Dynamic Partial Reconfiguration of FPGAs for Energy-Efficient Machine Learning Inference in IoT Systems

Ethan Chen, Junting Deng, Chia Jen Cheng, Jiachen Xu, John Kan, Yuyi Shen, and Vanessa Chen
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA, USA
ethanchen@cmu.edu, {juntingd, chiajenc, jxu3, johnkan, yuyis1}@andrew.cmu.edu, and vanessachen@cmu.edu

***Abstract*—This work introduces a dynamic partial reconfiguration (PR) approach for field-programmable gate arrays (FPGAs) to enhance the performance, energy efficiency, and adaptability of IoT systems. By enabling run-time reconfiguration, PR allows flexible allocation of resources for machine learning (ML) tasks, minimizing energy consumption and hardware demands. The method replaces fixed-length floating-point arithmetic with variable bit-width representations, reducing memory usage and computational complexity without sacrificing accuracy. Demonstrated on Graph Convolution Network (GCN)-YOLO models, this approach efficiently handles diverse ML layers while achieving significant improvements in resource utilization, latency, and energy savings. These results establish a scalable framework for real-time edge intelligence in IoT applications.**

Keywords — FPGA, Dynamic Partial Reconfiguration, GCN

I. Introduction

The partial reconfiguration (PR) dynamically reconfigures field-programmable gate array (FPGA) hardware to optimize resource utilization across various applications from data centers to edge devices [1-5]. This paper presents utilizing dynamically partial reconfiguration as a solution to the competing demands of high performance, good energy efficiency and low latency in dynamic environments. Dynamic resource utilization will be implemented via partial reconfiguration for flexible computing systems to perform task-dependent sensing and machine-learning-based decision making while ensuring continuity of mission-critical tasks. Advances in PR within field-programmable gate arrays offers run-time reconfiguration of hardware while providing potentially lower energy for greater functionality. While FPGAs provide a quickly reconfigurable hardware platform, much of the issue with FPGA in its present usage in nearly any scenario is that it requires more energy than an application-specific integrated circuit (ASIC) for the same task. For the conventional FPGA-based design, a large amount of energy and hardware resource must be allocated for floating-point arithmetic involving forward pass, gradient calculation, and backpropagation for the required precision while implementing machine learning (ML) models. The proposed method will significantly improve the resource usage and computing efficiency of FPGA to surpass ASIC-comparable solutions through dynamically exploiting reconfigurable algorithm-hardware co-design.

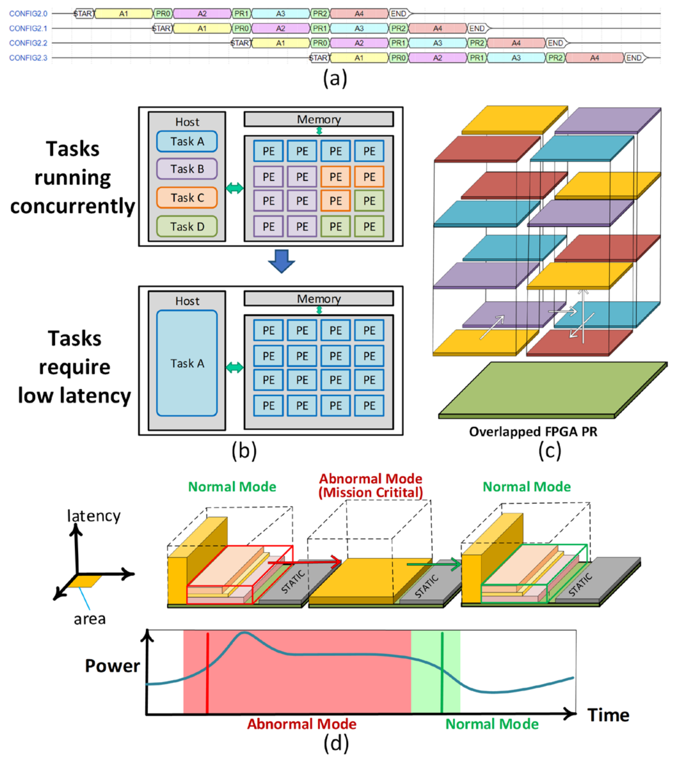

Fig. 1. Dynamic partial reconfiguration of FPGAs enables on-the-fly accommodation of unmapped code segments, adapting to computing demands for sequential, pipelined, and parallel operations as required.

II. Partial Reconfiguration

This approach can leverage reconfigurable regions within FPGAs to accommodate code partitions outside predefined clusters like CPU, GPU, or accelerators, automatically generating efficient hardware designs for specific computation types. Fig. 1(a) illustrates examples of reconfiguration scenarios. CONFIG 0-3 represents the sequential utilization of regions like A{1,2,3,4}, each potentially containing pipelined regions. In event-driven scenarios, Fig. 1(b) shows tasks running concurrently, replaced by a low-latency critical function when

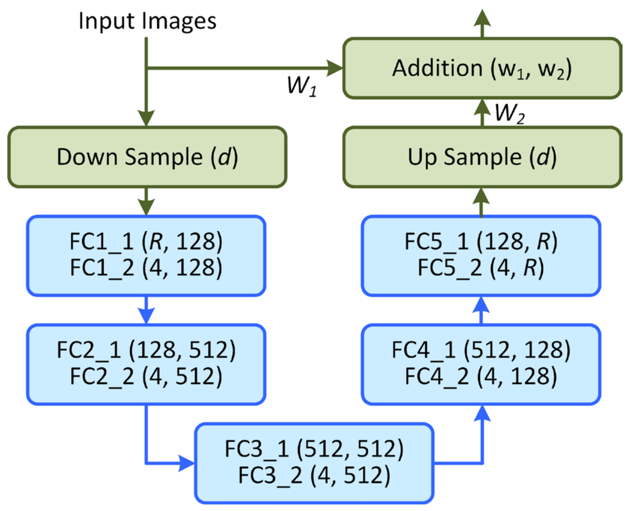

Fig. 2. The five-layer GCN architecture used in this work, where d represents the downsampling rate and R indicates the input feature dimension.

TABLE I: YOLOv5x-GCNx Model Size

Model	**Data**	**# of Parameters (M)**
Interface	Input	4.19
	Output	4.39
GCN	Weights	33.6
	Activations	4.19
YOLO	Weights	103
	Activations	1,094

needed. Fig. 1(c) demonstrates the simultaneous utilization of different fabric sections with distinct circuits, optimizing resource usage. Fig. 1(d) outlines normal and abnormal operations. Proper optimization and scheduling supervised by a "static" region (grey block) are essential. In normal mode, non-critical functions (green container) can overlap or reconfigure independently, while in abnormal mode, critical functions (gold block) require high performance. After returning to normal, non-critical functions resume their operation. Further benchmarks will assess accuracy and execution time versus resource utilization, especially during events that trigger increased accuracy requirements. Testing will evaluate power consumption, reconfiguration latency, and throughput over runtime on real-world data. These PR advancements offer runtime hardware adaptation, reducing energy consumption, execution time, and compilation errors, while enhancing functionality by ensuring efficient execution of specialized code partitions on FPGAs.

III. FPGA Implementation

A. Architecture Overview

To meet both resource usage and accuracy requirements, we implement our object detection model using a dynamically reconfigurable accelerator on an FPGA. Two main challenges complicate this implementation. First, the hardware must support diverse layer types with unique characteristics: the graph convolutional network (GCN) in Fig. 2 primarily involves fully connected layers with Sigmoid activations, while the YOLO segment contains sequences of convolutional layers, batch normalization, and SiLU activations in varying configurations. Second, achieving high accuracy on a high-resolution dataset like VEDAI [6] ($1024 \times 1024 \times 4$) demands a model that is both deep and wide, making the static mapping of all layers onto the FPGA impractical. Many layers have weights and activations that exceed the FPGA's on-chip memory capacity of 0.97 MB in an Ultra96-V2 [7]. Overall, the model comprises 10 fully connected layers, 125 convolutional layers, and 131 activation functions, as outlined in Table I.

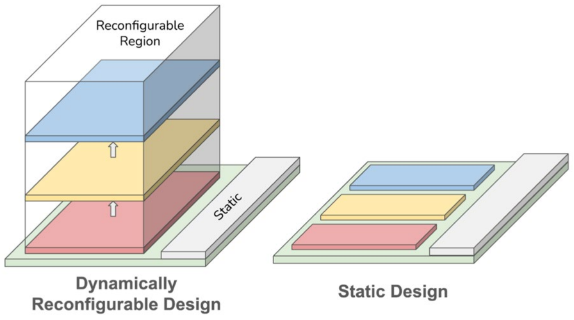

Fig. 3. Dynamically reconfigurable design vs. static design of FPGA. Dynamic reconfiguration allows for time-multiplexing of workloads. In contrast, static design can lead to performance degradation due to resource contention between distinct workloads.

To fulfill our model's need to support diverse layers while meeting the resource and area constraints, we leverage FPGA dynamically partial reconfiguration [8]. As illustrated in Fig. 3, this approach allows the FPGA to be divided into static and reconfigurable regions, enabling runtime updates by loading configuration files without disrupting ongoing executions. These configuration files are stored in main memory, and the time required for reconfiguration is proportional to the size of the reconfigurable region [9]. Our implementation includes 1 GCN and 3 CNN accelerator modules with dynamic reconfiguration loading and scheduling managed by the ARM Cortex-A53 cores. Input images, model parameters, and intermediate activations are stored in main memory and transferred via Direct Memory Access (DMA) for efficient processing. The FPGA fabric is fully allocated to dynamic reconfiguration, as smaller partitions cannot accommodate substantial workloads.

B. Hardware Design

This subsection outlines the implementation and key considerations for the accelerator modules. Given the model's size, weights and activations are loaded in tiles to fit within hardware constraints. Here, T_x denotes the tile size, and x represents the index of a parameter set along the X dimension.

GCN: The graph convolutional network consists of 10 fully-connected layers, where each layer processes inputs with dimensions $Row \times Column$ and weights sized $Depth \times Column$, producing an output of dimensions $Row \times Column$

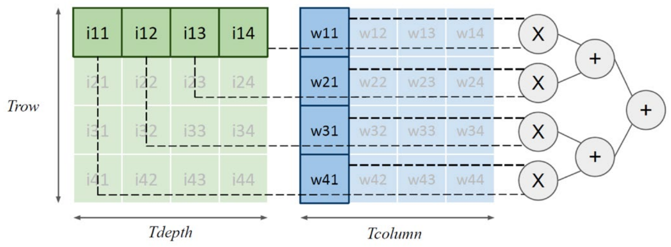

Fig. 4. One tiled and pipelined dot product operation, where i and w denote input and weight, respectively.

TABLE II: Variations of Convolutional Layer Dimensions

Dimension	Variation
S	1, 2
K	1, 3
O_r, O_c	64, 128, 256, 512, 1024
O_d	51, 80, 160, 320, 640, 1280
I_d	4, 80, 160, 320, 640, 1280, 1920, 2560

sized. Since activations are relatively small in this section of the model, most inter-layer communications occur on-chip, minimizing slower accesses to main memory. Within each fully-connected layer, inputs and weights are loaded in tiles of $T_{row} \times T_{column}$ and $T_{depth} \times T_{column}$ elements, respectively. These tiles are then fed into a multiply-accumulate (MAC) tree, which performs pipelined dot products, as shown in Fig. 4.

YOLO: The standard 2D-convolution layer (Conv2d) takes in I_d input feature maps and produces O_d output feature maps, each with dimensions $O_r \times O_c$. Each input feature map is convolved with O_d sets of weights, which slide over $K-by-K$ regions of the input with a stride of S. The relationship between input dimensions and these parameters is given by: $I_r/I_c = S \cdot (O_r/O_c - 1) + K$. The convolution operation can be described by the following equation, where O, I, W, and B denote output, input, weights, and bias, respectively.

$$O[o_d][o_r][o_c] = \text{Conv2d}(I, W, B)$$
$$= \sum_{i_d}^{I_d} \sum_{k_i}^{K} \sum_{k_j}^{K} W[o_d][i_d][k_i][k_j]$$
$$\times I[i_d][S \cdot o_r + k_i][S \cdot o_c + k_j] + B[o_d]$$

To efficiently manage the large parameters and activations, tiling is applied across dimensions O_r, O_c, O_d, I_r, I_c, and I_d. Given the substantial size of activation and weights, we utilize a dataflow architecture as described in [10], which allocates all on-chip memory to tile-sized buffers, thereby reducing access to main memory. As shown in Fig. 5, the convolution operation is parallelized using T_{od} pipelined MAC tree, each with a width of

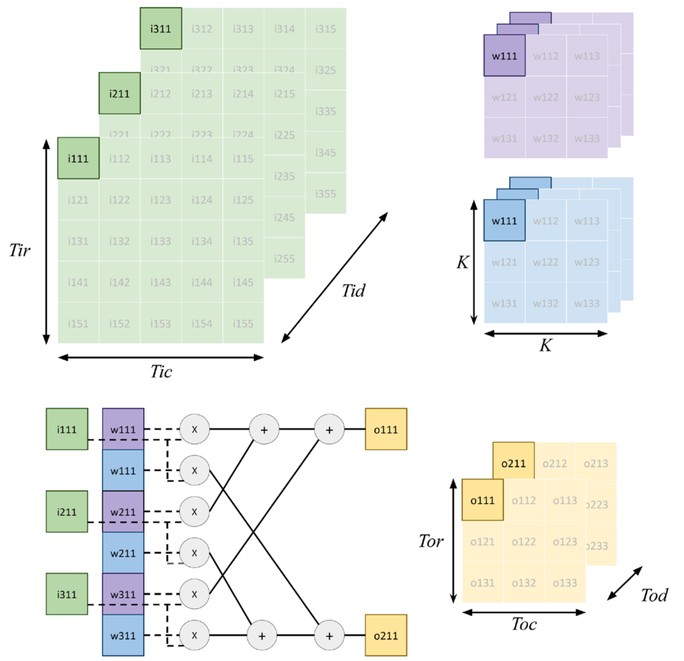

Fig. 5. One tiled and pipelined convolution operation, where i, w, and o denote input, weights, and output, respectively.

T_{id}. The degree of parallel computations is determined by T_{id} and T_{od}, where the tiling sizes are constrained by available hardware resources and the specific layer dimensions listed in Table II. More specifically, the use of adders and multipliers scales with $T_{id} \times T_{od}$ and the on-chip buffer size, given by $T_{or} \times T_{oc} \times T_{od} + T_{ir} \times T_{ic} \times T_{id} + K^2 \times T_{id} \times T_{od}$, must remain within the bounds of the total available on-chip memory.

Batch Normalization: Batch normalization (BN) is widely used to promote faster and more stable model convergence by normalizing inter-layer activations to achieve zero mean μ and unit standard deviation σ. During training, scaling and shifting parameters (γ, β) are learned to adjust these normalized values. A small constant ϵ is used to avoid division by zero and improve numerical stability. To reduce overall computation and data transfer, these BN parameters are merged with the convolution weights after training, resulting in updated weights W', and biases, B', which are calculated as functions of the learned BN parameters, as illustrated in the following equation.

$$\text{BN}(O[o_d][o_r][o_c])$$
$$= \gamma[o_d] \cdot \frac{(O[o_d][o_r][o_c] - \mu[o_d])}{\sqrt{\sigma[o_d]^2 - \epsilon}} + \beta[o_d]$$
$$= \text{Conv2d}(I, W', \beta')$$
$$W' = \frac{\gamma \cdot W}{\sqrt{\sigma^2 - \epsilon}}, \quad B' = \beta + \frac{\gamma \cdot (B - \mu)}{\sqrt{\sigma^2 - \epsilon}}$$
$$B' = \beta + \gamma \cdot (B - \mu)/\sqrt{\sigma^2 - \epsilon}$$

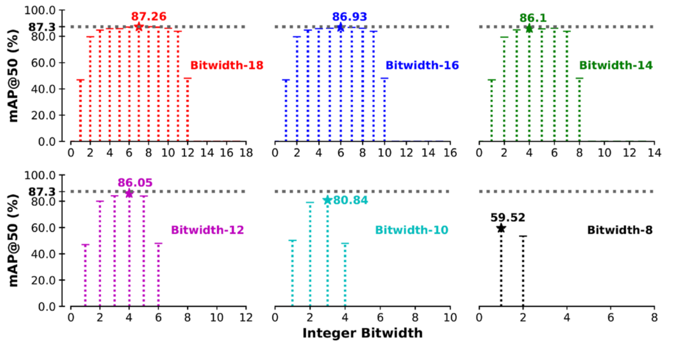

Fig. 6. Effects of varying integer bit-widths on model accuracy at different total bit-widths.

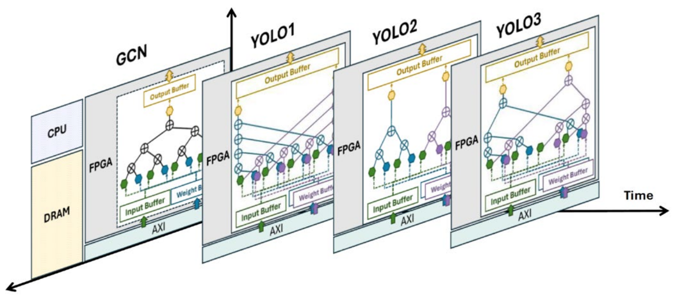

Fig. 7. Dynamically reconfigured detection model with 4 accelerator modules.

Activation: To optimize DSP utilization and memory transfer, YOLOv5x-GCNx is initially trained in floating-point precision and subsequently quantized to a custom 16-bit fixed-point format, with 7 bits for the integer part and 9 bits for the fractional part. Parameter and activation bit-widths are carefully selected to minimize accuracy loss, as illustrated in Fig. 6. Additionally, the Sigmoid activation function is approximated with a low-cost piecewise linear function following the approach in [11]. This function is also adapted for the SiLU activation function, leveraging the relation $\mathrm{SiLU}(x) = x \times \mathrm{Sigmoid}(x)$. To streamline processing, activations are fused with the subsequent fully-connected or convolutional layer.

Dynamic Reconfiguration: Among the four accelerator modules, one is specifically customized for GCN, while the others represent three variations of convolutional layers used in the YOLO component in Fig. 7. The average parallelism of each convolutional layer is determined by the tiling along the input and output feature map dimensions. Consequently, the implementations of these modules vary in their tiling amounts to conform to hardware resource constraints and the specific layer dimensions detailed in Tables II and III. Since the allocated reconfigurable region is essentially the entire FPGA fabric, the sizes of the configuration files (4 MB) and the reconfiguration times (10 ms) remain relatively constant across all modules.

Without dynamic reconfiguration, two straightforward strategies can be employed: (1) converting all operations into matrix-matrix multiplications, or (2) converting all operations into convolutions. The im2col method realizes convolutional layers as matrix-matrix multiplications, thereby eliminating the need for reconfiguration. However, this approach transforms input feature maps into columns, resulting in memory redundancy of $K^2 \times O_r \times O_c/(I_r \times I_c)$ on the input feature maps. This redundancy worsens the already large size of activations, which dominate off-chip data movement [12]. Conversely, the hardware illustrated in Fig. 5 can perform the matrix-matrix multiplications utilized in fully connected layers by imposing specific constraints on certain dimensions [12]. Specifically, these constraints require that $O_r = 1, O_c = 1, S = 1$, and $K = 1$, which leads to the following equation . However, these constraints are impractical, as they severely limit the output feature map dimensions.

$$O[o_d] = Conv2d(I, W, B) = \sum_{i_d}^{I_d} W[o_d][i_d] \times I[i_d] + B[o_d]$$

TABLE III: Convolutional Layer Accelerator Modules and Tiling Sizes

Accelerator Module	T_{oc}	T_{ic}	T_{or}	T_{ir}	SiLU
1	8	16	64	64	V
2	32	4	32	64	V
3	3	64	64	64	X

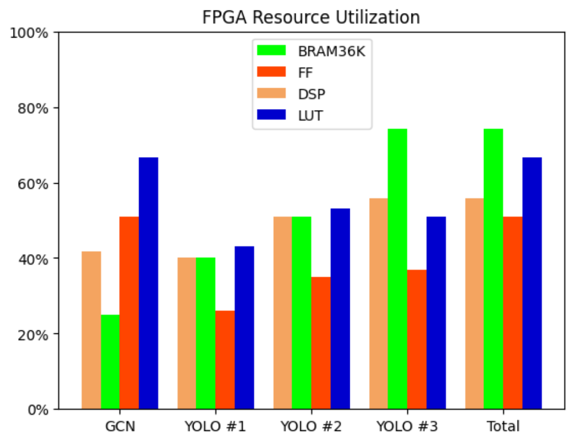

Fig. 8. FPGA resource utilization for all accelerator modules after place and route on the Ultra96-v2 board.

C. Experimental Results

In this subsection, we discuss the hardware cost associated with implementing YOLOv5x-GCNx on the FPGA and compare it to existing approaches for accelerating object detection.

A detailed breakdown of hardware resources for all accelerator modules is illustrated in Fig. 8. As shown, the FPGA has limitations in accommodating any pair of accelerator modules due to insufficient resources. This challenge is effectively addressed through the dynamic reconfiguration of the fabric logic. The total power consumption of the model is 3.4 W, which includes both static and dynamic power; of this, 1.7 W is contributed by the ARM CPUs.

TABLE IV: Comparison with Previous Works

Metric	This Work	ACAI '21 [13]	ICET '20 [14]
Platform	Ultra96-V2	ZYNQ-7035	ZCU102
Frequency	150 MHz	100 MHz	300 MHz
Precision	16-b	16-b	16-b
Data	VEDAI	COCO	COCO
Channels	RGB+IR	RGB	RGB
Resolution	**1024×1024**	416×416	416×416
#FC	10	0	0
#Conv2d	125	13	9
Size(GFLOPS)	**2530**	5.56	29.5
FF	71,760	-	90,989
LUT	46,929	61,700	95,136
DSP	201	485	609
BRAM	161	248	491
Power (W)	**3.4**	3.71	11.8

Several studies on object detection using FPGAs are presented to compare with our implementation in Table IV. Given the variations in hardware platforms, objectives, datasets, accuracies, and model sizes across these approaches, a direct comparison is challenging. Notably, the model sizes and resolutions of existing methods are significantly lower than those of our proposed work. To ensure fairness, all listed studies focus on object detection, employ some variant of YOLO, and utilize the same numerical precision. Our analysis prioritizes model accuracy and resource usage, which are reflected in chip area and total power consumption, demonstrating that the proposed approach can efficiently process large model sizes with high-resolution datasets.

IV. Conclusions

Advances in PR within FPGAs enable run-time reconfiguration of hardware, allowing for adaptive functionality while optimizing energy efficiency. In this paper, we present a practical implementation strategy that leverages dynamic reconfiguration to deploy the proposed model while adhering to stringent resource and power constraints on the Ultra96-V2 board. Our experimental results demonstrate that the proposed approach not only achieves superior power efficiency compared to existing works but also processes images with six times higher resolution, highlighting the scalability and effectiveness of the method. These findings underscore the potential of PR-based FPGA designs for energy-efficient, high-performance computing in resource-constrained environments.

Acknowledgment

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under Award HR0011-24-2-0350, and in part by the National Science Foundation under Grant 1953801.

References

[1] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger, "A configurable cloud-scale DNN processor for real-time AI," in 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), pp. 1–14, 2018.

[2] L. Cardona, B. Lorente, and C. Ferrer, "Partial crypto-reconfiguration of nodes based on FPGA for wsn," in 2014 International Carnahan Conference on Security Technology (ICCST), pp. 1–4, 2014.

[3] K. Papadimitriou, A. Dollas, and S. Hauck, "Performance of partial reconfiguration in FPGA systems: A survey and a cost model," ACM Trans. Reconfigurable Technol. Syst., vol. 4, Dec 2011.

[4] H. Al-Aqrabi, A. P. Johnson, and R. Hill, "Dynamic multiparty authentication using cryptographic hardware for the internet of things," in 2019 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), pp. 21–28, 2019.

[5] R. Garcia, A. Gordon-Ross, and A. D. George, "Exploiting partially reconfigurable FPGAs for situation-based reconfiguration in wireless sensor networks," in 2009 17th IEEE Symposium on Field Programmable Custom Computing Machines, pp. 243–246, 2009.

[6] S. Razakarivony and F. Jurie, "Vehicle detection in aerial imagery : A small target detection benchmark," Journal of Visual Communication and Image Representation, vol. 34, pp. 187–203, 2016.

[7] https://www.xilinx.com/products/boards-and-kits/1-vad4rl.html.

[8] K. Vipin and S. A. Fahmy, "Fpga dynamic and partial reconfiguration: A survey of architectures, methods, and applications," ACM Comput. Surv., vol. 51, July 2018.

[9] M. Nguyen, N. Serafin, and J. C. Hoe, "Partial reconfiguration for design optimization," in 2020 30th International Conference on Field-Programmable Logic and Applications (FPL), pp. 328–334, 2020.

[10] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '15, (New York, NY, USA), p. 161–170, Association for Computing Machinery, 2015.

[11] I. Tsmots, O. Skorokhoda, and V. Rabyk, "Hardware implementation of sigmoid activation functions using fpga," in 2019 IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM), pp. 34–38, 2019.

[12] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," Proceedings of the IEEE, vol. 105, no. 12, pp. 2295–2329, 2017.

[13] Q. Xiong, C. Liao, Z. Yang, and W. Gao, "A method for accelerating yolo by hybrid computing based on arm and fpga," in Proceedings of the 2021 4th International Conference on Algorithms, Computing and Artificial Intelligence, ACAI '21, (New York, NY, USA), Association for Computing Machinery, 2022.

[14] S. Zhang, J. Cao, Q. Zhang, Q. Zhang, Y. Zhang, and Y. Wang, "An fpga-based reconfigurable cnn accelerator for yolo," in 2020 IEEE 3rd International Conference on Electronics Technology (ICET), pp. 74–78, 2020.