Algorithm-Hardware Co-Optimization for Energy-Efficient Drone Detection on Resource-Constrained FPGA

Han-sok Suh*, Jian Meng*, Ty Nguyen†, Shreyas K. Venkataramanaiah*, Vijay Kumar†, Yu Cao*, Jae-sun Seo*
*Arizona State University, Tempe, AZ, USA
†University of Pennsylvania, Philadelphia, PA, USA

2021 International Conference on Field-Programmable Technology (ICFPT) | 978-1-6654-2010-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICFPT52863.2021.9609840

***Abstract*—Convolutional neural network (CNN) based object detection has achieved very high accuracy, e.g. single-shot multi-box detectors (SSD) 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 paper, we designed and co-optimized algorithm and hardware for energy-efficient drone detection on resource-constrained FPGA devices. We trained 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, throughput optimization. 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 Xilinx Zynq ZU3EG FPGA device on the Open Vision Computer version 3 (OVC3) platform. Our design achieves 2.7X higher energy efficiency than prior works using the same FPGA device, at a low-power consumption of 1.98 W.**

***Index Terms*—FPGA accelerator, convolutional neural network, object detection, algorithm-hardware co-design.**

I. 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), etc. 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 exhibits stringent power/energy requirements, which prohibits a high degree of parallelism or a massive amount of storage for the compute hardware on UAVs. Nevertheless, 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 [1]. 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 [2]. 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) [3], which uses VGG-16 as the backbone CNN. While recently there have been variants of the original SSD that uses compact CNN models such as MobileNet for the backbone CNN, such compact models inevitably sacrifice mAP. In recent algorithm works [4], VGG-style CNNs have been revived to show favorable accuracy-speed trade-off compared to state-of-the-art CNNs, where the regularity of using only 3×3 convolution kernels aids faster hardware speed. On the other hand, recent works on adversarial bit flip attacks [5] have shown that, large models such as VGG family show higher resilience to such attacks, compared to compact models such as MobileNet family. For example, [5] reported that, by flipping just two (tar-

This work is partially supported by NSF grant 1652866, and C-BRIC, one of six centers in JUMP, a SRC program sponsored by DARPA.

geted) bits in the MobileNet-V2 model, the baseline accuracy of 72.01% severely degraded to 0.19% for ImageNet. While VGG models have a relatively large number of parameters, the resilience to adversarial attacks remains an advantage, compared to more compact models.

To that end, we use the SSD model with VGG-16 as the backbone CNN throughout this paper, while we explore different widths of VGG-16 CNN to show the trade-off of model size, mAP, throughput, and energy-efficiency. We trained a hardware-friendly variant of the original SSD model using the drone dataset presented in [6]. 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 UniPOT scheme results in better trade-off in hardware utilization and mAP, compared to 4-bit or lower quantization with more complex quantization schemes such as additive-power-of-two (APOT) [7] that 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 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.
- 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 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 trade-off compared to prior works.

II. Multi-Drone Dataset

In this study, we evaluate the performance of our FPGA-based object detection algorithm on the outdoor, multi-drone dataset introduced in [6]. This dataset features an *Autel X-Star* [8] and an *FLA-450* flying in a cluttered, outdoor environment. The top row of Fig. 1 shows examples of this dataset. We follow their experiment settings and also split the data into three sets: a training set consisting of $1,000$ frames, and a test set with 800 frames, and a validation set with 200 frames.

We also make use of the synthetic dataset introduced in [6] to pretrain the model before fine-tuning on the real multi-drone dataset as it has been shown to be greatly effective [6]. The bottom row of Fig. 1 provides some examples of this synthetic dataset in comparison with real images (top row). We use $31,000$ synthetic images for training and $14,000$ synthetic images for evaluation.

Fig. 1: Example of real images (top row) and synthetic images (bottom row) under the same surrounding environment [6].

III. Algorithm Optimization

A. Custom SSD Model Adaptation

In this work, we adopt SSD300-HW [9], a variant of SSD300x300 [3] 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 [6]. At the end of 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, 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 model by adopting the width multiplier [10] to the VGG CNN, towards achieving better higher throughput. In particular, we first trained the narrower VGG-16 model (e.g, 0.5×, 0.25×) with ImageNet dataset, and subsequently cascaded the pre-trained CNNs to the full size (1.0×) SSD model.

To obtain optimal prior boxes for the SSD model, we also use k-nearest neighbor (KNN) clustering as suggested in [3]. 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 saturated. Based 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 [3], we do not flip these prior boxes because it is unnecessary.

B. 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 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 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 requires high precision scaling [11]–[13] or extra pre-processing steps [14], which causes hardware overhead when such algorithms are fully implemented onto hardware devices.

In comparison to uniform quantization, the power-of-two (POT) quantization converts the multiplication into shifting operation and substantially simplifies the hardware. However, the lopsided resolution of the POT quantization degrades the accuracy. To address the accuracy degradation, additive-power-of-two (APOT) [7] 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 look-up tables (LUTs). However, compared to uniformly quantized weights, representing each summed APOT level requires higher data precision than n-bit (e.g., 2^4=16 APOT levels requires 5-7 bits to store each level in hardware), and this aggravates the hardware implementation overhead. Furthermore, layer-wise learnable scaling factors that require high precision 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 broadcasted to the entire DNN model (Fig. 2). Constraining the distribution of weights and activations by the unified POT value can avoid the high-precision scaling in (2) and subsequently simplifies the hardware implementation.

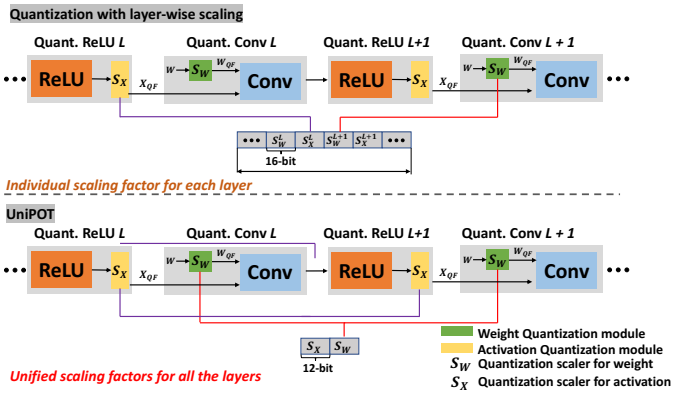

Fig. 2: Comparison of other low-precision quantization and the proposed UniPOT quantization schemes.

TABLE I: 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 [12]	4/4	4 bit	82.37	Layer-wise
APOT [7]	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**

Table I summarizes the software performance of VGG-based SSD on the multi-drone dataset [6] by applying different quantization strategies. Uniformly quantizing the model down to 4-bit [12] leads to significant accuracy degradation. Deploying the APOT-quantized model requires layer-wise scaling and high data precision with noticeable accuracy degradation. 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 Fig. 2, each quantized-ReLU layer (QReLU) layer including a regular ReLU function and an activation quantization module ((2)) to (4)) generates the digitized convolution output [15]. Similar to the offline integer transformation in the prior works [13], [16], the de-quantization scaling factors for activations/weigths in (4) can be extracted and folded into the scaler that belongs to the next layer. This means that instead of computing (5), we compute (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.

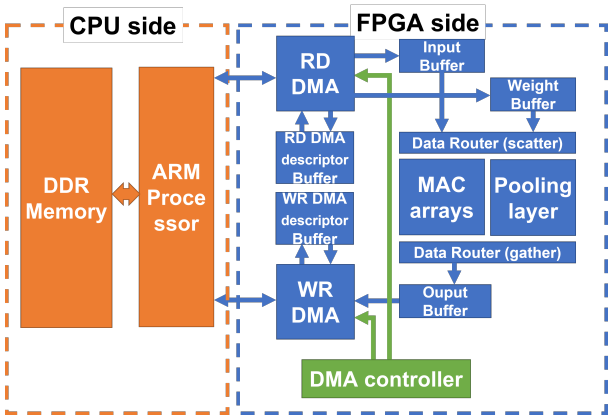

Fig. 3: Overall architecture of FPGA accelerator.

IV. 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.

A. Overall Hardware Architecture

Fig. 3 shows the overall hardware block diagram and data flow. 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 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 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.

We adapted the loop unrolling and loop tiling strategy introduced in [17]. Table II describes the terminologies for CNN algorithm and FPGA design parameters used in this work.

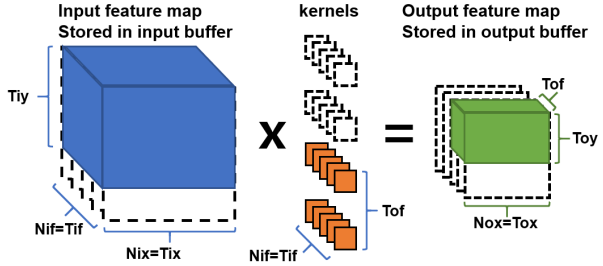

Fig. 4: Loop tiling strategy of our proposed architecture. Adapted from [17].

Adjusting 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 Tiy dimension only, as shown in Fig. 4. Once MAC operation is done, output feature map will be stored in output buffer with Tof and Toy tiling strategy (Fig. 4). 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 next batch of input feature maps to continue the accumulation. Thus, one iteration of 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 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 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.

B. Design Space Exploration

In this sub-section, we explain how we optimized our baseline architecture design to reduce the size of design while getting the best performance out of it. The entire design process we used in our work is given in the Fig. 5. The steps we used in our design space exploration is given below:

- Step 1: Increase parallelism by increasing number of MAC units in the design. This will achieve maximum DSP counts, which equals to $Pox \times Poy \times Pof$ from our design parameters.

TABLE II: 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

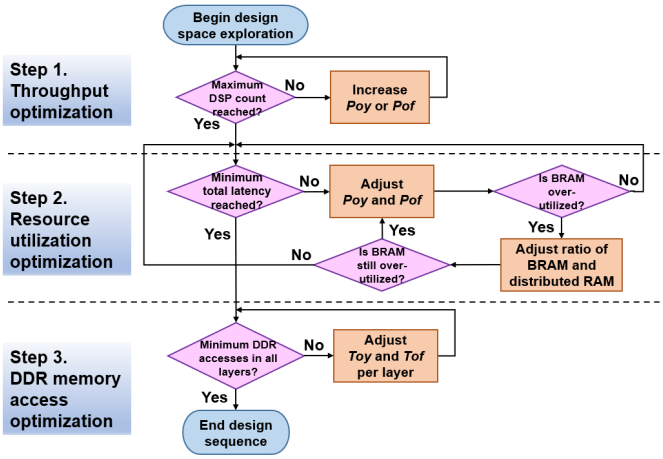

Fig. 5: Flow chart of our proposed design space exploration.

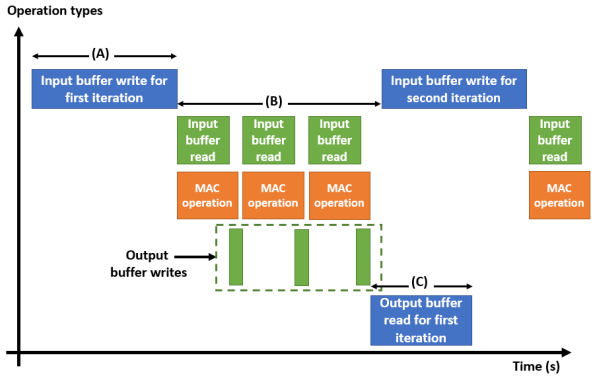

Fig. 6: Timing diagram of MAC array operation including DRAM communication.

- Step 2: With parallelism fixed, test different (Pox, Poy, Pof) combinations to find best performance in terms of total latency. During this phase, we tried to distribute available FPGA resources evenly throughout our design.
- Step 3: Adjust the loop tiling size of Toy and Tof to reduce the number of DDR memory accesses.

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 DSP slices in the FPGA. In our target FPGA, 360 DSP resources are available. Since the largest power of 2 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.

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. 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. In our architecture, we have 4 banks of input buffer which each of them stores Pox amount of pixels in a row. This means we need $4 \times Pox \times$ 8-bit (activation bit-width) of bandwidth, where our DMA bandwidth is 256-bit. Therefore, the optimal choice of Pox is 8 in our design. To decide how many parallelism will be in Poy and Pof, we tried to assign more parallelism to a design parameter which comes with smaller buffer size to reduce BRAM resource utilization in FPGA.

In Fig. 4, 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 needs to be minimally stored in each buffer. On the other hand, Nif is not tiled and thus input buffer is relatively larger than the other two buffers. Now, if we increase Toy, this will increase Tiy and 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.

During the design search, if any combination of Pox, Poy, and Pof led to over-utilization of BRAM, we attempted to replace a part of buffer design with distributed RAM, which employs LUT plus LUTRAM resources available in the FPGA. Then the total latency of our FPGA design is estimated and the design search continues. In our design, the latency for one convolution layer is comprised 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 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 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 is optimally chosen as 8, 2, and 16, respectively.

3) Step 3: DDR memory access optimization: Fig. 6 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 bottleneck in the DDR memory performance. The latency of (A), (B), and (C) varies for different convolution layers. In the cases of 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 conv4_2 layer. During input buffer write, read DMA module was spending too much time waiting 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 Toy or Tof design parameter without increasing the size of input/output/weight buffers. For fine-

grain optimization, we fine-tuned Toy and Tof for each layer of the CNN.

C. 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 generates two outputs at the same time, as shown in Fig. 7a. The DSP slices in our FPGA design works at 400 MHz of frequency, twice of the operation frequency (200 MHz) of the main CNN accelerator. Although DDR DSP is standard in Xilinx's deep learning processing unit (DPU) IP [18], 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 (Fig. 7a). 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. The number of stages in pipeline registers are determined based on the information in the Xilinx technical manual. Clock crossing logics are inserted to the control signals, such as the accumulator reset signal and multiplexer select signals. Fig. 7b 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.

D. DMA Descriptor Buffer Design

In prior works with larger FPGAs [9], DMA descriptors containing DDR addresses are pre-calculated and the entire set of DMA descriptors required to complete one inference were stored in the DMA descriptor buffers (Fig. 8a). 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. DMA buffer size requirement can be calculated from the number of read/write DMA descriptors multiplied by each DMA descriptor size, 32-bit. 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.

Fig. 8b 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 read/write DMA descriptor buffer was decided by the maximum input/output channel size exists in our SSD model, where one DMA descriptor corresponds to the one tile in one feature map. To accumulate Nif pixels from L_{th} layer in the MAC array without stalling and to prepare Nif pixels from $(L+1)_{th}$ layer for the ensuing computation, we need $2 \times Nif$ DMA descriptors in the buffer to prevent stalling convolution computations in consecutive layers.

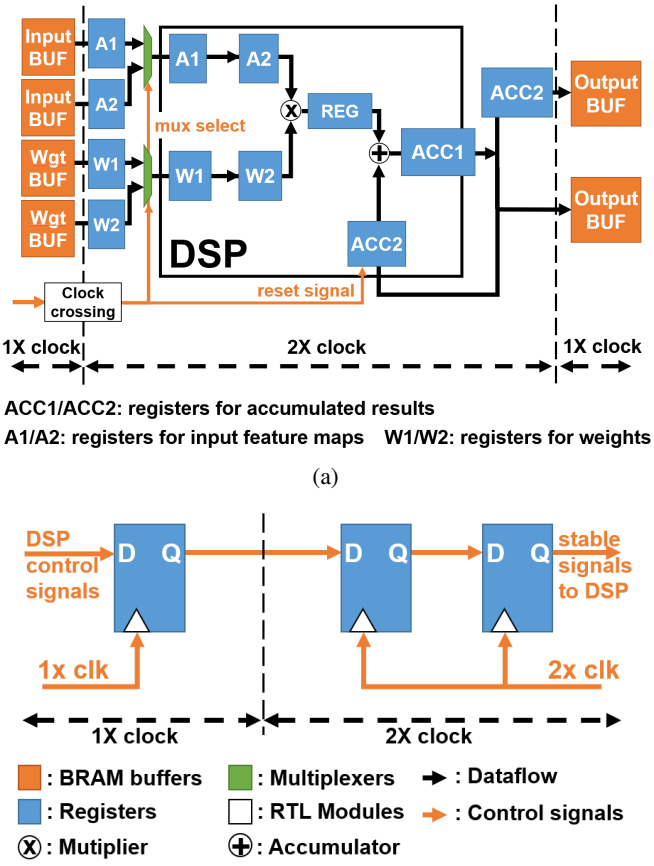

Fig. 7: (a) Dual data rate DSP implementation. (b) Clock crossing logic used in DDR DSP control signals.

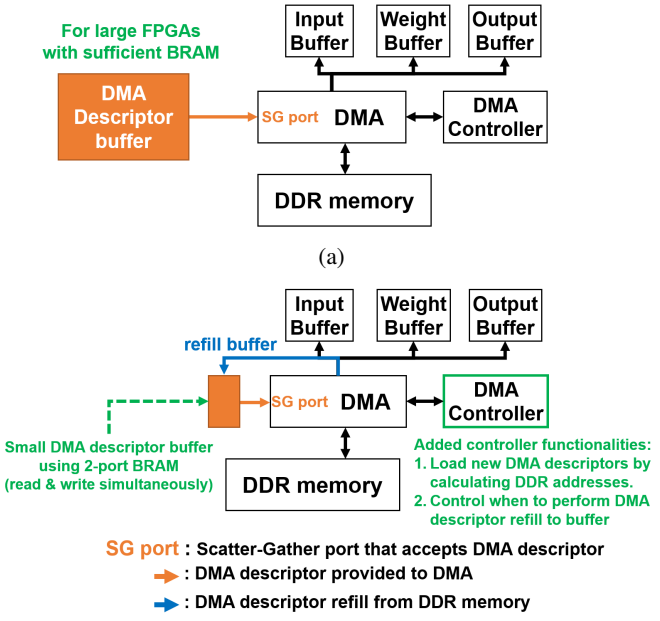

Fig. 8: (a) Conventional DMA descriptor buffer design for large FPGAs with sufficient BRAM. (b) Proposed DMA design with small descriptor buffer for small FPGAs with limited BRAM.

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 × 2048 = 64 Kb of capacity in the buffer. The same logic applies to the write DMA descriptor buffer. With our DMA descriptor design, descriptor buffer size is only 64 Kb for each DMA module for read and write. By using the proposed DMA scheme with small DMA descriptor buffer, the capacity requirement of read DMA descriptor buffer is reduced by 240× (15.36 Mb / 64 Kb), and that of write DMA descriptor buffer is reduced by 23.7× (1.52 Mb / 64 Kb).

V. Experiment Results

A. Software Experiments

We investigated the SSD model with different width multipliers (1.0×, 0.5×, 0.25×). We evaluated the model performance on both multi-drone dataset [6] and Pascal VOC dataset. For multi-drone detection task, we first load the pre-trained full-precision VGG-16 backbone CNN model trained by ImageNet dataset, and then fine-tune the entire VGG-based SSD model with the synthetic multi-drone images [6] 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 Pascal VOC dataset with the 8-bit backbone VGG CNN, to make it comparable with prior works. Table IV shows that the 1.0× SSD model with 8-bit precision exhibits no mAP degradation compared to the full-precision baseline mAP of 77.2% [3].

B. Hardware Experiments

Our system is implemented and tested by using the OVC3 system. The FPGA used in OVC3 is Xilinx Zynq ZU3EG. The SSD and backbone CNN with 8-bit precision using the UniPOT quantization scheme are implemented on the FPGA

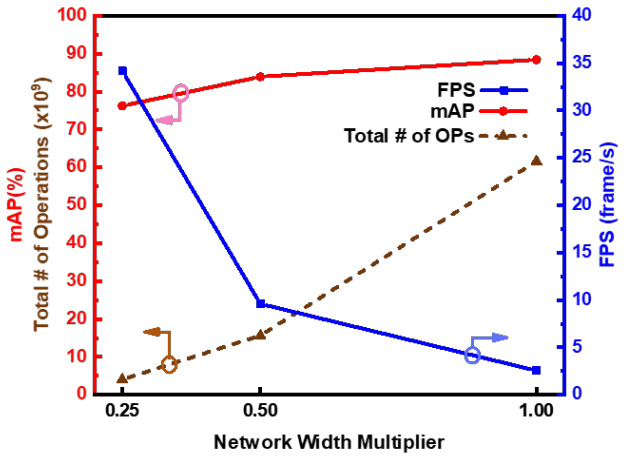

Fig. 9: Hardware inference results from different network width multiplier settings.

TABLE III: FPGA resource utilization for implementations of different 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

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.

Fig. 9 shows the performance and mAP values of our FPGA designs across different SSD width models. With algorithm-hardware co-design and optimizations, we could achieve up to 34.2 FPS for the 0.25× SSD model. The implementation of the 0.5× model achieves 9.60 FPS, with only 2% mAP degradation compared to that of 1.0× SSD model. The resource utilization for FPGA implementations of different SSD width models is reported in Table III. In our FPGA design, we can adjust the ratio of distributed RAM and BRAM utilization for

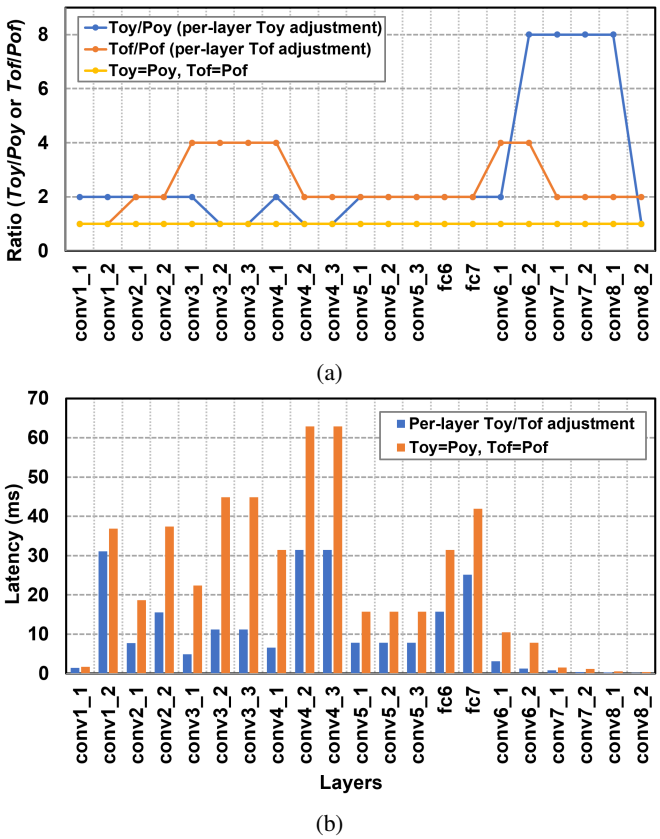

Fig. 10: (a) Per-layer optimization of Tof/Pof and Toy/Poy. (b) Input buffer write latency with/without Toy and Tof optimization.

TABLE IV: Comparison with prior object detection works using the same/similar FPGA platform.

	[19]	[20]	[21]	[22]	Ours (0.25×)	Ours (0.5×)	Ours (1.0×)
FPGA Platform	Zynq ZU3EG	Zynq ZU2EG	Zynq ZU3EG	PYNQ-Z1	Zynq ZU3EG	Zynq ZU3EG	Zynq ZU3EG
Frequency (MHz)	215	215	214	143	200	200	200
Backbone CNN	MobileNet	MobileNet	Skynet	VGG-16	VGG-16	VGG-16	VGG-16
Input image size	512×512	300×300	360×160	448×252	300×300	300×300	300×300
Precision (Activation/Weight)	4/3 bits	8/8 bits	9/11 bits	8/8 bits	8/8 bits	8/8 bits	8/8 bits
Total # of OPs in CNN	5.50	-	-	8.73	4.04	15.65	61.60
Avg. Performance (GOPS)	202.76	-	-	104.42	137.97	150.24	157.83
Power (W)	6.9	-	7.26	4.1	2.4	2.6	2.0
Energy-Efficiency (GOPS/W)	29.4	-	-	25.5	57.5	57.8	78.9
FPS	18	31.0	25.05	11.96	34.18	9.60	2.56
mAP (Pascal VOC dataset)	66.4	-	-	-	63.7	73.9	77.2
mAP (multi-drone dataset)	-	-	-	-	76.2	83.9	88.4

Fig. 11: Drone detection results by FPGA with bounding boxes on example images from the multi-drone dataset.

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 access is minimized. Fig. 10b shows the input buffer write latency reduction achieved by per-layer Toy/Tof adjustment, where 2-4× latency reduction is shown for bottleneck layers such as conv3_2 and conv4_2. Only the convolution layers for backbone CNN are shown in Fig. 10a, since the SSD layers only consume $<8\%$ of the overall latency. Fig. 11 shows several examples of drone detection results by our FPGA.

In Table IV, we compare our proposed hardware implementation with prior works that use the same or similar sized FPGAs. Among the prior works in Table IV, [19] reported the most comprehensive results including throughput (GOPS), FPS, power consumption, and mAP for Pascal VOC dataset, while using the same FPGA ZU3EG as our work. Compared to [19], we achieve 2.0-2.7× better energy-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 [19], our FPS/W is 5.5× higher than that of [19]. Unfortunately, other works [20]–[22] only reported IoU values, but not the corresponding mAP values for Pascal VOC or other datsets. While Skynet [21] 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).

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 inference 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.

VI. Conclusion

In this work, we co-optimized algorithm and hardware for high-throughput and low-power drone detection on resource-constrained FPGA devices. On the algorithm side, we employed an 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 Xilinx Zynq ZU3EG FPGA device on the Open Vision Computer version 3 (OVC3) platform.

REFERENCES

[1] M. Quigley, K. Mohta, S. S. Shivakumar, M. Watterson, Y. Mulgaonkar, M. Arguedas, K. Sun, S. Liu, B. Pfrommer, V. Kumar, and C. J. Taylor, "The Open Vision Computer: An Integrated Sensing and Compute System for Mobile Robots," *CoRR*, vol. abs/1809.07674, 2018. [Online]. Available: http://arxiv.org/abs/1809.07674

[2] "Open vision computer github repository," https://github.com/osrf/ovc/.

[3] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot Multibox Detector," in *European Conference on Computer Vision (ECCV)*, 2016, pp. 21–37.

[4] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun, "RepVGG: Making VGG-Style ConvNets Great Again," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 13 733–13 742.

[5] A. S. Rakin, Z. He, and D. Fan, "TBT: Targeted Neural Network Attack With Bit Trojan," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[6] T. Nguyen, I. D. Miller, A. Cohen, D. Thakur, A. Guru, S. Prasad, C. J. Taylor, P. Chaudhari, and V. Kumar, "Pennsyn2real: Training object recognition models without human labeling," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5032–5039, 2021.

[7] Y. Li, X. Dong, and W. Wang, "Additive Powers-of-Two Quantization: An Efficient Non-uniform Discretization for Neural Networks," in *International Conference on Learning Representations (ICLR)*, 2019.

[8] Autel, "Autel x-star," https://www.autelrobotics.com/x-star-camera-drone/, October 2020.

[9] Y. Ma, T. Zheng, Y. Cao, S. Vrudhula, and J. Seo, "Algorithm-hardware co-design of single shot detector for fast object detection on FPGAs," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–8.

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

[11] E. Park and S. Yoo, "PROFIT: A Novel Training Method for Sub-4-bit MobileNet Models," in *European Conference on Computer Vision (ECCV)*, 2020, pp. 430–446.

[12] J. Choi, S. Venkataramani, V. Srinivasan, K. Gopalakrishnan, Z. Wang, and P. Chuang, "Accurate and Efficient 2-bit Quantized Neural Networks," in *Conference on Machine Learning and Systems (MLSys)*, 2019.

[13] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 2704–2713.

[14] M. Lin, R. Ji, Z. Xu, B. Zhang, Y. Wang, Y. Wu, F. Huang, and C.-W. Lin, "Rotated Binary Neural Network," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[15] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1806.08342*, 2018.

[16] Y. Zhao, X. Gao, X. Guo, J. Liu, E. Wang, R. Mullins, P. Y. Cheung, G. Constantinides, and C.-Z. Xu, "Automatic generation of multi-precision multi-arithmetic cnn accelerators for fpgas," in *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2019, pp. 45–53.

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

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

[19] F. Li, Z. Mo, P. Wang, Z. Liu, J. Zhang, G. Li, Q. Hu, X. He, C. Leng, Y. Zhang, and J. Cheng, "A system-level solution for low-power object detection," in *IEEE/CVF International Conference on Computer Vision Workshops*, 2019.

[20] D. Wu, Y. Zhang, X. Jia, L. Tian, T. Li, L. Sui, D. Xie, and Y. Shan, "A high-performance CNN processor based on FPGA for MobileNets," in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*, 2019, pp. 136–143.

[21] X. Zhang, C. Hao, H. Lu, J. Li, Y. Li, Y. Fan, K. Rupnow, J. Xiong, T. Huang, H. Shi *et al.*, "Skynet: A champion model for DAC-SDC on low power object detection," *arXiv preprint arXiv:1906.10327*, 2019.

[22] X. Xu, X. Zhang, B. Yu, X. S. Hu, C. Rowen, J. Hu, and Y. Shi, "DAC-SDC Low Power Object Detection Challenge for UAV Applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 2, pp. 392–403, 2021.