.K.; Mishra, A.; Esmaeilzadeh, H. Dnnweaver: From High-Level Deep Network Models to Fpga Acceleration. *The Workshop on Cognitive Architectures*. 2016. Available online: http://www.act-lab.org/doc/paper/2016-cogarch-dnn_weaver.pdf (accessed on 1 April 2022).
13. Zhang, C.; Sun, G.; Fang, Z.; Zhou, P.; Pan, P.; Cong, J. Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks. *IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst.* **2018**, *38*, 2072–2085. [CrossRef]
14. Umuroglu, Y.; Fraser, N.J.; Gambardella, G.; Blott, M.; Leong, P.; Jahre, M.; Vissers, K. Finn: A framework for fast, scalable binarized neural network inference. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2017; pp. 65–74.
15. Guan, Y.; Liang, H.; Xu, N.; Wang, W.; Shi, S.; Chen, X.; Sun, G.; Zhang, W.; Cong, J. FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates. In Proceedings of the 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Napa, CA, USA, 30 April–2 May 2017; pp. 152–159.
16. Prakash, S.; Callahan, T.; Bushagour, J.; Banbury, C.; Green, A.V.; Warden, P.; Ansell, T.; Reddi, V.J. CFU Playground: Full-Stack Open-Source Framework for Tiny Machine Learning (tinyML) Acceleration on FPGAs. *arXiv* **2022**, arXiv:2201.01863.
17. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. Available online: tensorflow.org (accessed on 15 March 2022).
18. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*; Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2019; pp. 8024–8035.
19. Jia, Y.; Shelhamer, E.; Donahue, J.; Karayev, S.; Long, J.; Girshick, R.; Guadarrama, S.; Darrell, T. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv* **2014**, arXiv:1408.5093.
20. PYNQ—An Open Source Project from Xilinx. Available online: https://github.com/xilinx/pynq (accessed on 20 February 2022).
21. Xilinx Kria—Adaptive System-on-Module. Available online: https://www.xilinx.com/products/som/kria.html (accessed on 20 February 2022).
22. Xilinx Zynq UltraScale+ MPSoC. Available online: https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html (accessed on 20 February 2022).
23. Flamis, G.; Kalapothas, S.; Kitsos, P. Workflow on CNN utilization and inference in FPGA for embedded applications: 6th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM 2021). In Proceedings of the 2021 6th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM), Preveza, Greece, 24–26 September 2021; pp. 1–6. [CrossRef]
24. NVIDIA; Vingelmann, P.; Fitzek, F.H. CUDA, Release: 10.0.130. Available online: https://developer.nvidia.com/cuda-toolkit (accessed on 10 March 2022).
25. Xilinx Legacy DNNDK. Available online: https://www.xilinx.com/html_docs/xilinx2019_2/vitis_doc/ccz1607591898756.html (accessed on 28 February 2022).
26. Xilinx Getting Started with Ubuntu. Available online: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/2037317633/Getting+Started+with+Certified+Ubuntu+20.04+LTS+for+Xilinx+Devices (accessed on 5 March 2022).
27. Xilinx DPU on PYNQ. Available online: https://github.com/Xilinx/DPU-PYNQ (accessed on 5 March 2022).
28. Vitis AI Model Zoo. Available online: https://github.com/Xilinx/Vitis-AI/tree/v1.4/models/AI-Model-Zoo (accessed on 5 March 2022).
29. Van Rossum, G.; Drake, F.L. *Python 3 Reference Manual*; CreateSpace: Scotts Valley, CA, USA, 2009.
30. Kluyver, T.; Ragan-Kelley, B.; Pérez, F.; Granger, B.; Bussonnier, M.; Frederic, J.; Kelley, K.; Hamrick, J.; Grout, J.; Corlay, S.; et al. *Jupyter Notebooks—A Publishing Format for Reproducible Computational Workflows*; Positioning and Power in Academic Publishing: Players, Agents and Agendas; Loizides, F., Schmidt, B., Eds.; IOS Press: Amsterdam, The Netherlands, 2016; pp. 87–90.
31. Han, S.; Kang, J.; Mao, H.; Hu, Y.; Li, X.; Li, Y.; Xie, D.; Luo, H.; Yao, S.; Wang, Y.; et al. ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2017; Association for Computing Machinery: New York, NY, USA, 2017; FPGA '17; pp. 75–84. [CrossRef]
32. Kria SoM Getting Started. Available online: https://xilinx.github.io/kria-apps-docs/home/build/html/index.html# (accessed on 8 March 2022).

33. Deng, L. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Process. Mag.* **2012**, *29*, 141–142. [CrossRef]
34. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
35. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 1–9.
36. Maas, A.L.; Hannun, A.Y.; Ng, A.Y. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings ICML*; Citeseer: Princeton, NJ, USA, 2013; Volume 30, p. 3.
37. Bradski, G. The OpenCV Library. *Dr. Dobb's J. Software Tools* **2000**, *25*, 120–123.
38. Harris, C.R.; Millman, K.J.; van der Walt, S.J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N.J.; et al. Array programming with NumPy. *Nature* **2020**, *585*, 357–362. [CrossRef]
39. Jakob, W.; Rhinelander, J.; Moldovan, D. *pybind11—Seamless operability between C++11 and Python*; 2017. Available online: https://github.com/pybind/pybind11 (accessed on 14 March 2022).
40. Platform Assets Container. Available online: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/2057043969/Snaps+-+xlnx-config+Snap+for+Certified+Ubuntu+on+Xilinx+Devices#Platform-Assets-Container (accessed on 6 March 2022).
41. Zhang, Y.; Suda, N.; Lai, L.; Chandra, V. Hello edge: Keyword spotting on microcontrollers. *arXiv* **2017**, arXiv:1711.07128.
42. Warden, P. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv* **2018**, arXiv:1804.03209.
43. Huang, L.; Yang, Y.; Deng, Y.; Yu, Y. Densebox: Unifying landmark localization with end to end object detection. *arXiv* **2015**, arXiv:1509.04874.
44. Kaggle. Available online: https://www.kaggle.com (accessed on 8 March 2022).
45. MultiTask Model in the Vitis AI Library. Available online: https://docs.xilinx.com/r/en-US/ug1354-xilinx-ai-sdk/MultiTask (accessed on 15 March 2022).
46. Yu, F.; Chen, H.; Wang, X.; Xian, W.; Chen, Y.; Liu, F.; Madhavan, V.; Darrell, T. BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning. *arXiv* **2018**, arXiv:1805.04687. doi:10.48550/ARXIV.1805.04687.
47. Lee, S.; Kim, J.; Shin Yoon, J.; Shin, S.; Bailo, O.; Kim, N.; Lee, T.H.; Seok Hong, H.; Han, S.H.; So Kweon, I. VPGNet: Vanishing Point Guided Network for Lane and Road Marking Detection and Recognition. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 October 2017.
48. Caltech Lanes Dataset Includes Four Clips Taken Around Streets in Pasadena, CA at Different Times of Day. Available online: http://www.mohamedaly.info/datasets/caltech-lanes (accessed on 15 March 2022).
49. Redmon, J.; Farhadi, A. Yolov3: An Incremental Improvement. *arXiv* **2018**, arXiv:1804.02767.
50. Redmon, J. Darknet: Open Source Neural Networks in C. 2013–2016. Available online: http://pjreddie.com/darknet/ (accessed on 15 March 2022).
51. Cordts, M.; Omran, M.; Ramos, S.; Scharwächter, T.; Enzweiler, M.; Benenson, R.; Franke, U.; Roth, S.; Schiele, B. The cityscapes dataset. In *CVPR Workshop on the Future of Datasets in Vision, In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 8–12 June 2015*; IEEE: New York, NY, USA, 2015; Volume 2.
52. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
53. Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vis.* **2015**, *115*, 211–252. [CrossRef]
54. Kria™ KV260 Vision AI Starter Kit Benchmark. Available online: https://github.com/Xilinx/kria-kv260-ai-benchmark (accessed on 15 March 2022).
55. The NLP SmartVision Design Built on KV260 Vision AI Starter Kit. Available online: https://xilinx.github.io/kria-apps-docs/main/build/html/docs/nlp-smartvision/nlp_smartvision_landing.html (accessed on 15 March 2022).
56. Platformstats—A Linux Utility for Collecting Platform Statistics Including die Temperature, CPU Speed, Power Utilization. Available online: https://github.com/Xilinx/platformstats (accessed on 15 March 2022).
57. Instruments, Texas. *INA260 Precision Digital Current and Power Monitor with Low-Drift, Precision Integrated Shunt*. 2016. Available online: https://www.ti.com/product/INA260 (accessed on 15 March 2022).
58. Intel Neural Compute Stick. Available online: https://www.intel.com/content/www/us/en/developer/tools/neural-compute-stick/overview.html (accessed on 8 March 2022).
59. Qualcomm Snapdragon 870 5G Mobile Platform. Available online: https://www.qualcomm.com/products/snapdragon-870-5g-mobile-platform (accessed on 5 March 2022).