Algorithm-hardware Co-optimization for Energy-efficient Drone Detection on Resource-constrained FPGA

HAN-SOK SUH and JIAN MENG, Arizona State University, USA
TY NGUYEN and VIJAY KUMAR, University of Pennsylvania, USA
YU CAO and JAE-SUN SEO, Arizona State University, USA

Convolutional neural network (CNN)-based object detection has achieved very high accuracy; e.g., single-shot multi-box detectors (SSDs) can efficiently detect and localize various objects in an input image. However, they require a high amount of computation and memory storage, which makes it difficult to perform efficient inference on resource-constrained hardware devices such as drones or unmanned aerial vehicles (UAVs). Drone/UAV detection is an important task for applications including surveillance, defense, and multi-drone self-localization and formation control. In this article, we designed and co-optimized an algorithm and hardware for energy-efficient drone detection on resource-constrained FPGA devices. We trained an SSD object detection algorithm with a custom drone dataset. For inference, we employed low-precision quantization and adapted the width of the SSD CNN model. To improve throughput, we use dual-data rate operations for DSPs to effectively double the throughput with limited DSP counts. For different SSD algorithm models, we analyze accuracy or mean average precision (mAP) and evaluate the corresponding FPGA hardware utilization, DRAM communication, and throughput optimization. We evaluated the FPGA hardware for a custom drone dataset, Pascal VOC, and COCO2017. Our proposed design achieves a high mAP of 88.42% on the multi-drone dataset, with a high energy efficiency of 79 GOPS/W and throughput of 158 GOPS using the Xilinx Zynq ZU3EG FPGA device on the Open Vision Computer version 3 (OVC3) platform. Our design achieves 1.1 to 8.7× higher energy efficiency than prior works that used the same Pascal VOC dataset, using the same FPGA device, but at a low-power consumption of 2.54 W. For the COCO dataset, our MobileNet-V1 implementation achieved an mAP of 16.8, and 4.9 FPS/W for energy-efficiency, which is ~1.9× higher than prior FPGA works or other commercial hardware platforms.

CCS Concepts: • **Computer systems organization → Neural networks**; **Reconfigurable computing;**

Additional Key Words and Phrases: FPGA accelerator, object detection, algorithm-hardware co-design, neural networks

ACM Reference format:
Han-Sok Suh, Jian Meng, Ty Nguyen, Vijay Kumar, Yu Cao, and Jae-Sun Seo. 2023. Algorithm-hardware Co-optimization for Energy-efficient Drone Detection on Resource-constrained FPGA. *ACM Trans. Reconfig. Technol. Syst.* 16, 2, Article 33 (May 2023), 25 pages.
https://doi.org/10.1145/3583074

This work is partially supported by NSF grant 1652866, and C-BRIC, one of six centers in JUMP, a SRC program sponsored by DARPA.
Authors' addresses: H.-S. Suh, J. Meng, Y. Cao, and J.-S. Seo, Arizona State University, Arizona, 781 E Terrace Road, ISTB4 591, Tempe, AZ 85287, USA; emails: {hsuh6, jmeng15, Yu.Cao, jaesun.seo}@asu.edu; T. Nguyen and V. Kumar, University of Pennsylvania, 220 S. 33rd Street, Philadelphia, PA 19104, USA; emails: {tynguyen, kumar}@seas.upenn.edu.

1936-7406/2023/05-ART33 $15.00
https://doi.org/10.1145/3583074

1 INTRODUCTION

Convolutional neural networks (CNNs) have been very successful for many computer vision applications including image recognition, object detection, and localization. Object detection is a core computer vision task that is critical for autonomous driving, smart robotics, **unmanned aerial vehicles (UAVs)**, and more. Particular to UAVs, drone detection is an important task for applications including surveillance, defense, and multi-drone self-localization and formation control. While the state-of-the-art CNNs for object detection achieve very high **mean average precision (mAP)** for datasets such as Pascal VOC and Microsoft COCO, they still require millions of weights and billions of operations to obtain high mAP. On the other hand, UAVs operating on a battery exhibit stringent power/energy requirements, which prohibits a high degree of parallelism or a massive amount of storage for the compute hardware on UAVs. Nevertheless, a close to real-time object detection operation is required for making proper decisions for autonomous flights.

GPU is a popular hardware platform to perform object detection, benefiting from its massively parallel processing cores. However, due to high price and low energy efficiency, GPU is not an ideal solution for CNN inference acceleration, especially for edge devices or customized applications. ASICs have the highest energy efficiency, but their limited configurability can introduce a significant risk of premature obsolescence. With AI algorithms evolving at a fast pace, ASICs usually lag behind the cutting edge due to the long design cycle. To that end, FPGAs have a unique advantage with higher energy efficiency than GPUs, while offering faster time to market and potentially longer life cycle than ASICs.

The **Open Vision Computer (OVC)** platform was designed to support high-speed, vision-guided autonomous drone flight [20]. The particular objective was to develop a system that would be suitable for relatively small-scale flying platforms where size, weight, power consumption, and computational performance were all important considerations. Both the software and hardware resources are open sourced. Targeting drone detection tasks in this work, we employ the **OVC version 3 (OVC3)** system that can be attached onto UAVs. OVC3 includes a Xilinx Zynq Ultrascale+ SoC, with a quad-core ARM application processor and ZU3EG FPGA fabric. Compared to large-scale FPGAs that have thousands of DSP slices and hundreds of Mb of **block RAM (BRAM)**, ZU3EG is a resource-constrained FPGA that includes only 360 DSP slices, 7.6 Mb BRAM, and 70,560 **look-up tables (LUTs)**.

For the object detection algorithm, we employ the widely used **single-shot multi-box detector (SSD)** [13], which uses VGG-16 as the backbone CNN. In recent algorithm works [4], VGG-style CNNs have been revived to show favorable accuracy-speed tradeoff compared to state-of-the-art CNNs, where the regularity of using only 3×3 convolution kernels aids faster hardware speed. In addition, we also implemented the compact MobileNet-V1 backbone network to show object detection results on Pascal VOC and COCO datasets.

To that end, we first use the SSD model with VGG-16 as the backbone CNN in our experiments, while we explore different widths of VGG-16 CNN to show the tradeoff of model size, mAP, throughput, and energy efficiency. In addition, we implemented MobileNet-V1 in the same resource-constrained FPGA to evaluate and compare our design to prior works that used similar FPGAs or other commercial embedded hardware. For the VGG-16 backbone network, we trained a hardware-friendly variant of the original SSD model using the drone dataset presented in [17]. For the MobileNet-V1 experiment, we retrained the MobileNet-V1 SSD baseline model presented in [6] with the COCO dataset. For low-precision quantization, we propose UniPOT, a uniform/unified quantization algorithm with **power-of-two (POT)** quantization boundary, which avoids the use of high-precision scaling factors throughout the CNN model and simplifies the FPGA hardware implementation. Considering the SSD model mapping onto the resource-constrained FPGA, we

observed that using 8-bit precision with the UniPOT scheme results in better tradeoff in hardware utilization and mAP, compared to 4-bit or lower quantization with more complex quantization schemes such as **additive-power-of-two (APOT),** [11] which requires high-precision scaling factors that consume precious hardware resources.

On the hardware side, we also performed a number of optimizations with the resource-constrained Xilinx ZU3EG FPGA device. We first maximized the utilization of parallel convolutions within the available 360 DSP slices and employed a dual-data rate DSP design to double the throughput. Subsequently, across three SSD models with different widths, we optimized the maximum amount of on-device activation/weight storage by using both BRAM and LUTs.

Overall, the main contributions of this work are:

- We present an energy-efficient drone detection accelerator on a resource-constrained FPGA, which is part of the OVC3 platform built for autonomous drone flight.
- On the algorithm side, we trained VGG-based SSD with multi-drone dataset, using a uniform/**unified quantization scheme (UniPOT)** for efficient FPGA mapping.
- As a comparable result to other work, we offer results from MobileNet-V1 with the COCO dataset experimented with our hardware.
- On the hardware side, we optimized the DSP/memory utilization in resource-constrained FPGA across different SSD models, employed dual-data rate DSP design to double the throughput, and reduced DDR latency aided by DMA descriptor buffer design.
- We demonstrated drone detection on the Xilinx ZU3EG FPGA and analyzed mAP, throughput, and energy across three SSD model widths (1.0×, 0.5×, and 0.25×).
- Our 0.5× model implementation achieves 57.8 GOPS/W energy efficiency, 150 GOPS throughput, and 83.9% mAP, showing favorable tradeoff compared to prior works.
- For general objection detection on the COCO dataset, our FPGA design shows ~1.9 to 14.94× higher FPS/W compared to prior FPGA works and commercial embedded hardware with minimal mAP degradation.

2 ALGORITHM OPTIMIZATION

2.1 Custom SSD Model Adaptation

In this work, we adopt SSD300-HW [15], a variant of SSD300x300 [13] with small modifications for hardware-friendly purposes. Particularly, in layer fc6, the dilation value is changed from 6 to 1 to focus more on small objects, which also account for the majority of objects in our target drone dataset [17]. At the end of the conv4_3 layer, a single scale factor for the layer normalization step is shared among all channels to avoid complexity in the hardware implementation. On top of these modifications, we additionally remove layers conv9_1 and conv9_2, as well as other layers that take only the output of these two layers as the input because the receptive field after conv9_2 is 300x300, which is too large for our application.

Although the full-size SSD model showed high accuracy, since the model had a high number of operations (>60 billion operations), we also investigated shrinking the size of the model by adopting the width multiplier [6] to the VGG CNN, toward achieving better, higher throughput. In particular, we first trained the narrower VGG-16 model (e.g, 0.5×, 0.25×) with the ImageNet dataset and subsequently cascaded the pre-trained CNNs to the full-size (1.0×) SSD model. Table 1 shows the SSD300HW network structure.

To obtain optimal prior boxes for the SSD model, we also use **k-nearest neighbor (KNN)** clustering as suggested in [13]. We increase the number of clusters from 1 to 20, cluster the dimension values of ground-truth boxes in the training dataset, and calculate corresponding clustering accuracies. This procedure is stopped when the clustering accuracy starts becoming saturated. Based

Table 1. SSD300HW Network Structure with the Number of Input/Output Channels

Layer Type	Kernel	Output Channel	Input Channel	Feature Size
Input		3		300
Conv1_1	3	64	3	300
Conv1_2	3	64	64	300
Conv2_1	3	128	64	150
Conv2_2	3	128	128	150
Conv3_1	3	256	128	75
Conv3_2	3	256	256	75
Conv3_3	3	256	256	75
Conv4_1	3	512	256	38
Conv4_2	3	512	512	38
Conv4_3	3	512	512	38
Conv5_1	3	512	512	19
Conv5_2	3	512	512	19
Conv5_3	3	512	512	19
Fc6	3	1024	512	19
Fc7	1	1024	1024	19
Conv6_1	1	256	1024	19
Conv6_2	3	512	256	19
Conv7_1	1	128	512	10
Conv7_2	3	256	128	10
Conv8_1	1	128	256	5
Conv8_2	3	256	128	5
Conv9_1	1	128	256	3

on this procedure, we select the set 13 boxes that achieves 86.60% in clustering accuracy. We further remove 2 boxes whose dimensions are almost identical, resulting in 11 prior boxes. Unlike the implementation in [13], we do not flip these prior boxes because it is unnecessary.

2.2 MobileNet-V1 Model

In the case of mobile devices like drones, it is necessary to compress a network model further, since the number of parameters in the SSD300 model might not fit on the target device. In this respect, we also implemented the MobileNet-V1 with fewer parameters in our FPGA, which will aid a more compact hardware implementation. Table 2 shows the MobileNet-V1 network structure.

When we use 8-bit precision for activation/weight quantization, the storage requirement for the weights of SSD300HW is 22.0 MB and for the weights of MobileNet-V1 it is 3.15 MB. Compared to SSD300HW, MobileNet-V1 reduces the weight storage requirement by ~86% and also largely reduces the computational complexity, which overall helps reduce the FPGA resource utilization.

2.3 Low-precision Quantization

The VGG-based SSD model with the ImageNet-trained convolutional feature extractor has been the best off-the-shelf SSD model, achieving >77% mAP on the Pascal VOC dataset. Such superior inference performance involves a large amount of computation and storage (~60 GOPs and ~138M weights), which makes it expensive for hardware deployment. To bridge this gap, we applied the

Table 2. MobileNet-V1 Network Structure with the Number of Input/Output Channels

Layer Type	Kernel	Output Channel	Input Channel	Feature Size
Input		3		224
Conv1/s2	3	32	3	224
Conv1 dw/s1	3	32	32	112
Conv2/s1	1	64	32	112
Conv2 dw/s2	3	64	64	112
Conv3/s1	1	128	64	56
Conv3 dw/s1	3	128	128	56
Conv4/s1	1	128	128	56
Conv4 dw/s2	3	128	128	56
Conv5/s1	1	256	128	28
Conv5 dw/s1	3	256	256	28
Conv6/s1	1	256	256	28
Conv6 dw/s2	3	256	256	28
Conv7/s1	1	512	256	14
Conv8 dw/s1 x5	3	512	512	14
Conv8/s1 x5	1	512	512	14
Conv9 dw/s2	3	512	512	14
Conv9/s1	1	1,024	512	7
Conv10 dw/s2	3	1,024	1,024	7
Conv10/s1	1	1,024	1,024	7
FC/s1	1	1,000	1,024	1

low-precision quantization to the entire SSD model to alleviate the hardware resource consumption while maintaining high inference accuracy.

Generally, given the floating-point weight W, in-training uniform quantization can be formulated into the steps below:

$$W_c = \min(\max(W, -a), a) \quad \text{Clipping} \tag{1}$$

$$S = \frac{2^{n-1} - 1}{a} \quad \text{Scaling} \tag{2}$$

$$W_Q = \text{round}(W_c \times S) \quad \text{Quantization} \tag{3}$$

$$W_{QF} = \frac{W_Q}{S} \quad \text{De-quantization} \tag{4}$$

Many recent quantization algorithms aggressively reduce the bit precision (e.g., sub-4-bit) to lower the total storage and number of operations. However, the low-precision uniform quantization algorithms in the literature often require high-precision scaling [3, 7, 18] or extra preprocessing steps [12], which causes hardware resource over-utilization when such algorithms are fully implemented onto hardware devices.

In comparison to uniform quantization, the POT quantization converts the multiplication into a shifting operation and substantially simplifies the hardware. However, the lopsided resolution of the POT quantization degrades the accuracy. To address the accuracy degradation, APOT [11] has been proposed to quantize the weight and activation into 2^n levels where each level is the sum of multiple POT terms. Given the number of additive terms, the digitized levels are deterministic, which can be saved into LUTs. Then, we need to place a high-precision comparator to choose the

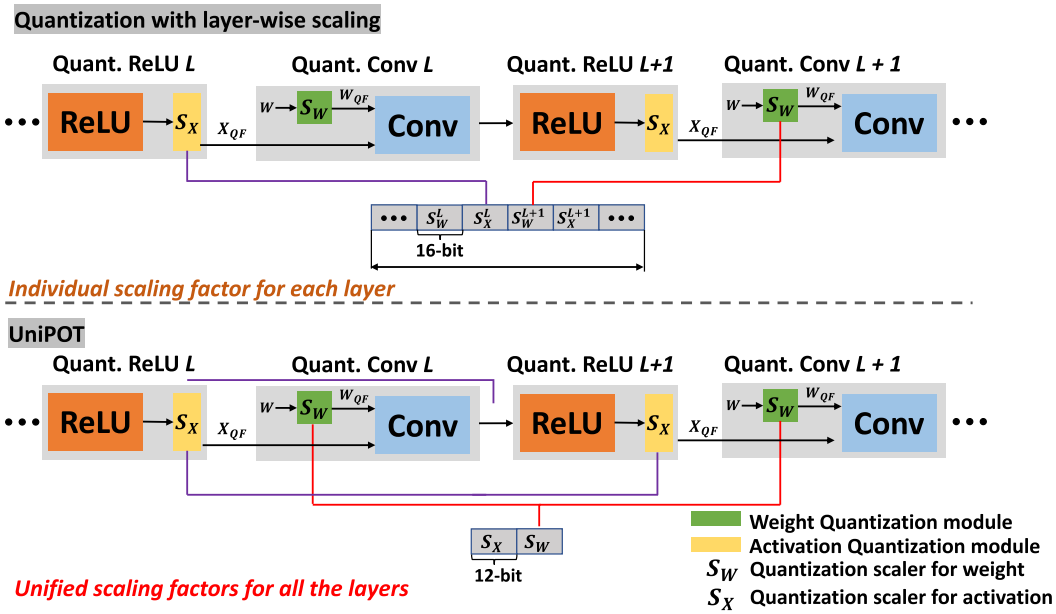


Fig. 1. Comparison of other low-precision quantization and the proposed UniPOT quantization schemes.

Table 3. mAP Evaluation of SSD Model (1.0× Width) on the Multi-drone Dataset with Different Weight/Activation Quantization Schemes

Method	W/A	Precision	mAP (%)	Scaling
Baseline	32/32	32 bit	89.64	-
PACT [3]	4/4	4 bit	82.37	Layer-wise
APOT [11]	4/4	5–7 bit	87.42	Layer-wise
UniPOT (This work)	4/4	4 bit	69.47	**Unified**
UniPOT (This work)	8/8	8 bit	**89.40**	**Unified**

quantization level of each variable. Furthermore, layer-wise learnable scaling factors that require a high-precision multiplier are employed in the APOT scheme.

In this work, we propose UniPOT, a uniform and unified quantization algorithm with the POT quantization boundary. Given the pre-trained DNN model, the weight and activation will be clipped by a tunable POT value, $a_w, a_x \in \{1/2, 1/4, 1/8, \ldots, 8, 16\}$. The selected quantization boundary will be applied to all the layers of the network. Therefore, the quantization scaling factor will have limited data precision and get broadcast to the entire DNN model (Figure 1). Constraining the distribution of weights and activations by the unified POT value can avoid the high-precision scaling in Equation (2) and subsequently simplifies the hardware implementation.

Table 3 summarizes the software performance of VGG-based SSD on the multi-drone dataset [17] by applying different quantization strategies. Uniformly quantizing the model down to 4-bit [3] leads to significant accuracy degradation. Deploying the APOT-quantized model requires layer-wise scaling and high data precision with noticeable accuracy degradation.

Table 4 summarizes the hardware resource consumption of supporting different layer-wise scaling schemes, based on our experimental implementation on the ZU3EG FPGA device. The

Table 4. PE-wise Hardware Resource Consumption for Layer-wise High-precision Scaling with the Proposed UniPOT Algorithm

Method	LUTs	FFs	DSPs	Cost/Multiplier
UniPOT (This work)	111	100	1	**1×**
APOT [11]	345	159	1	2.38×

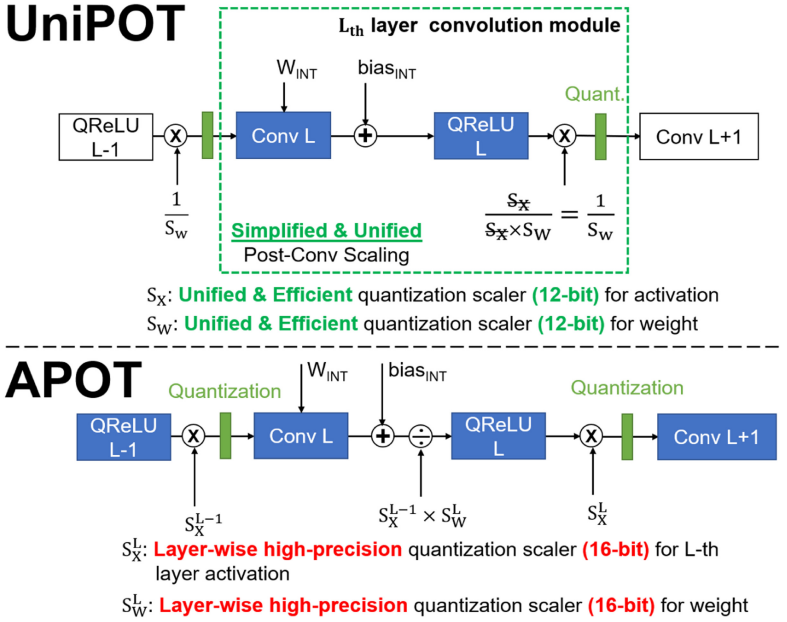


Fig. 2. Hardware implementation of two quantization methods.

APOT [11] quantization algorithm treats the layer-wise quantization boundary as the trainable parameters to minimize the quantization error. Such layer-wise adaptive quantization boundary further leads to the distinct post-convolution scaling factors, as depicted in Figure 1 (top). However, implementing the layer-wise high-precision multiplication of APOT [11] exceeds the on-chip resource budget of the selected ZU3EG FPGA device, as summarized in Table 4. On the contrary, the proposed UniPOT algorithm simplifies the post-convolution scaling process with unified scaling factors, as shown in Figure 2. With the 8-bit precision for both activation and weights, UniPOT achieves the optimal tradeoff between accuracy and reduction of hardware resource consumption. Compared to the APOT-based [11] layer-wise scaling, the proposed UniPOT algorithm reduces the resource consumption by 2.38×.

We use our own version of APOT quantization in Table 4 because we wanted to show that our UniPOT is more hardware friendly than any other quantization method in the literature while quantization is the key to successful implementation of the digital hardware accelerator, which must use limited bit-width to represent any values in the neural network. To properly compare our implementation to other works, we need to get the resource utilization of 1 PE in LUTs and RAMs with the same type of FPGA. However, this has not been reported in most prior works. Thus, we used our own version of APOT quantization in the comparison, which we implemented by finding the best optimized quantization algorithm because even hardware-friendly APOT quantization

that only uses shift registers for multiplication was not enough to be fitted into our target FPGA device due to the existence of a high-precision extra multiplier for the scaling factors.

Noticeably, supporting the layer-wise high-precision multiplication of APOT [11] causes 2.38× resource overhead compared to the proposed UniPOT algorithm. Given a total of 360 DSPs with limited resources, it becomes very challenging to support the expensive APOT algorithm with the ZU3EG FPGA device. Therefore, we select UniPOT with 8-bit precision for both weight and activation to guarantee high inference accuracy while maintaining the hardware simplicity.

As shown in Figure 1, each **quantized-ReLU (QReLU)** layer including a regular ReLU function and an activation quantization module (Equations (2) to (4)) generates the digitized convolution output [9]. Here, we use 12 bits to represent the activation quantization scaler and 16 bits for the weight quantization scaler.

Similar to the offline integer transformation in the prior works [7, 26], the de-quantization scaling factors for activations/weights in Equation (4) can be extracted and folded into the scaler that belongs to the next layer. This means that instead of computing Equation (5), we compute Equation (6). We use the linearity of the QReLU layer and pass the divider operation of the scaler to the post-QReLU layer side, so that we only have one scaler multiplication for each convolution layer:

$$OF_{Clip} = \text{QReLU}(OF * (1/S)), \tag{5}$$

$$OF_{Clip}/S = \text{QReLU}(OF), \tag{6}$$

where OF is the output feature map and OF_{Clip} is the clipped version of the output feature map. Overall, simplifying the quantization process minimizes the number of multipliers/dividers and reduces the total DSP usage.

3 FPGA HARDWARE DESIGN AND OPTIMIZATION

Xilinx Zynq ZU3EG FPGA in the OVC3 system is our target hardware device. ZU3EG FPGA only has 360 DSP slices and 7.6 Mb of BRAM, and these resources are less than those of large-scale FPGAs by an order of magnitude.

3.1 Overall Hardware Architecture

Figure 3 shows the overall hardware block diagram and dataflow. To simplify implementation and to separate read DMA operations from write DMA operations, we are using two DMA modules with a dedicated DMA descriptor buffer for each of them. Once the read DMA module reads input image tiles and weights from the memory, it will be written to the input buffer and weight buffer. Before pixels and weights are fed into MAC arrays, there is a data router that will rearrange pixels and weights into orders to maximize the reuse of input feature maps. In our data router design, FIFOs are employed to reuse pixels that will be fed into registers that are directly connected to MAC arrays. Each FIFO takes pixels from a register that holds the next row of the feature map. With this design, we just need to read pixels from input pixel buffers at the very first time of MAC array computation. After that, we shift and load pixels within register arrays until the kernel screen reaches the end of *Poy* computation. Then, the FIFO feeds pixels from the adjacent register arrays without needing to read these pixels from the input pixel buffer. Subsequently, the MAC array will get these data to perform convolution and accumulation. Each PE in the MAC array will calculate one output pixel in an output stationary dataflow. The architecture of PEs that support the output stationary dataflow is shown in Figure 4, where each PE performs one MAC operation per cycle.

We adapted the loop unrolling and loop tiling strategy introduced in [14]. Table 5 describes the terminologies for the CNN algorithm and FPGA design parameters used in this work. Adjusting

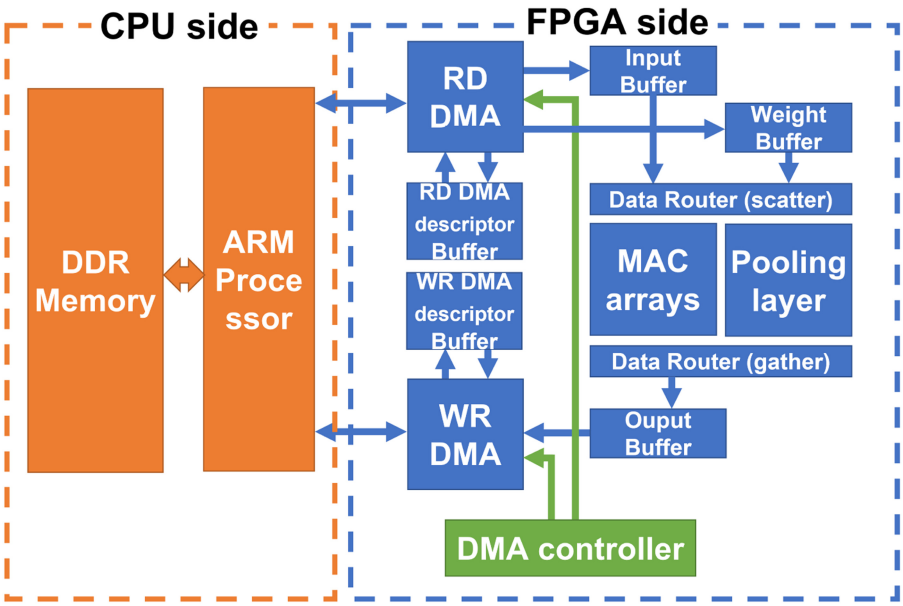


Fig. 3. Overall architecture of FPGA accelerator.

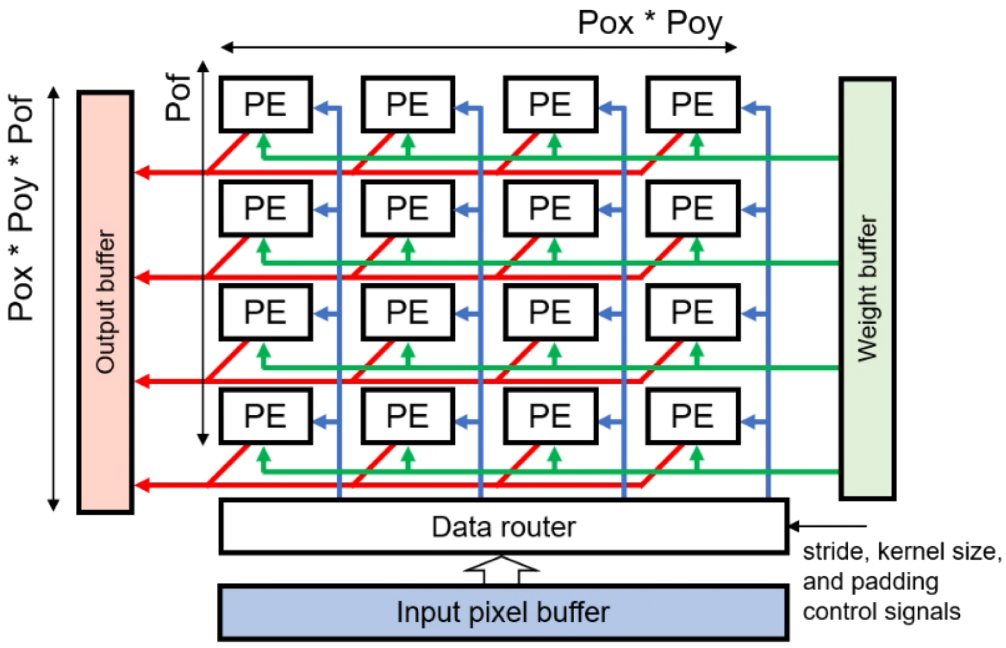


Fig. 4. Architecture of PE array for output stationary dataflow.

these hardware design parameters directly affects the throughput and latency results of our CNN accelerator.

For input buffer tiling, we tile input feature maps in the Tiy dimension only, as shown in Figure 5. Once the MAC operation is done, the output feature map will be stored in an output buffer with a

Table 5. Description of CNN Algorithm and FPGA Design Parameters

	Kernel (Width/Height)	Input Feature Map (Width/Height)	Output Feature Map (Width/Height)	# of Input Feature Maps	# of Output Feature Maps
Convolution Dimensions (N^{**})	Nkx, Nky	Nix, Niy	Nox, Noy	Nif	Nof
Loop Tiling (T^{**})	Tkx, Tky	Tix, Tiy	Tox, Toy	Tif	Tof
Loop Unrolling (P^{**})	Pxk, Pky	Pix, Piy	Pox, Poy	Pif	Pof

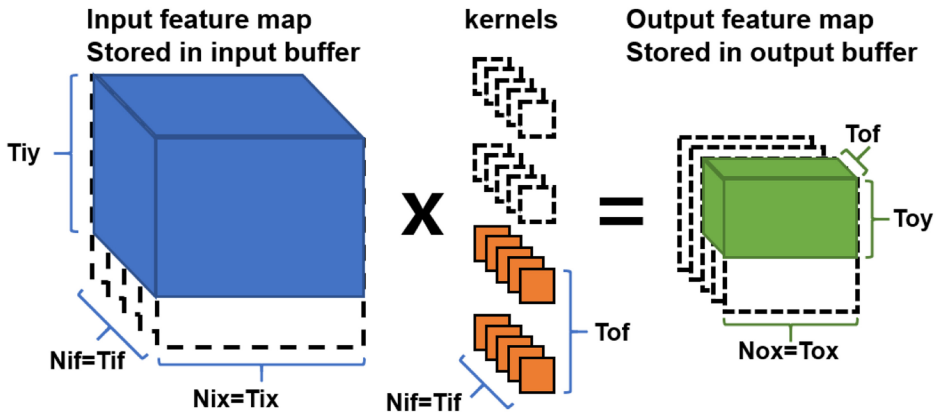


Fig. 5. Loop tiling strategy of our proposed architecture. Adapted from [14].

Tof and Toy tiling strategy (Figure 5). In our tiling scheme, Nif is equal to Tif, which means that we read all the necessary input feature maps that are required in accumulation and perform the summation in the DSP without stalling or reading the next batch of input feature maps to continue the accumulation. Thus, one iteration of the MAC operation will calculate and store $Pox \times Poy \times Pof$ amount of output feature map pixels to the output buffer. Since we are not tiling pixels in the Nix dimension, Nox tiling is not used and Nox is equal to Tof. Once $Nox \times Tof \times Toy$ amount of output pixels are ready in the output buffer, the write DMA process will start writing these pixels into DDR memory.

To maximize the hardware performance in the resource-constrained FPGA, we investigated the following design methodologies and strategies, which will be described in detail in the rest of the section.

- Comprehensive design space exploration
- Dual-data rate DSP design
- Small **direct memory access (DMA)** descriptor buffers.

3.2 Design Space Exploration

In this sub-section, we explain how we optimized our baseline architecture design to reduce the size of the design while getting the best performance out of it. The entire design process we used in our work is given in the Figure 6. The steps we used in our design space exploration are given below:

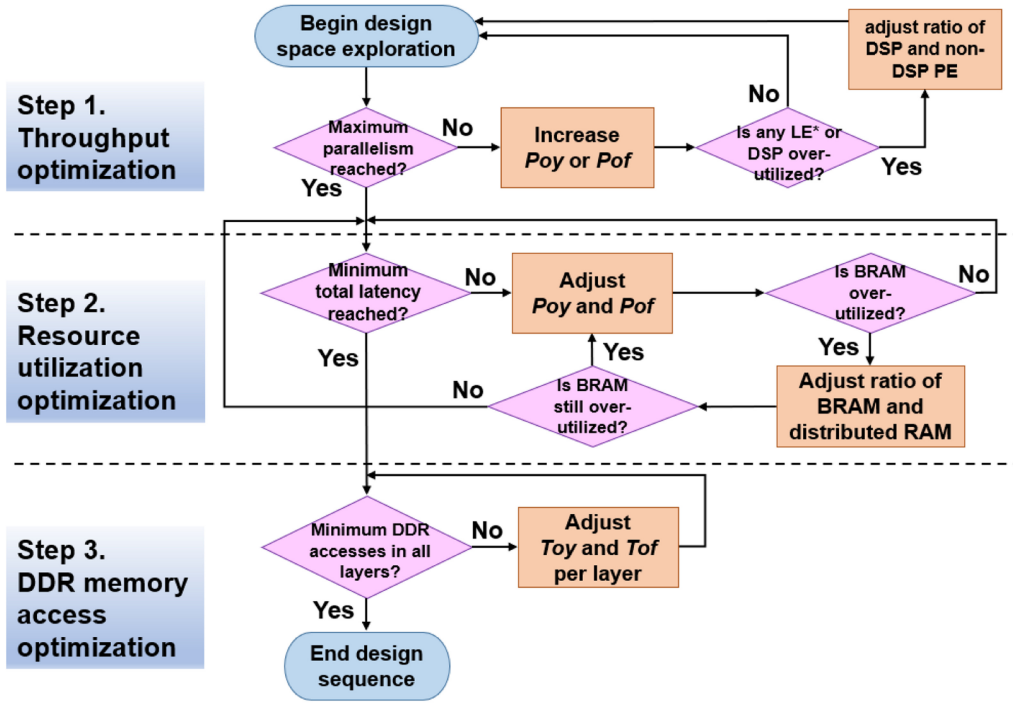


Fig. 6. Flow chart of our proposed design space exploration.

- Step 1: Increase parallelism by increasing number of MAC units in the design. This will achieve maximum PE counts, which equals $Pox \times Poy \times Pof$ from our design parameters. Test if any logic elements or DSPs are over-utilized and, if so, adjust the ratio of non-DSP PE and DSP. Here, non-DSP PE means a PE that only uses logic elements.
- Step 2: With fixed parallelism, find the best ratio of distributed RAM vs. block RAM to make the design fit into target FPGA. Also, test different (Pox, Poy, Pof) combinations to find the best performance in terms of total latency.
- Step 3: Adjust the loop tiling size of Toy and Tof to reduce the number of DDR memory accesses.

3.2.1 Step 1: Throughput Optimization. First, we attempt to employ the highest $Pox \times Poy \times Pof$ number, which will result in the highest parallelism of MAC arrays implemented by both DSP slices and non-DSP PE using logic elements in the FPGA. We employed a design parameter α that determines how many PEs in the MAC array should be implemented using DSP, and the rest of the PEs will be implemented using logic elements. However, our accelerator using UniPOT is implemented by using only DSPs, because we found it has better hardware utilization when our PEs are all implemented using DSPs, since only a few shift register/constant multipliers are needed for scaling factor multiplication. But in the case of APOT quantization, which uses extra multipliers for scaling factor multiplication, we can use this parameter to decide whether we will use DSPs to implement such extra multipliers. In our target FPGA, 360 DSP resources are available. Since the largest POT less than 360 is 256, the number of parallelism (loop unrolling) in our design is 512 ($2\times Pox\times Poy\times Pof$), as we can use up to 256 DDR DSPs. Also, in any step of the optimization process, if we searched the entire design space and a design that fits onto FPGA does not exist, then we delete that combination of Poy and Pof from the design space and start the DSE search over again.

3.2.2 Step 2: Resource Utilization Optimization. Once we found the upper limit of the maximum parallelism, we searched the best combination of Pox, Poy, and Pof that can result in the lowest total latency among possible choices of combination. Then, we test if the BRAM is over-utilized in the design. If the selected Poy and Pof combination uses too much BRAM, we adjust the parameter β to move a certain amount of buffer capacity from BRAM to distributed RAM, which uses LUT and LUTRAM to implement on-chip memory. This is the most important part of our DSE search process, and to the best of our knowledge, our method is the only DSE that automatically adjusts this ratio to squeeze every on-chip resource out from FPGA. During the search, Pox is considered as a constant because we are not tiling over the x dimension in the input feature maps and thus Pox is only dependent on the DMA bit-width to read as much as data from DDR memory. For input pixels, we need $Pox \times Poy \times [pixel\ bitwidth] = 8 \times 4 \times 8\text{-bit} = 256$ bits, since we need $Pox \times Poy$ pixels at a time in PE arrays. On the other dimension of 2D PE array, weights are being fed and we need $Pof \times [weight\ bitwidth] = 16 \times 8\text{-bit} = 128$ bits amount of data. On the DDR memory side, we are using 1.2 GHz as an operating frequency of DDR memory, and according to Xilinx, we can calculate speed of DDR memory below:

$$[Data\ Rate] \times [Number\ of\ DDR\ Interfaces] \times [DDR\ Interface\ Width] = 2.4\text{GHz} \times 2 \times 32bits = 150Gbps. \tag{7}$$

Therefore, we have sufficient bandwidth on the DDR memory side, but we only have two physical high-performance AXI ports in our FPGA that is connected to the DDR memory. Each AXI port has 256 bits of bandwidth in our design, and it is being used to feed pixels and weights to PE arrays and obtain outputs out of the PE arrays. Therefore, we have to wait a few cycles while we copy one tile of pixels and weights to the input buffer and weight buffer, and the real calculation in PE arrays starts once the input and weight buffer are filled with tiles. Similarly, once computation for all tiles in the input/weight buffer is done, the output buffer starts writing results to the DDR memory and we can copy another set of input and weight tiles while it is copying results to DDR memory. Since we do not have a ping-pong buffer in our design, PE arrays must be halted while we are reading/copying data from/to DDR memory.

This means we need $4 \times Pox \times$ 8-bit (activation bit-width) of bandwidth, where our DMA bandwidth is 256 bits. Therefore, the optimal choice of Pox is 8 in our design. To decide how much parallelism will be in Poy and Pof, we tried to assign more parallelism to a design parameter that comes with smaller buffer size to reduce BRAM resource utilization in FPGA.

In Figure 5, the input buffer is always larger than the weight and output buffers, because we are tiling over the Nof dimension. Even if the input channel and output channel sizes are identical, Tof tiling reduces the amount of weights and outputs that need to be minimally stored in each buffer. On the other hand, Nif is not tiled and thus the input buffer is relatively larger than the other two buffers. Now, if we increase Toy, this will increase Tiy and the input buffer capacity will increase to store more tile pixels. Instead of increasing Poy, which will proportionally increase Toy and the input buffer size, we first increased Pof (which does not increase the input buffer size), and then the rest of parallelism was assigned to Poy. We swept a number of combinations of (Pof, Poy) sets, and (16, 2) was the best design parameter setting that resulted in the lowest total latency.

Once the adjustment of Poy/Pof and the adjustment between BRAM usage and distributed RAM usage is done, the total latency of our FPGA design is estimated and the design search continues. In our design, the latency for one convolution layer is composed of input buffer latency and MAC array calculation time. The former is the number of input image tiles in a convolution times the latency that it takes to process a given tile. The latter can be calculated by ($Tif \times Tox/Pox \times Toy/Poy \times Tof/Pof \times \#\ of\ input\ image\ tiles$), which provides a good estimation on the number of

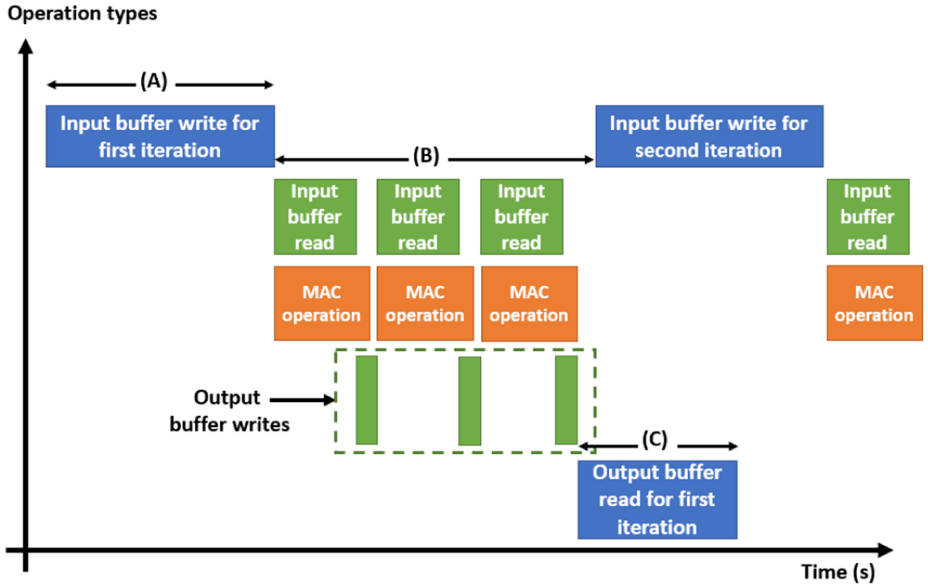


Fig. 7. Timing diagram of MAC array operation including DRAM communication.

cycles consumed in the MAC array. By summing up the latency of all layers, we can obtain the total latency. Pox, Poy, and Pof of our design are optimally chosen as 8, 4, and 16, respectively.

3.2.3 Step 3: DDR Memory Access Optimization. Figure 7 shows the latency breakdown for one iteration of MAC array computation. Upon completion of Pox, Poy, and Pof combination search for the best total latency, we found that our design has a bottleneck in the DDR memory performance. The latency of (A), (B), and (C) varies for different convolution layers. In the cases of the conv4_2 and conv4_3 layers, our most time-consuming convolution layers, (A) consumed higher latency than (B) or (C), which was caused by the bottleneck from DDR memory read speed. In our initial latency measurement, input buffer write latency (A) consumed >70% of total latency in the conv4_2 layer. During input buffer write, the read DMA module was spending too much time waiting on data from the DDR memory. To alleviate this bottleneck, we tried to reduce the DDR memory accesses by increasing the tile size of feature maps. In our CNN accelerator, this can be achieved by increasing either the Toy or Tof design parameter without increasing the size of input/output/weight buffers. For fine-grained optimization, we fine-tuned Toy and Tof for each layer of the CNN.

For degrees of freedom in the optimization factor, we have a number of parameters that can be adjusted, but we mainly tuned Poy, Pof, Toy, and Tof for design space exploration. A constraint exists that Toy and Tof must be integer multiples of Poy and Pof, respectively. Also, design parameters α and β cannot exceed 1 because they decide the percentage of modules that will be moved from one to the other logic resources, e.g., logic elements or distributed RAM.

Techniques such as double-buffering could further increase the throughput by hiding the communication latency to/from DDR memory, which will be possible for some larger FPGA devices. However, our target FPGA device, ZU3EG, only has 1.175 MB of total on-chip memory, while our design uses both LUTRAM and BRAM for input and output buffers. Both of them should be doubled to integrate double-buffering into our target FPGA, but the total on-chip memory was not sufficient to enable this. Balancing the limited on-chip resources to make enough space for both

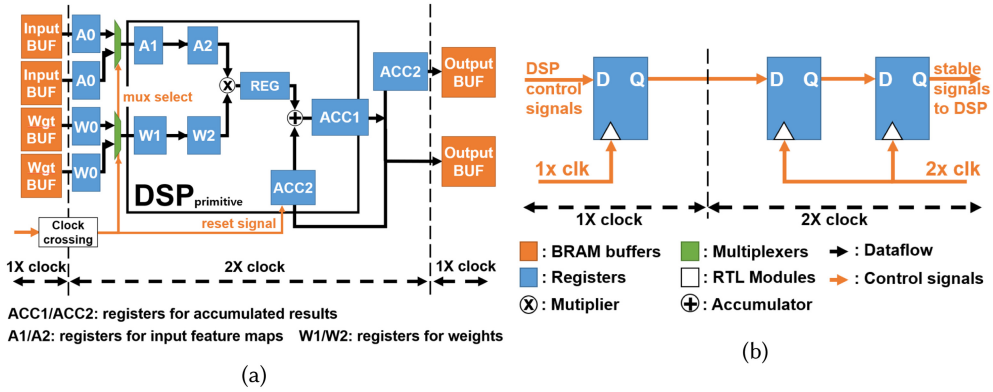


Fig. 8. (a) Dual-data-rate DSP implementation. (b) Clock-crossing logic used in DDR DSP control signals.

LUTRAM and BRAM was also limited, because we found that faster BRAM should be used in some buffers to avoid critical path problems. To that end, double-buffering was not employed for our FPGA design with the ZU3EG device.

3.3 Dual-data-rate DSP Design

For DSP count reduction, we use dual-data-rate DSP, which can feed two data into one DSP at a time and generate two outputs at the same time, as shown in Figure 8(a). The DSP slices in our FPGA design work at 400 MHz of frequency, twice the operation frequency (200 MHz) of the main CNN accelerator. Although DDR DSP is standard in Xilinx's **deep learning processing unit (DPU)** IP [23], how it is configured is not published. By referring to the IP diagram, we implemented our own custom design of dual-data-rate DSP, by adding time-matching flip-flops and clock-crossing logics before and after the DSP48E2 primitive (Figure 8(a)). The two sets of registers on the input side are input pipeline registers. Output data lines are separated and fed back to DSP again for accumulation.

A0/A1/A2 are pipeline registers that can latch the activation data that gets conveyed to the DSP module. Similarly, W0/W1/W2 are pipeline registers that latch the weight data. For everything that is in the "DSP" box in Figure 8(a), we used integrated registers and multipliers inside the DSP to set up the internal DSP configuration. For example, ACC1 and ACC2 are pipeline registers that latch accumulated output data at the end of calculation, where we placed two registers so they can hold two independent results in each register in a time-multiplexed manner with our custom control logic for DDR-DSP.

The number of stages in the pipeline registers is determined based on the information in the Xilinx technical manual. Clock-crossing logics are inserted into the control signals, such as the accumulator reset signal and multiplexer select signals. Figure 8(b) shows how flip-flops are inserted to prevent signal meta-stability in the frequency-crossing domain. We could increase the effective DSP counts by 2× using this technique.

In Table 6, we compared our technique to other DSP packing techniques [8, 16]. [8] reduces the number of DSPs the most, but they need to place extra on-chip memory to store the multiplier parameters. In addition, the technique in [8] uses extra hardware resources for the concatenation of inputs and shift registers for post-processing, and an extra accumulator also exists outside of the DSP using LUTs at the end. This is why their implementation overall uses a high amount of resources. Regarding [16], even though it is stated that $4n$ bits are enough to do a double multiplication in one DSP, there is still a chance of overflow due to the large accumulation in convolution

Table 6. Resource Utilization Comparison of DSP Packing Techniques for 1 PE with 8-bit Quantization

	LUT	DFF	DSP	BRAM
[8]	57.06	64.19	0.33	0.48
[16]	11	12	0.5	0
Ours	9	29.5	0.5	0

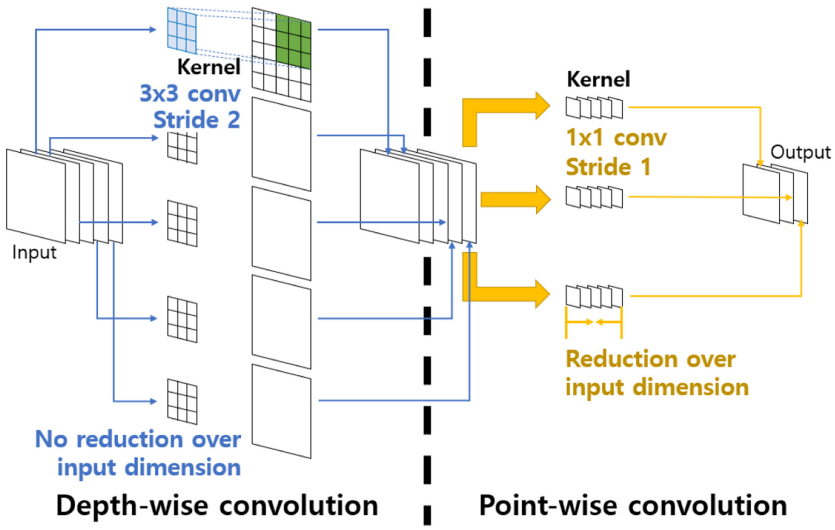


Fig. 9. Depth-wise separable convolution layer split into two convolution operations.

layers with large output channels. Depending on how many output channels exist in the convolution layer, we may need more than $4n$ bits. For example, if we continuously multiply $1000000_{(2)}$ with $1000000_{(2)}$ and accumulate them 1,024 times, this leads to 24 bits for the resulting binary number. To avoid the overflow in this case, 24 + 1 (guard bit) + 24 = 49-bit accumulator is required, and this is impossible in our DSP, which only has 48-bit accumulator. On the contrary, our proposed technique only uses one DSP with a few additional registers for clock crossing and pipeline registers; hence, our implementation shows the least amount of hardware resource consumption without the need for any accumulator outside of the DSP.

3.4 MobileNet-V1 HW Implementation

To obtain the best performance while using resources on FPGA as little as possible, we also implemented MobileNet-V1 CNN. For this, a **depth-wise separable convolution (DSConv)** layer and its corresponding hardware were added to the DSP design. This DSP design also benefits from our DDR DSP design by accommodating an accumulation pattern in the DSConv layer, which is somewhat different from normal convolution layers.

In the DSConv layer, there are two convolution layers; one is the depth-wise convolution layer and the other is the point-wise convolution layer. To perform the DSConv operation without adding any extra PEs and DSPs to the design, we separated a DSConv layer into a **depth-wise convolutional layer (DWConv)** and **point-wise convolutional layer (PWConv)**, as shown in Figure 9. With this strategy, we can reuse the same DSP architecture for these consecutive

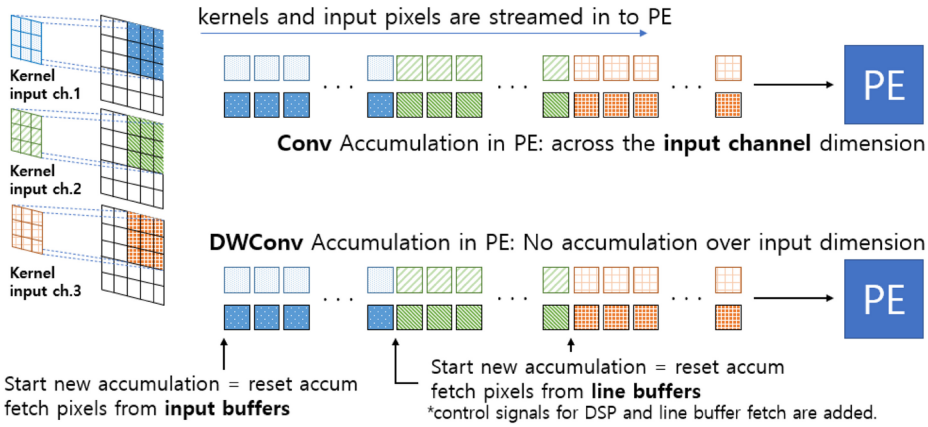


Fig. 10. DWConv layer input feeding pattern compared to the normal convolution layer.

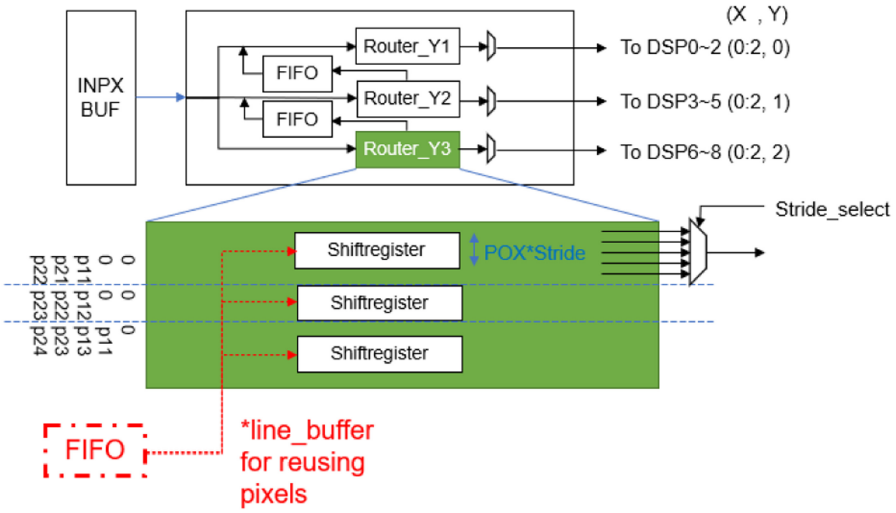


Fig. 11. Structure of line buffer for pixel reuse in depth-wise convolution (*Pox*=3, *Poy*=3).

convolution calculations. This is important for applications with limited logic resources such as a small FPGA. Also, there are two notable differences in these layers compared to plain convolution layers. One is that the DWConv layer does not do any accumulation across the output channel dimension after the convolution calculation. Considering this, we came up with a new DSP design that can skip and reset accumulation in DSPs. Our strategy of feeding inputs to the DSPs for the DWConv layer is shown in Figure 10.

For this new input pattern, we implemented the pixel reuse modules additionally for the DW-Conv operation. This will also reduce the DDR latency and buffer-to-PE data movement by reusing the pixels that have been already fetched. The structure of this design is shown in Figure 11. With these line buffers, we can feed the next pixels to PEs without accessing input buffers or DDR memory. This means that the buffer needs to hold 2× more pixels than the *Pox* dimension in both line buffers (FIFOs) and a pixel buffer to feed in initial pixel tiles to line buffers. While we are increasing the buffer size by 2× in the direction of *Pox*, this can be affordable for MobileNet-V1, due to the low

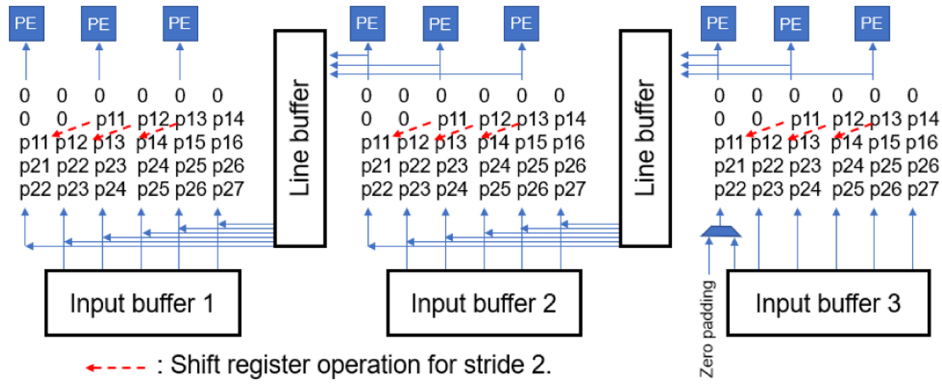


Fig. 12. Details of line buffer dataflow for pixel reuse in depth-wise convolution.

resource requirement of the compact model (e.g., ~1/7 of SSD300HW). Figure 12 illustrates how the data pipeline is configured for reusing pixels. Without pixel reuse, $Tox \times Toy$-sized tiles need to be read multiple times even for overlapping pixels. PEs can only calculate $Pox \times Poy$ amount of output pixels at a time in this case. This takes 12 cycles in our design, and multiplying the number of direct reads from the buffer adds an extra 15.73 ms of delay for processing the "conv10 dw/s2" layer in MobileNet-V1 for the COCO dataset. Compared to the layer delay of 5.45 ms in our accelerator design with pixel reuse, pixel reuse in the DWConv layer will reduce this latency from 21.18 to 5.45 ms (3.89× reduction).

The second difference is in the point-wise convolution layer. This can be deemed as a plain convolution layer with a 1×1 kernel, but it needs to support convolution with zero padding. Our hardware and compiler implementation was designed accordingly to include these requirements. We added hardware to our input pixel pipeline to select whether we will load zero padding or not to PE, as implemented by the multiplexer shown in Figure 12.

3.5 DMA Descriptor Buffer Design

In prior works with larger FPGAs [15], DMA descriptors containing DDR addresses are precalculated and the entire set of DMA descriptors required to complete one inference was stored in the DMA descriptor buffers (Figure 13(a)). This improves the FPGA performance since the delay for calculating DDR addresses is eliminated. However, when the tile size is very small due to the capacity limit of on-chip BRAM, DMA needs to move small tiles more frequently and thus the number of DMA descriptors for copying those tiles will increase. The DMA buffer size requirement can be calculated from the number of read/write DMA descriptors multiplied by each DMA descriptor size, 32 bits. Our 1.0× model required 15.36 Mb of read DMA buffer with conventional DMA architecture. Using a similar analysis, 1.52 Mb of write DMA buffer was required to hold the entire write DMA descriptor. Since ZU3EG only has 7.6 Mb of BRAM, evidently the conventional DMA scheme cannot be used for our target FPGA.

Figure 13(b) shows our proposed DMA system design. In this system, the entire read/write DMA descriptor will be stored in DDR memory. To reduce the size of DMA buffers, we designed small DMA descriptor buffers that can refill DMA descriptors from DDR memory. The size of the read/write DMA descriptor buffer was decided by the maximum input/output channel size that exists in our SSD model, where one DMA descriptor corresponds to one tile in one feature map. To accumulate Nif pixels from the L_{th} layer in the MAC array without stalling and to prepare Nif pixels from the $(L+1)_{th}$ layer for the ensuing computation, we need $2 \times Nif$ DMA descriptors in the buffer to prevent stalling convolution computations in consecutive layers.

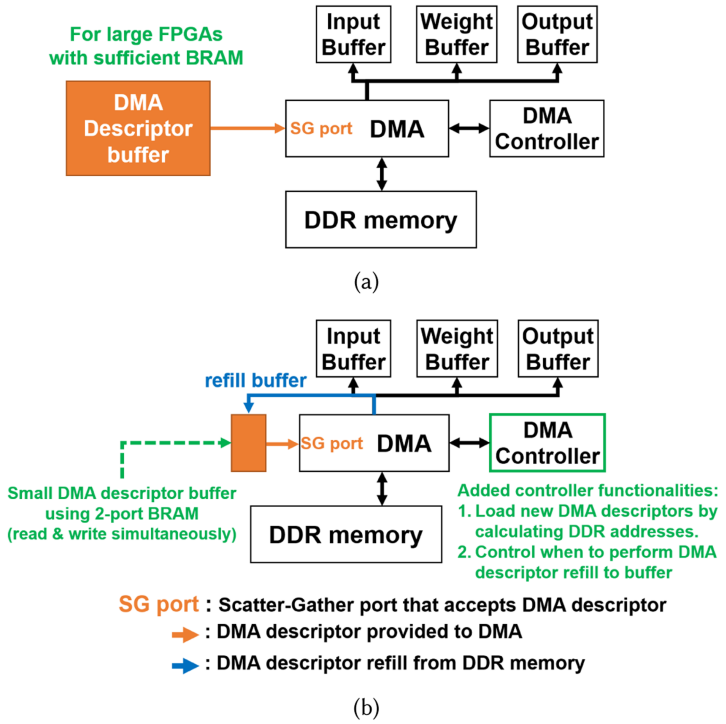


Fig. 13. (a) Conventional DMA descriptor buffer design for large FPGAs with sufficient BRAM. (b) Proposed DMA design with small descriptor buffer for small FPGAs with limited BRAM.

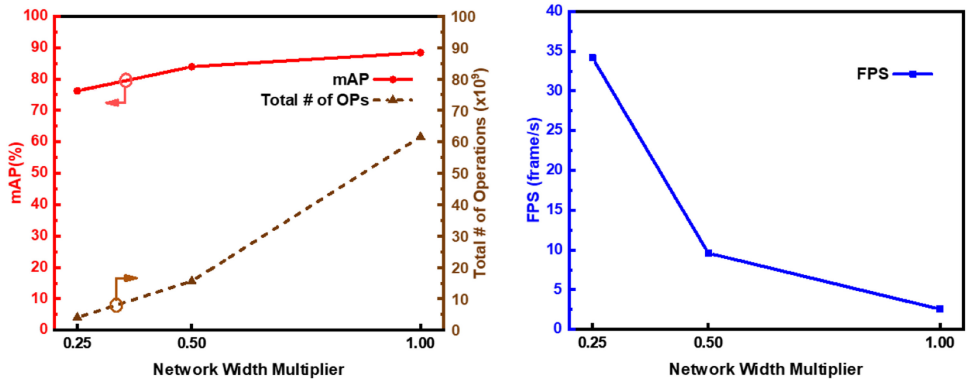
In our 1.0× SSD model, the largest input channel sizes of two adjacent convolution layers are 1,024 and 1,024. Thus, we need up to 2,048 DMA descriptor buffers ready in the DMA buffer. This corresponds to 32-bit × 2,048 = 64 Kb of capacity in the buffer. The same logic applies to the write DMA descriptor buffer. With our DMA descriptor design, the descriptor buffer size is only 64 Kb for each DMA module for read and write. By using the proposed DMA scheme with a small DMA descriptor buffer, the capacity requirement of the read DMA descriptor buffer is reduced by 240× (15.36 Mb/64 Kb), and that of the write DMA descriptor buffer is reduced by 23.7× (1.52 Mb/64 Kb).

4 EXPERIMENT RESULTS

4.1 Software Experiments

We investigated the VGG16-SSD model with different width multipliers (1.0×, 0.5×, 0.25×). For MobileNet-V1 SSD, the 1× full-width model was implemented. We evaluated the model performance on our custom multi-drone dataset [17], the Pascal VOC dataset, and the Microsoft COCO dataset. For the multi-drone detection task, we first load the pre-trained full-precision VGG-16 backbone CNN model trained by the ImageNet dataset and then fine-tune the entire VGG-based SSD model with the synthetic multi-drone images [17] while applying the 8-bit UniPOT quantization. After that, the low-precision fine-tuning process will be continuously performed on the real images. We heuristically select 0.125 and 16 as the quantization boundary for weight and activation, respectively.

We also fine-tuned the SSD model on the Pascal VOC dataset with the 8-bit backbone VGG CNN to make it comparable with prior works. Table 9 shows that the 1.0× SSD model with 8-bit precision



(a) (b)

Fig. 14. Hardware inference results from different SSD network width multiplier settings for (a) mAP and the total number of operations (b) FPS.

exhibits no mAP degradation compared to the full-precision baseline mAP of 77.2% [13]. For the MobileNet-V1 SSD model, we retrained the baseline model from Caffe model-zoo with the COCO dataset. The baseline model is the reproduction of [6], and we used the same 8-bit quantization for this model.

4.2 Hardware Experiments

Our system is implemented and tested by using the OVC3 system. The FPGA used in OVC3 is Xilinx Zynq ZU3EG. Our implementation was done using HDL with SystemVerilog. Our design tool used is Vivado 2019.2 for hardware bitstream generation, and Vitis 2019.2 for Linux boot image creation. Then, the Linux image is loaded to an ARM processor, which is equipped on a ZU3EG board by default.

The SSD and backbone CNNs with 8-bit precision using the UniPOT quantization scheme are implemented on the FPGA device, while the **non-maximum suppression (NMS)** and post-processing modules are performed by the CPU in OVC3. Our accelerator runs at 200 MHz and DSPs run at 400 MHz of frequency, aided by our dual-data-rate DSP design.

The reason is that the maximum frequency of Xilinx's DSP in our device variant, ZU3EG, is 400 MHz. We can run the DSP only either at the maximum (double) 400 MHz frequency or at the baseline frequency of 200 MHz. Due to this constraint, we run our non-DSP parts of our FPGA design at 200 MHz to simplify the clock domains of the overall accelerator and eliminate the use of additional FIFOs for clock-domain crossing.

Figures 14(a) and 14(b) show the performance and mAP values of our FPGA designs across different VGG16-SSD width models. With algorithm-hardware co-design and optimizations, we could achieve up to 34.2 FPS for the 0.25× VGG16-SSD model. The implementation of the 0.5× model achieves 9.60 FPS, with only 2% mAP degradation compared to that of the 1.0× VGG16-SSD model.

For the MobileNet-V1 backbone network, we achieve up to 12.67 FPS. For the COCO dataset, our implementation exhibits 1.9% mAP degradation compared to the FP32 model [19]. A prior work [11] shows 0.2% higher mAP, but the model size of [11] (8.9 MB) was 2.8× larger than our quantized network model (3.15 MB). The MobileNet-V1 with Pascal VOC dataset achieved mAP of 65.89%. While this is 9.21% lower than the state-of-the-art work in [19], note that FP32 precision was used in [19] with larger input image size, which will introduce more computational complexity

Table 7. FPGA Resource Utilization for Implementations of Different VGG16-SSD Width Models

Resource	Utilization			Available in FPGA
SSD Model Width	1.0×	0.5×	0.25×	
LUT	64,128	64,613	64,544	70,560
LUTRAM	14,032	14,155	14,153	20,880
FF	74,759	75,482	75,468	141,120
BRAM	184.5	119	102	216
DSP	263	263	263	360

Table 8. FPGA Resource Utilization for Implementations of the MobileNet-V1 Model

Resource	Utilization	Available in FPGA
LUT	53,533	70,560
LUTRAM	2,065	28,800
FF	77,058	141,120
BRAM	205.50	216
DSP	264	360

to the entire network. On the other hand, in comparison to compressed network models, our work achieved better energy efficiency in terms of FPS/W than [11] (2.6 FPS/W vs. 2.87 FPS/W) with a small loss of 0.51% mAP.

The resource utilization for FPGA implementations of different VGG16-SSD width models is reported in Table 7. Table 8 shows the resource utilization of FPGA implementation for MobileNet-V1. In our FPGA design, we can adjust the ratio of distributed RAM and BRAM utilization for a certain buffer (input/weight/output), which especially aided the 1.0× implementation to achieve higher resource utilization and better performance with a limited amount of BRAM.

Furthermore, to fully utilize the buffer capacity in our accelerator, we fine-tuned and adjusted Toy and Tof values for each convolution layer. This will enlarge the size of tiles when necessary, ensuring that the buffers will be filled with data at all times and that the number DDR memory accesses is minimized. Figure 15(b) shows the input buffer write latency reduction achieved by per-layer Toy/Tof adjustment, where 2 to 4× latency reduction is shown for bottleneck layers such as conv3_2 and conv4_2. Only the convolution layers for backbone CNN are shown in Figure 15(a), since the SSD layers only consume <8% of the overall latency. Figure 16 shows the latency per layer for our MobileNet-V1 implementation for both Pascal VOC and COCO datasets. Due to the kernel size difference (3×3 vs. 1×1), the DWConv layer takes more time than the PWConv layer due to intra-channel accumulation and multiplication operations in convolution. Figure 17 shows several examples of drone detection results by our FPGA.

In Table 9, we compare our proposed hardware implementation with prior works that target the same or similar UAV applications. Unfortunately, other works [22, 24, 25] only reported IoU values, but not the corresponding mAP values for drone-specific datsets. While Skynet [25] achieved 3.45 FPS/W for object detection, our 0.25× SSD achieved 14.24 FPS/W (4.1× higher) and 0.5× SSD achieved 3.69 FPS/W (1.1× higher). Compared to [24], we achieve 2.2 to 3.1× better energy

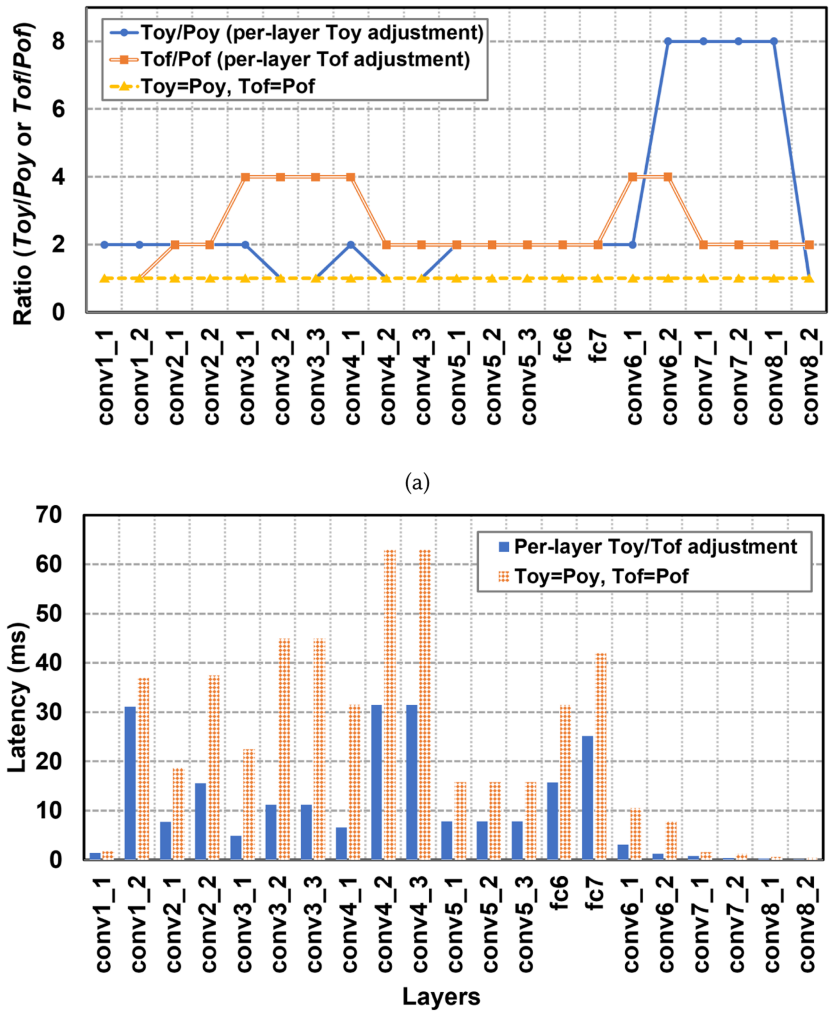


Fig. 15. VGG16-SSD optimization results. (a) Per-layer optimization of Tof/Pof and Toy/Poy. (b) Input buffer write latency with/without Toy and Tof optimization.

efficiency (GOPS/W) for all three widths of SSD (1.0×, 0.5×, and 0.25×). For our 0.25× SSD model implementation that achieves similar mAP as [24], our FPS/W is 4.88× higher than that of [24].

Table 10 shows the performance, energy efficiency, and mAP results for general object detection from other mobile, edge, or FPGA devices. Among the prior works in Table 10, [10] reported the most comprehensive results including throughput (GOPS), FPS, power consumption, and mAP for the Pascal VOC dataset while using the same FPGA ZU3EG as our work.

While both [10] and our work used the same FPGA device and both used the MobileNet-V1 model, the input image size is higher for the MobileNet model in [10], which means that there are more operations. On the other hand, [10] used lower 4/3-bit precision, while we employed 8-bit precision with the UniPOT algorithm. In addition, [10] used ARM cores besides the FPGA device for certain computing/control operations. This heterogeneous architecture of CPU and FPGA in [10]

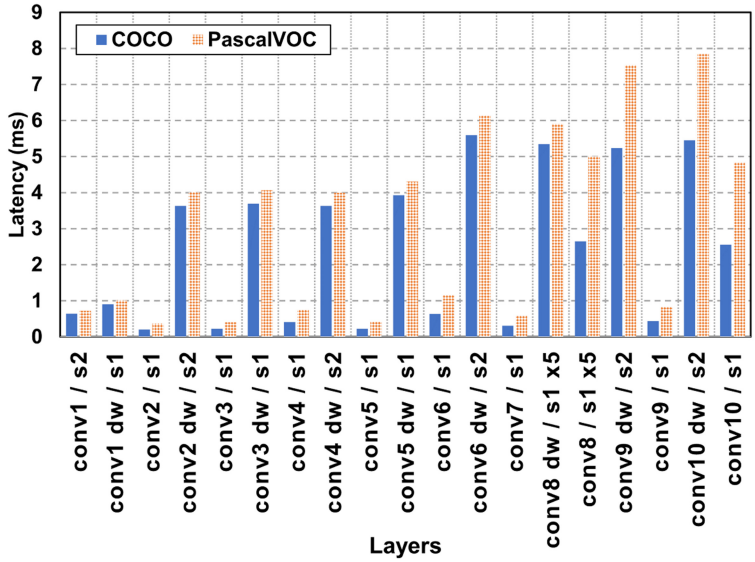


Fig. 16. Layer-wise latency results of our MobileNet-V1 FPGA implementation for COCO and Pascal VOC.

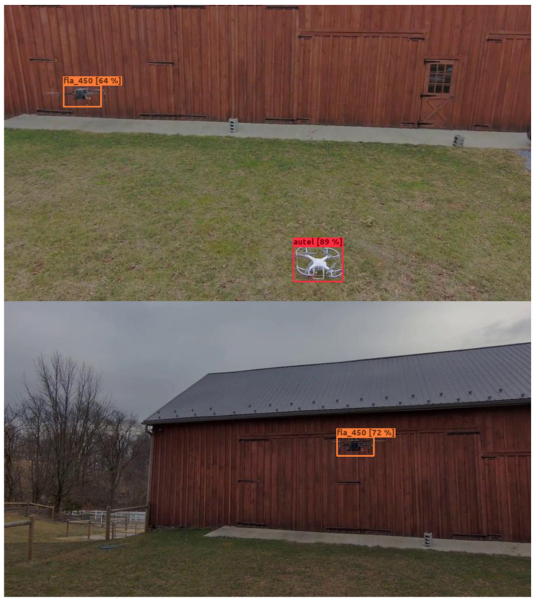


Fig. 17. Drone detection results by FPGA with bounding boxes on images from the multi-drone dataset.

likely enhanced the throughput (GOPS) with further pipelining and helped achieve high GOPS/W, whereas our work only used FPGA for the overall computation.

While achieving similar mAP for the Pascal VOC dataset, for other metrics of energy efficiency, our work achieved 8.6% lower energy/image (J/image) and 10% higher FPS/W than those of [10]. In other words, when we compare the energy on a frame basis, our work achieves a bit higher energy efficiency (J/image and FPS/W) than [10]. The much higher GOPS/W value in [10] could have

Table 9. Comparison with Prior Object Detection Works for UAV Applications

	[25]	[24]	Ours (0.25×)	Ours (0.5×)	Ours (1.0×)
FPGA Platform	Zynq ZU3EG	PYNQ-Z1	Zynq ZU3EG	Zynq ZU3EG	Zynq ZU3EG
Frequency (MHz)	214	143	200	200	200
Backbone CNN	Skynet	VGG-16	VGG-16	VGG-16	VGG-16
Input image size	360×160	448×252	300×300	300×300	300×300
Precision (Activation/Weight)	9/11 bits	8/8 bits	8/8 bits	8/8 bits	8/8 bits
Total # of OPs in CNN	-	8.73	4.04	15.65	61.60
Avg. Performance (GOPS)	-	104.42	137.97	150.24	157.83
Power (W)	7.26	4.1	2.4	2.6	2.0
Energy Efficiency (GOPS/W)	-	25.5	57.5	57.8	78.9
Energy Efficiency (FPS/W)	3.45	2.92	14.24	3.69	1.28
FPS	25.05	11.96	34.18	9.60	2.56
mAP (multi-drone dataset)	-	-	76.2	83.9	88.4

been achieved due to the execution of a larger model that led to higher resource FPGA utilization. However, at the end, frame-based energy or how much total energy it consumes to finish a given task is more important, rather than the mere value of GOPS/W, as pointed out in [21].

In comparison with ARM Cortex-A53 processor-based implementation [1], our work achieves 8.7× higher FPS/W energy efficiency than [1] for the Pascal VOC dataset. Note that [1] used a very small CNN model, which only used four convolution layers and three fully connected layers, and also their input image size is very small (32 × 32). As a result, their custom CNN model only has 61.7M operations and thus exhibits much lower GOPS and GOPS/W.

To fully benefit from our proposed implementation, we can trade off performance and accuracy to make the hardware best fit the application's needs. Where the real-time detection of objects is more important, we can deploy our 0.25× model algorithm to hardware to run it over 30 FPS, which is enough to infer image frames from a video in real-time. On the other hand, if accuracy is more important, we can use our 0.5× or 1.0× model to get the best accuracy within small edge FPGA devices such as ZU3EG.

Table 10 shows the results of MobileNet-V1 with our proposed hardware design and comparison to other FPGA works as well as commercial embedded/mobile hardware. Snapdragon and Nvidia's Jetson series use much higher operating frequency with higher power consumption. Also, higher floating-point precision was used in these works, which cannot efficiently fit onto the small devices like ZU3EG we used. Our MobileNet-V1 FPGA implementation for the COCO dataset achieves 4.9 FPS/W, which is ∼1.9× higher than the prior FPGA works and commercial embedded/mobile hardware.

5 CONCLUSION

In this work, we co-optimized algorithm and hardware for high-throughput and low-power drone detection on resource-constrained FPGA devices. In mobile/edge devices, power consumption is very limited due to the capacity of the battery and thus having a high energy efficiency with marginal/affordable mAP degradation is important. In this project, we propose the optimal mAP and FPS tradeoff for less than 5W of energy consumption and compare our FPGA results with those of the state-of-the-art works that deployed to the edge/mobile devices. Our proposed design satisfies both the on-chip and DSP resource limit and power constraint, without losing too much mAP (0.51% for PascalVOC and 0.2% for COCO dataset compared to the state-of-the-art compressed network model) for UAV-based multi-box object detection applications.

Table 10. Comparison with Prior General Object Detection Works Using Mobile/Edge Platforms

	[19]	[2]	[5]	[10]	[1]	Ours (VOC)	Ours (COCO)
Platform	Snapdragon 845	Nvidia Jetson AGX XAVIER	Nvidia Jetson TX1	Zynq ZU3EG	Cortex-A53 (Armv8-A)	Zynq ZU3EG	
Frequency (Hz)	1.7/2.8G (Boost)	1.2/2.265G (Boost)	~1.91G	215M	1.2G	200M	
Backbone CNN	ThunderNet	MobileNet-V2	Tinier-YOLO	MobileNet-V1	Custom CNN	MobileNet-V1	
Input image size (VOC/COCO)	320/224	512	416	512	32	300	224
Precision (Activation/Weight)	FP32	FP32	-	4/3 bits	FP32	8 bits	
Total # of OPs in CNN	-	-	-	5.50G	61.7M	2.40G	1.21G
Avg. Performance (OPS)	-	-	2.56B	202.8G	75.3M	17.44G	15.27G
Power (W)	-	10~30	~10	6.9	3.7	2.54	2.57
Energy Efficiency (J/image)	-	-	-	0.383	3.03	0.35	0.20
Energy Efficiency (GOPS/W)	-	-	-	29.4	0.02	6.87	5.94
Energy Efficiency (FPS/W)	1.53	0.77~2.3	~2.51	2.6	0.33	2.87	4.93
FPS	13.8	23	25.1	18	1.22	7.28	12.67
mAP (VOC)	75.1	73.0	65.7	66.4	72	65.89	-
mAP (COCO)	18.7	-	17.0	-	-	-	16.8

On the algorithm side, we employed a uniform and unified low-precision quantization scheme termed UniPOT to achieve high mAP and simple hardware mapping. On the hardware side, we performed comprehensive design space exploration to fully utilize the limited FPGA resources and optimize throughput. We also employed dual-data-rate operations for DSPs to double the throughput. Across three different widths of SSD models, we optimized and analyzed mAP, FPGA hardware utilization, throughput, and energy efficiency. Our FPGA design achieves a high mAP of 88.42 on the drone dataset, together with a high energy efficiency of 79 GOPS/W and throughput of 158 GOPS using the Xilinx Zynq ZU3EG FPGA device on the Open Vision Computer version 3 (OVC3) platform.

REFERENCES

[1] Mohanad Abd Shehab, Ammar Al-Gizi, and Salah M. Swadi. 2021. Efficient real-time object detection based on convolutional neural network. In *2021 International Conference on Applied and Theoretical Electricity (ICATE'21)*. IEEE, 1–5.

[2] Yu-Chen Chiu, Chi-Yi Tsai, Mind-Da Ruan, Guan-Yu Shen, and Tsu-Tian Lee. 2020. Mobilenet-SSDv2: An improved object detection model for embedded systems. In *2020 International Conference on System Science and Engineering (ICSSE'20)*. IEEE, 1–5.

[3] Jungwook Choi, Swagath Venkataramani, Vijayalakshmi Srinivasan, Kailash Gopalakrishnan, Zhuo Wang, and Pierce Chuang. 2019. Accurate and efficient 2-bit quantized neural networks. In *Conference on Machine Learning and Systems (MLSys'19)*.

[4] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. 2021. RepVGG: Making VGG-Style ConvNets great again. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR'21)*. 13733–13742.

[5] Wei Fang, Lin Wang, and Peiming Ren. 2019. Tinier-YOLO: A real-time object detection method for constrained environments. *IEEE Access* 8 (2019), 1935–1944.

[6] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[7] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR'18)*. 2704–2713.

[8] Ercan Kalali and Rene Van Leuken. 2021. Near-precise parameter approximation for multiple multiplications on a single DSP block. *IEEE Trans. Comput.* 71, 9 (2021), 2036–2047.

[9] Raghuraman Krishnamoorthi. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342* (2018).

[10] Fanrong Li, Zitao Mo, Peisong Wang, Zejian Liu, Jiayun Zhang, Gang Li, Qinghao Hu, Xiangyu He, Cong Leng, Yang Zhang, and Jian Cheng. 2019. A system-level solution for low-power object detection. In *IEEE/CVF International Conference on Computer Vision Workshops*.

[11] Yuhang Li, Xin Dong, and Wei Wang. 2019. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. In *International Conference on Learning Representations (ICLR'19)*.

[12] Mingbao Lin, Rongrong Ji, Zihan Xu, Baochang Zhang, Yan Wang, Yongjian Wu, Feiyue Huang, and Chia-Wen Lin. 2020. Rotated binary neural network. *Advances in Neural Information Processing Systems* 33 (2020), 7474–7485.

[13] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. 2016. SSD: Single shot multibox detector. In *European Conference on Computer Vision (ECCV'16)*. 21–37.

[14] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jaesun Seo. 2018. Optimizing the convolution operation to accelerate deep neural networks on FPGA. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26, 7 (2018), 1354–1367.

[15] Yufei Ma, Tu Zheng, Yu Cao, Sarma Vrudhula, and Jaesun Seo. 2018. Algorithm-hardware co-design of single shot detector for fast object detection on FPGAs. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'18)*. 1–8.

[16] Dong Nguyen, Daewoo Kim, and Jongeun Lee. 2017. Double MAC: Doubling the performance of convolutional neural networks on modern FPGAs. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'17)*. IEEE, 890–893.

[17] Ty Nguyen, Ian D. Miller, Avi Cohen, Dinesh Thakur, Arjun Guru, Shashank Prasad, Camillo J. Taylor, Pratik Chaudhari, and Vijay Kumar. 2021. PennSyn2Real: Training object recognition models without human labeling. *IEEE Robotics and Automation Letters* 6, 3 (2021), 5032–5039.

[18] Eunhyeok Park and Sungjoo Yoo. 2020. PROFIT: A novel training method for sub-4-bit MobileNet models. In *European Conference on Computer Vision (ECCV'20)*. 430–446.

[19] Zheng Qin, Zeming Li, Zhaoning Zhang, Yiping Bao, Gang Yu, Yuxing Peng, and Jian Sun. 2019. ThunderNet: Towards real-time generic object detection on mobile devices. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 6718–6727.

[20] Morgan Quigley, Kartik Mohta, Shreyas S. Shivakumar, Michael Watterson, Yash Mulgaonkar, Mikael Arguedas, Ke Sun, Sikang Liu, Bernd Pfrommer, Vijay Kumar, and Camillo J. Taylor. 2018. The open vision computer: An integrated sensing and compute system for mobile robots. *CoRR* abs/1809.07674 (2018). arXiv:1809.07674 http://arxiv.org/abs/1809.07674.

[21] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. 2020. How to evaluate deep neural network processors: TOPS/W (alone) considered harmful. *IEEE Solid-State Circuits Magazine* 12, 3 (2020), 28–41. https://doi.org/10.1109/MSSC.2020.3002140

[22] Di Wu, Yu Zhang, Xijie Jia, Lu Tian, Tianping Li, Lingzhi Sui, Dongliang Xie, and Yi Shan. 2019. A high-performance CNN processor based on FPGA for MobileNets. In *IEEE International Conference on Field Programmable Logic and Applications (FPL'19)*. 136–143.

[23] Xilinx. 2020. Product Guide of DPU (Deep Learning Processing Unit) IP. https://www.xilinx.com/support/documentation/ip_documentation/dpu/v3_2/pg338-dpu.pdf.

[24] Xiaowei Xu, Xinyi Zhang, Bei Yu, X. Sharon Hu, Christopher Rowen, Jingtong Hu, and Yiyu Shi. 2021. DAC-SDC low power object detection challenge for UAV applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43, 2 (2021), 392–403.

[25] Xiaofan Zhang, Cong Hao, Haoming Lu, Jiachen Li, Yuhong Li, Yuchen Fan, Kyle Rupnow, Jinjun Xiong, Thomas Huang, Honghui Shi, et al. 2019. Skynet: A champion model for DAC-SDC on low power object detection. *arXiv preprint arXiv:1906.10327* (2019).

[26] Yiren Zhao, Xitong Gao, Xuan Guo, Junyi Liu, Erwei Wang, Robert Mullins, Peter Y. K. Cheung, George Constantinides, and Cheng-Zhong Xu. 2019. Automatic generation of multi-precision multi-arithmetic CNN accelerators for FPGAs. In *2019 International Conference on Field-Programmable Technology (ICFPT'19)*. IEEE, 45–53.

Received 29 May 2022; revised 3 December 2022; accepted 27 January 2023