IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2023.DOI

Real-Time Multi-Task ADAS Implementation on Reconfigurable Heterogeneous MPSoC Architecture

GUNER TATAR[1,2] (Student Member, IEEE), SALIH BAYAR[2]

[1]Department of Electrical Electronic Engineering, Fatih Sultan Mehmet Vakif University, Istanbul, TURKIYE (e-mail: gtatar@fsm.edu.tr)
[2]Department of Electrical and Electronic Engineering, Marmara University, Istanbul, TURKIYE (e-mail: salih.bayar@marmara.edu.tr)

Corresponding author: Guner TATAR (e-mail: gtatar@ fsm.edu.tr).

ABSTRACT The rapid adoption of Advanced Driver Assistance Systems (ADAS) in modern vehicles, aiming to elevate driving safety and experience, necessitates the real-time processing of high-definition video data. This requirement brings about considerable computational complexity and memory demands, highlighting a critical research void for a design integrating high FPS throughput with optimal Mean Average Precision (mAP) and Mean Intersection over Union (mIoU). Performance improvement at lower costs, multi-tasking ability on a single hardware platform, and flawless incorporation into memory-constrained devices are also essential for boosting ADAS performance. Addressing these challenges, this study proposes an ADAS multi-task learning hardware-software co-design approach underpinned by the Kria KV260 Multi-Processor System-on-Chip Field Programmable Gate Array (MPSoC-FPGA) platform. The approach facilitates efficient real-time execution of deep learning algorithms specific to ADAS applications. Utilizing the BDD100K, KITTI, and CityScapes datasets, our ADAS multi-task learning system endeavours to provide accurate and efficient multi-object detection, segmentation, and lane and drivable area detection in road images. The system deploys a segmentation-based object detection strategy, using a ResNet-18 backbone encoder and a Single Shot Detector architecture, coupled with quantization-aware training to augment inference performance without compromising accuracy. The ADAS multi-task learning offers customization options for various ADAS applications and can be further optimized for increased precision and reduced memory usage. Experimental results showcase the system's capability to perform real-time multi-class object detection, segmentation, line detection, and drivable area detection on road images at approximately 25.4 FPS using a 1920x1080p Full HD camera. Impressively, the quantized model has demonstrated a 51% mAP for object detection, 56.62% mIoU for image segmentation, 43.86% mIoU for line detection, and 81.56% IoU for drivable area identification, reinforcing its high efficacy and precision. The findings underscore that the proposed ADAS multi-task learning system is a practical, reliable, and effective solution for real-world applications.

INDEX TERMS ADAS, Deep learning, Deep processing unit, Memory allocation, Multi-task learning, MPSoC-FPGA architecture, Vitis-AI, Quantization aware training

I. INTRODUCTION

The rise of deep learning (DL) and vision-based technology has ushered in a new era of discussion surrounding autonomous driving. An autonomous driving system is a complex construct comprising numerous sensors and modules performing specific functions. The capacity to perceive and react to various environmental constituents, such as surrounding vehicles, pedestrians, and traffic signs, is fundamental to a robust autonomous driving system. The advent of DL has catalyzed remarkable advancements over conventional algorithms in these processes. DL methods can be applied to diverse machine vision tasks, yielding accuracy, speed, and reliability enhancements.

Although DL finds applicability across multiple domains, including healthcare, finance, and entertainment, its prominence is particularly evident in Advanced Driver Assistance Systems (ADAS) tasks. For instance, architectures such as VGG-Net [1], GoogleNet [2], DenseNet and ResNet [3] have been proposed for image classification and have successfully executed all tasks. In object recognition, multi-stage

detectors, such as R-CNN [4] and Faster R-CNN [4], and single-stage object detection approaches, such as SSD [5] and YOLO [6], have been effectively deployed in contemporary applications. Similarly, models such as DeconvNet [7] and SegNet [8] are recognized for their proficiency in semantic segmentation, while LaneNet [9] and VPGNet [10] have demonstrated success in lane detection tasks.

A thorough analysis of each model reveals that their complexities can be attributed to the multitude of layers and parameters involved. This intricacy originates from the ambition of designers to augment the performance of Deep Neural Networks (DNNs) by fabricating more extensive architectures. Indeed, such complexity presents difficulties in training, extracting, and implementing deepening architectures, which grow concurrently with escalating requirements. Moreover, employing each specified DL model for a unique task renders their concurrent utilization in multi-tasking applications more feasible.

Two distinct strategies emerge as potential solutions to circumvent this issue. The initial strategy involves the operation of each task-specific model on distinct hardware platforms, a method unavoidably associated with financial implications. In contrast, the second strategy pertains to cultivating and expanding multi-tasking learning models. As the terminology suggests, multi-tasking learning operates on the premise that numerous tasks, interconnected and running on single hardware using the same infrastructure, will result in increased efficiency. Though this method may slightly heighten the model's complexity for the designer, it simultaneously obviates the financial burden of running distinct learning models on the same platform. The execution of multi-task learning on constrained hardware through multi-tasking is rapidly emerging as a promising and efficacious strategy within ADAS.

This new sub-field within artificial intelligence is characterized by the simultaneous operation of multiple learning tasks using a single model, capitalizing on the disparities and similarities across tasks. This novel paradigm has the potential to proffer significant benefits across a plethora of domains. Sharing backbone encoder computations can considerably reduce overall computational complexity. Furthermore, due to the inherent and reciprocal relationships among various tasks, multi-task learning has the potential to confer superior learning efficiency and predictive accuracy.

Optimization of both software and hardware architectures is an essential step in augmenting processing speed and minimizing inference time. Thus, hardware-software co-design emerges as a vital consideration for ADAS optimization within a Multiprocessor System-on-Chip Field Programmable Gate Array (MPSoC-FPGA) context, spanning both software and hardware dimensions. This approach facilitates the maximum utilization of embedded resources, thereby achieving a high degree of energy efficiency.

Given the complexities discussed, an innovative architecture is proposed herein, specifically tailored to navigate financial constraints and meet the need for executing multiple tasks simultaneously on a designated hardware platform. The novelty of the present study primarily lies in the improvements instituted within the hardware accelerator platform, synchronously with software enhancements. In forthcoming chapters, thorough elucidation of the unique software enhancements, the collaborative methodology involving hardware-software co-design, and the specialized memory allocation and pipeline structures implemented on the hardware accelerator side will be provided. This comprehensive examination aims to furnish a more in-depth understanding of the various components and the innovative approaches employed in the proposed architecture.

A. MOTIVATION AND BACKGROUND

DL has a wide range of uses in ADAS to increase autonomous driving capability and prevent possible loss of life and property. DL was considered for ADAS applications due to its proficiency in computer vision tasks, which are deemed essential for the functionalities of ADAS. DL models, such as CNNs and DNNs, are highly effective in processing and extracting information from image and video data obtained from sensors. They can automatically learn meaningful features, eliminating the need for manual feature engineering. DL also excels at complex pattern recognition and enables end-to-end learning, simplifying system architecture. Additionally, DL models are scalable, adaptable, and can generalize well to different driving scenarios. While other learning algorithms are used in certain components of ADAS, DL's strengths in computer vision make it highly applicable in ADAS applications. Thus, we concentrated on developing and accelerating DL-based ADAS algorithms.

Hardware selection should be made with ADAS tasks for the specified DL models to be run at the desired performance and without resource consumption problems by making software optimizations on hardware accelerator platforms. Custom circuit blocks and chip architectures are accelerated to create hardware accelerators.

Similarly, operational cores are employed to strike a balance between performance and functional flexibility. Speed, image size, power consumption, flexibility, accuracy, and memory are all constraints in embedded systems and applications. Understanding the benefits of these high-tech products requires a thorough understanding of DL models and hardware acceleration structures. Since DL algorithms are based on neural networks, understanding their fundamental architecture and configuration is critical.

Today, most DL applications are cloud-based in open-source, public clouds with companies like Google, Microsoft, and Amazon [14]. DL networks of these companies analyze large amounts of data and carry out tasks by the CPUs of these companies filled with thousands of servers. These CPU-based structures are sometimes insufficient for analyzing such large amounts of data. In such cases, CPUs are coupled with other hardware, such as FPGA, GPU or ASIC-based accelerators, for utilizing the DL networks. For example, while many companies use computers equipped with multi-

core GPUs, Google uses its Tensor processor unit, and Microsoft uses an FPGA system [14], [15]. What is striking in the literature studies and research, due to their inherent parallelism capability of [16] application-specific platforms FPGAs and ASICs' have recently gained popularity. Inherent parallelism is beneficial for DL model training because of reducing the execution time and accuracy of programs. In this way, the designer can focus on the work of the DL model by eliminating the parallelization workload. To run the DL model on MPSoC architectures (ARM and FPGA together), designers require good hardware-software co-design. Because the parts with the highest computational density are executed on the Deep Processing Unit (DPU) side, while the rest are executed on the ARM processor side.

The selection of the KRIA KV260 MPSoC FPGA for the implementation of this work was aligned with the requirements, primarily due to its compatibility with the ADAS application under study. The KRIA KV260 MPSoC FPGA aptly balances computational power, flexibility, and power efficiency, presenting specific features such as high-speed interfaces and dedicated hardware accelerators that closely align with our ADAS application's needs. Furthermore, the KRIA KV260 MPSoC FPGA boasts significant support and a mature ecosystem for deep learning development, facilitating more straightforward implementation and optimization of our deep learning algorithms. Similarly, the KRIA KV260 MPSoC FPGA is frequently utilized for applications including ADAS, robotics, and industrial automation, requiring a balance between processing power and FPGA flexibility.

B. RELATED WORK

This section briefly presents some of the most advanced previous work on multi-object detection, semantic segmentation, and multi-tasking ADAS systems. There are many studies in the literature related to ADAS. However, it is only possible to compare some studies precisely with each other. For this reason, we have taken similar parts of the studies conducted in the last years closest to the study we propose. We have determined the most common work points in terms of tasks, data-sets, software-hardware co-design, the hardware used, FPS values, and the accuracy of inference. Initially, we provided concise summaries of the critical aspects of state-of-the-art studies, structuring them semantically from single-task studies to multi-task studies. Subsequently, we compiled encapsulating table studies that briefly explained studies alongside other distinct studies. The inclusion of this Table 1 is pivotal, as it presents the reader with detailed insights and critical points pertinent to our subject matter in a comprehensive and accessible format.

What can be clearly seen in the latest state-of-the-art studies in Table 1 is the high rate of using GPU-based hardware architecture. One of the primary reasons for this is that a GPU with hundreds of cores capable of processing thousands of threads in parallel can accelerate the performance of some software by about 100 times when compared to other architectures. Likewise, the complex computational problems we expect computers to solve have increasingly parallel structures. For example, consider the massive amounts of video processing, image analysis, signal processing, and DL streams that must happen reliably and in real-time to run a self-driving vehicle. Furthermore, a GPU must achieve this processing speed in power-constrained systems like battery-powered electric vehicles while providing greater power and cost efficiency. Another essential feature is that GPU-based architectures are easy to program with advanced software packages, including CUDA, TensorFlow and PyTorch.

Indeed, while GPUs provide both good hardware acceleration capabilities and excellent usability, high energy consumption and high cost are constraints for battery-powered devices. Therefore, designers prefer architectures with similar GPU features as well as less energy-consuming and cost-effective architectures. Recently, application-specific platforms (e.g. FPGAs, ASICs) are becoming more popular due to their inherent parallelism capability, which is advantageous for DL algorithm training in program execution time and accuracy [16]–[20].

FPGAs are highly desirable due to their low energy consumption, inherent parallel processing capabilities, and fast results even at low frequencies [19], [20]. This means they will have a low-cost, high-performance hardware accelerator that designers prefer. While low-level hardware description languages (e.g., VHDL, Verilog) were formerly the only way to program an FPGA, the introduction of unified software platforms such as Xilinx-Vitis and Vitis-AI [21] have made coding in C/C++ and Python possible. In addition, the integration of high-level frameworks TensorFlow and Caffe with high-level languages such as C/C++ and Python, thanks to AMD-Xilinx Vitis, has increased the use of FPGAs in DL applications.

Based on given information, we highlighted the strengths and weaknesses of the studies that showed the closest similarity to our research from the references given in Table 1.

For example, in their insightful work, Ghorbel et al., [27], proposed a method that utilizes GPU acceleration to parallelize the eyes detection algorithm based on Viola and Jones, a development aimed at designing an innovative smart wheelchair. The authors subsequently apply this research to craft a human-machine interface to govern intelligent wheelchair control. Notably, their work incorporates a significant element of software-hardware co-design. However, their study appears to lack comprehensive coverage of multi-task learning, which warrants further exploration to augment the design and performance of intelligent wheelchairs. In the concluding remarks of their study, Ghorbel et al. explicitly acknowledge the persistent challenge posed by GPUs in the context of energy consumption and efficiency. This issue is particularly pronounced in electronic systems powered by batteries, potentially limiting the viability and practical applicability of their research in real-world settings. Moreover, the study employs the Omap4 4460 platform, which may be prohibitive considering the price-performance ratio. It is crucial to examine whether more cost-effective alternatives

TABLE 1. ADAS multi-tasks implementation in the literature

Ref.	Drivable area & Line detection	Semantic segmentation	Multi object det.	Data-sets	HW-SW co-design	Platform	FPS	Evaluation (%)
[27]	Y	N	N	FDDB	Y	C-based GPU Omap4460	55ms	87.33 (Accu.)
[22]	Y	N	N	COCO	N	NVIDIA Jetson Nano	20	60 (Accu.)
[24]	Y	Y	N	Cityscapes KITTI	N	NVIDIA RTX TITAN	N	72.32 (mIoU)
[25]	Y	Y	N	tuSimple Cityscapes BDD100K	N	NVIDIA RTX 2080Ti GPU	N	95.73 (Accu.) 70.9 (F1) 57.03 (Accu.)
[26]	Y	Y	N	Their	N	GTX 1080Ti Xavier NX	50 15	94.51 (F1)
[23]	Y	Y	N	Cityscapes	N	NVIDIA RTX 2080Ti & ROS	N	N
[33]	Y	Y	N	KITTI	N	NVIDIA TITAN GPU	N	94.21 (AP)
[28]	N	N	Y	VOT VBT	Y	Zynq UltraScale+ MPSoC ZCU3EG	123 53.3	66.25 (mIoU)
[29]	N	N	Y	Their	N	NVIDIA GTX 1060	16.68	0.986 (mAP)
[30]	N	N	Y	GTSRB	N	NVIDIA Jetson Xavier AGX & Xavier Nx	43.59 23.17	0.621 (mAP)
[32]	N	N	Y	CVC-09 FLIR ADAS OSU, KAIST	N	NVIDIA Jetson Nano	3	84.1 (mAP)
[34]	Y	N	Y	Air learning database	Y	Xavier NX Jetson TX2	6	91 (Accu.)
[31]	N	Y	N	Cityscapes	N	GTX TITAN X Maxwell (GPU)	30	70 (mIoU)
[35]	Y	Y	Y	N	N	NVIDIA Jetson Xavier & TI TDA2x	10 15	39.78 (mAP)
ADAS Multi task- learning (Proposed study)	Y	Y	Y	BDD100K Cityscapes KITTI	Y	KRIA KV260 MPSoC FPGA	25.4	Object det.: 51 (mAP) Segment.: 56.62 (mIoU) Drivable: 81.56 (mIoU) Line det.: 43.86 (IoU)

* Y/N: Yes/No, IoU: Intersection over Union, mIoU: mean Intersection over Union, mAP: mean Average Precision, FPS: Frame per Second, Accu: Accuracy

could achieve comparable, if not superior, results in designing intelligent wheelchairs and their respective human-machine interfaces. This would improve the accessibility and affordability of the technology for a broader user base.

In their study, the authors [28] undertook a comparative analysis of three disparate deep convolutional neural network hardware accelerator implementation methods. These encompassed coarse-grained, fine-grained, and sequential Vitis-AI strategies. Two bespoke DNN architectures were developed within the System Verilog and FINN frameworks, displaying the flexibility and applicability of these models. Notably, despite achieving high performance in terms of FPS rate, the authors did not explore multi-task implementation in their work, which leaves room for further investigation into improving computational efficiency. Another critical observation to be made about the study is their choice of the high-performance MPSoC ZCU3EG for the exclusive task of object detection. Given the processing capabilities of this particular system, it could potentially be better leveraged by distributing the computational load across multiple tasks, thereby optimizing resource use. This single-task focus leaves the potential for more comprehensive, multi-task approaches largely unexplored. Such methodologies could prove beneficial for enhancing processing efficiency and performance in future research endeavours.

An additional noteworthy study is the one conducted by Cho et al., [33], titled "Multi-task Self-supervised Visual Representation Learning for Monocular Road Segmentation," which attained a commendable 94.23% Average Precision (AP) performance. Within their research, the authors employed a multi-task framework for segmentation-based roadway identification. Their study utilized the KITTI dataset and relied on the considerably costly NVIDIA TITAN GPU as their hardware. In a distinctive shift from traditional literature, the authors examined the utility of unsupervised stereo-based indicators to acquire high-level semantic knowledge for monocular route detection. Their experimental outcomes indicated an above-average performance, albeit not significantly superior compared to our research. Our study boasts several advantages over Cho et al.'s research, such

as a broader array of utilized datasets (KITTI, BDD100K, and CityScapes), the adopted methodology, the concurrent execution of four distinct tasks, and the incorporation of a backbone encoder.

In the seminal work by Krishnan et al., [34], an in-depth exploration was conducted on the automation of domain-specific SoC design intended for autonomous vehicles, with particular emphasis on Unmanned Aerial Vehicles (UAVs). The scope of their research primarily encompassed nano, micro, and mini-UAVs, for which they utilized the capabilities of Xavier NX and Jetson TX2 to fashion domain-specific SoCs accelerators. Their research findings confirmed an acceleration factor of approximately 2.25x, 1.62x, and 1.43x, respectively, across the range of UAVs studied. It is crucial to note that the approach employed by the authors, though commendable in its results, diverges from ours in a significant aspect, namely the software-hardware co-design, which is the cardinal characteristic differentiating our study.

In addition to this distinguishing factor, our research incorporates semantic segmentation, which remarkably enhances the FPS value. This constitutes a considerable advantage when the two studies are juxtaposed. Furthermore, the original study conducted by Krishnan et al. exhibits a preference for GPU-based accelerator platforms. Despite their numerous benefits, it is widely recognized that such platforms are mainly inefficient in energy consumption, owing to their reliance on hundreds or even thousands of CUDA cores. This aspect further underscores our research's distinctiveness and potential advantages, which circumvents this significant energy inefficiency challenge.

In their substantive research, Lai et al. [35] introduced a Multi-Task Semantic Attention Network (MTSAN) designed to amalgamate segmentation and object detection functionalities for real-time applications in ADAS. Although their approach substantially reduced false alarm frequency, it did so at the cost of increased computational resources. The researchers reported an FPS rate of 10 at a resolution of 512 x 256 on NVIDIA's Jetson Xavier and a slightly improved 15 FPS at the exact resolution on Texas Instruments' TDA2x platform. However, given the considerable investment associated with NVIDIA's Jetson platforms, these FPS rates could be more impressive when juxtaposed with our model's superior performance.

Moreover, the authors characterized their hardware as low-power, a claim inconsistent with the commonly recognized high energy consumption intrinsic to GPU-based architectures. Their study further reveals a subtle under emphasis on the multi-task functionality of their model, creating an opportunity for a more thorough model evaluation and comparison. Contrasting the performance metrics of our model with those of the authors provides a clearer picture of our superiority. Enhancements on numerous fronts, including FPS, power consumption, and memory resource utilization, are observed in the presented model. It showcases superior real-time performance, exhibiting an improved FPS rate compared to [34], [35]. Furthermore, our model's mAP and mIoU results exceed those of the MTSAN model, which underscores our superior performance in object detection and segmentation tasks.

Regarding power consumption, our model operates on an MPSoC architecture, renowned for delivering robust computational performance while consuming significantly less power than GPU-based architectures. This issue summarises our model being more energy-efficient while providing full ADAS functionalities. Regarding memory resources, we leverage the Xilinx Kria KV260 platform, renowned for optimal memory resource utilization, which results in a more memory-efficient solution.

In summary, our model presents a more balanced solution for ADAS applications by providing robust performance coupled with energy efficiency and cost-effectiveness. This situation culminates in our model outperforming those proposed by [34], [35] across multiple evaluation parameters, asserting its superiority in this field.

As Table 1 shows, there are many methods and studies for ADAS development. Here, we have tried to present famous state-of-the-art outcomes closest to our work. In addition, studies involving software optimization and the use of hardware accelerators in designing ADAS applications have become popular. Hardware accelerators can take the form of CPUs, ASICs, GPUs, and FPGAs. In such event, CPUs are combined with other hardware for utilizing the DL networks. When hardware accelerators are inadequate, one or more of the platforms noted above, like CPUs, GPUs, FPGAs, and ASICs, are used together for the training inference of DL algorithms [36], [37].

Recently, application-specific platforms (e.g. FPGAs, ASICs) are becoming more popular [38] due to their structural parallelism capability, favouring DL algorithm training in program execution time and accuracy. It is to be highlighted that there is a gap in the literature on the need for a design encompassing high FPS throughput and mAP-mIoU value, better performance at a lower cost, multi-tasking on a single hardware, and integration into memory-constrained devices to enhance the performance of ADAS tasks. Therefore, undertaking an integrated hardware-software design incorporating the above-mentioned features is crucial to enable the widespread adoption of autonomous vehicle technologies.

In light of this information, we preferred the FPGA-based MPSoC architecture because of its ability to perform parallel processing, its price/performance compatibility and its more flexible structure. We have realized an efficient hardware-software co-design on the MPSoC structure. We created our own DPU architecture on the programmable logic (PL) side so that the ADAS multi-task learning yields better results than the existing studies. We arranged the computationally demanding parts of the DL model to be on the DPU and the video pre-processing, task allocations and inference parts on the ARM. For DPU-ARM data interaction, we used the advanced extensible interface - direct memory access (AXI-DMA) and pipeline architecture.

C. CONTRIBUTIONS

We have given the study's main contributions to the literature in a list as follows;

1) We developed ADAS multi-task learning system, which can perform several tasks, including semantic segmentation, multi-object detection, line detection, and derivable area detection, on a single piece of hardware. This approach can lead to efficient and effective development of embedded systems, particularly for ADAS applications.
2) We enhanced the efficiency of our model for resource-limited embedded devices by examining its backbone. To reduce its memory footprint on constrained platforms, we quantized the model using int-8 bits.
3) We constructed effective optimizations to the proposed model to improve its performance without affecting the accuracy of the inference.
4) We assembled the programmable DPU reserved for the convolutional neural network.
5) Through hardware-software co-design, our proposed approach offers high performance and low energy consumption when compared to other hardware architectures for similar tasks. We conducted a feasibility study by deploying the proposed model on low-power embedded devices and demonstrated real-time processing using a prototype design. Specifically, we investigated the ability to program traditional programming languages, such as Python and C++, in heterogeneous MPSoC architecture with hardware-software co-design.

The remaining paper scenario is as follows. Section II contains a discussion on the design methodology of the proposed model. Initially, an overlay is designed utilizing AMD-Xilinx Vivado 2022.1, which is then imported to the MPSoC-FPGA-based development board. The performance of software design and enhancements within the Python environment follows this. Notably, the focus here is also on the model's training, quantization, and compilation. Lastly, the experimental setup and execution of the $.xmodel$ file quantized and compiled using $vai\_q\_pytorch$ are deliberated. In Section III, a rigorous comparative analysis of our findings with analogous studies from the existing literature is undertaken, with primary emphasis on parameters such as power consumption, reliability, longevity, accuracy, and overall efficiency in order to optimize the quality of the design. Concluding the paper, Section IV encapsulates the summative observations drawn from the entirety of the article and clarifies prospective avenues for future research.

II. METHODOLOGY

This section raises a broad methodological strategy adopted for our research. It commences with a discussion on the structure of the multi-task learning network and its pertinent subtopics. Following this, an exploration of the hardware design section and its associated subtopics is presented, developed utilizing the Vitis unified software platform and the Vivado 2022.1 integrated development environment. Ultimately, the QAT for the model incorporating a ResNet-18 shared backbone encoder with SSD is addressed in this work.

A. INDUCTION

Designers extensively use DNNs in ADAS, and this study focuses on improving and accelerating DL-based ADAS algorithms. The development and acceleration of DNNs have been possible with software optimizations. Optimizations are usually possible by executing the algorithm faster, parallelizing it or incorporating libraries such as PThreads, OpenCL, and MPI into the algorithms. In addition, software optimizations are aimed at preventing bottlenecks of algorithms by using the specified methods. Unfortunately, studies on DNNs have caused models to be more complex and consist of millions or even billions of parameters. This concern brings about the inadequacy of software-based optimizations.

The evolution of semiconductor process technology has facilitated using hardware as an accelerator and optimizing software. This has culminated in a prevalent trend among recent state-of-the-art studies to integrate hardware and software. In alignment with these developments, a DPU-based hardware accelerator has been incorporated into our research, alongside software optimization utilizing high-level languages, namely C/C++ and Python.

The contributions of this study to the existing scholarly discourse are elaborated upon in Section I-C. Following this overview, the particulars will be explored in-depth within this section. The sequence of the discussion begins with the system's infrastructure, progresses to the hardware design, and finally delves into the software design and its associated optimization methodologies.

B. MULTI-TASK LEARNING NETWORK

In the current landscape, the majority of networks focus on resolving a singular, specific task. However, in practical applications, a higher degree of efficiency is typically achieved by unifying several individual algorithms into a single comprehensive learning framework. This unification is made possible by multi-task learning networks, which amalgamate multiple tasks into a single cohesive unit. The efficiency of this method hinges on leveraging the relationships between these distinct tasks and optimizing them for real-time applications.

Multi-task learning networks combine various functions into a suitable task, exploiting the interrelationship between different tasks. This study combines four different learning algorithms for ADAS and presents it as a single learning network. Integrating algorithms that perform multiple individual tasks into a single unified learning framework is more efficient in real-time applications. As a result, networks generalize a more accurate representation of functions by sharing features between each task, thus improving learning efficiency and increasing prediction accuracy. Additionally, multi-tasking learning can reduce overall network size

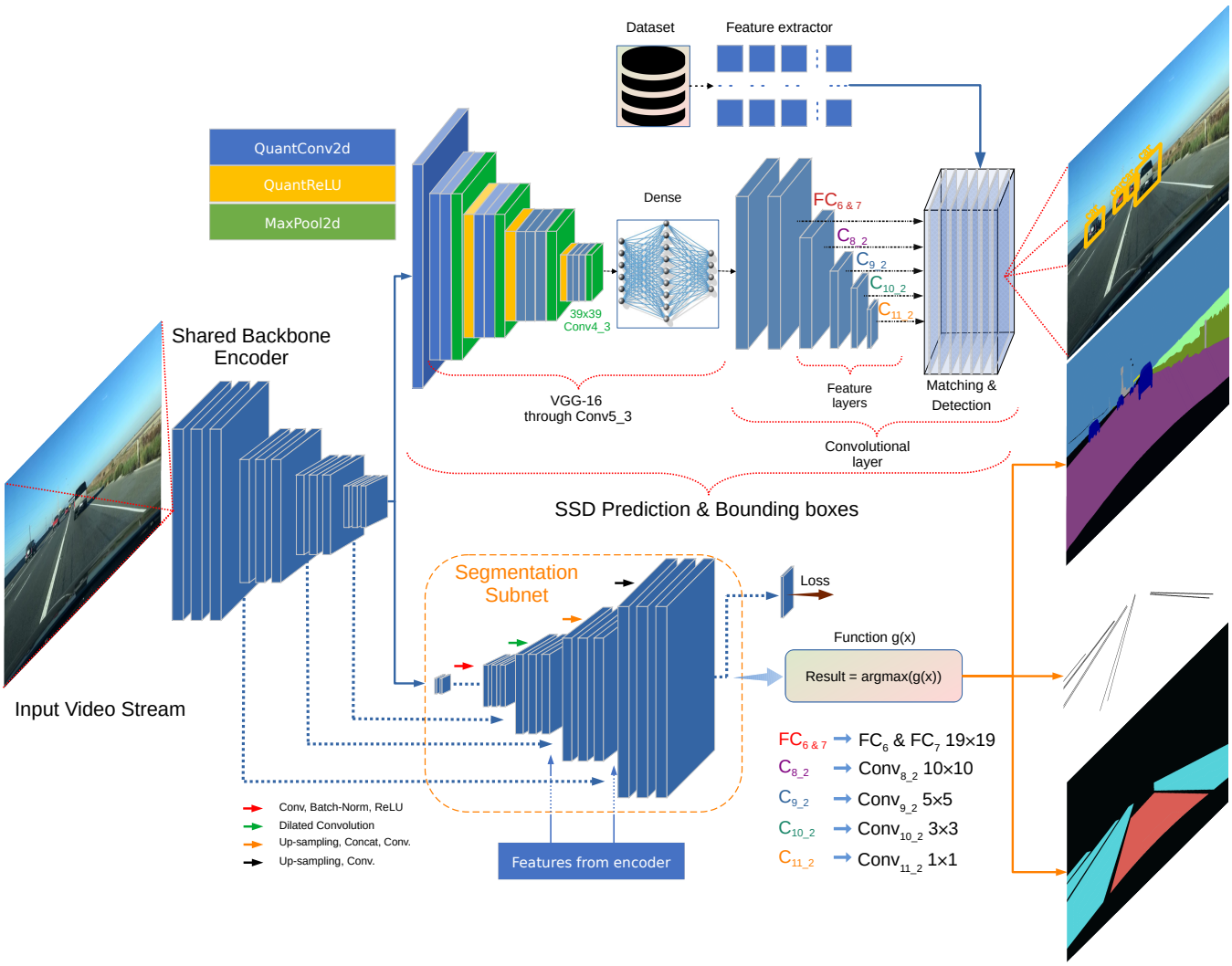


FIGURE 1. General overview of the ADAS multi-task learning.

(memory footprint) and computational complexity by sharing backbone layers. This provides convenience for resource-constrained platforms and is beneficial for quick inference needs.

Our proposed ADAS multi-task learning consists of a shared backbone encoder, a segmentation subnet, and a detection subnet, as in Fig. 1. The subnet models have been discussed in detail to gain a better understanding.

1) Shared backbone encoder

Various computer vision tasks, including object detection, are addressed using complex CNN architectures. The construction of object detection or segmentation architectures is based on CNN models initially trained for image classification, thanks to the principle of transfer learning. In this context, CNN serves as the feature extractor and forms the backbone of the object detection model. This backbone processes input data, focusing on specific features to extract detailed conceptual elements.

The study's model is rooted in the ResNet18 architecture rather than using deeper structures such as DenseNet, ResNet101, or GoogleNet. These have numerous hyperparameters and high computational densities. This choice enables the operation of the model on open-source and resource-limited embedded devices. The model is constructed through transfer learning, with initial training targeting object detection, segmentation, and classification. Performance optimization of the model is achieved by employing a shared backbone. This approach facilitates efficient multi-task learning for ADAS and reduces the computational load, positioning it as an effective solution for real-world applications. The strategic design of the model was undertaken with these considerations in mind.

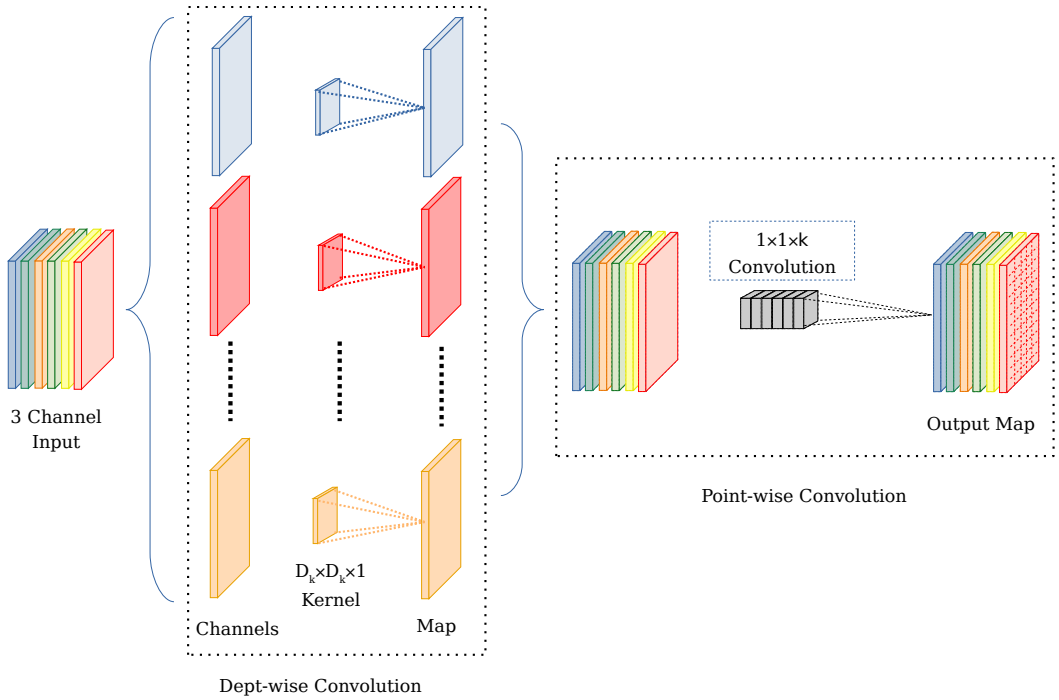


FIGURE 2. Separable dept-wise convolution architecture.

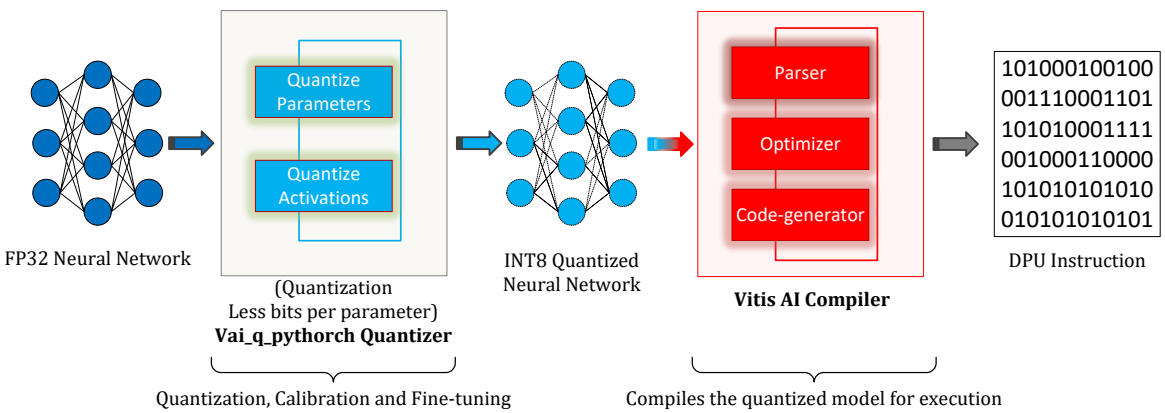


FIGURE 3. An overview of quantization and compilation of FP32 bits NN model.

2) Detection subnet

Our research adopted a single-stage SSD [5] as a detection decoder to fulfil real-time application needs and enable quick inference. The SSD employs a multi-scale feature map for swift object detection. However, as the CNN structure progressively downsizes the spatial dimensions, it also lowers the resolution of the feature map. Hence, the SSD uses lower-resolution layers for detecting and positioning larger-scale objects. As such, our research opted for a 4x4 feature map for identifying large objects. After VGG16, SSD introduces six extra convolution layers, with five dedicated to object detection. This layer configuration prompted us to generate six predictions on three rather than the conventional four, leading to approximately 8732 predictions across these layers. This strategic choice contributed to our model's efficiency and robustness, enhancing its ability to make precise object detection under various conditions.

3) Segmentation subnet

As illustrated in Fig. 1, the segmentation subnet's architectural design incorporates several learnable up-sampling layers. This subnet comprises a 3x3 convolution layer, layers of batch normalization, and multiple layers with activation functions. The output tensor size of one convolution block

layer and the input tensor size of another layer remain consistent, with only up-sampling altering the tensor size.
A convolution block is initially applied at the subnet's bottom, facilitating extracting of meaningful semantic features. Instead of using pooling to extract low-resolution features, three convolution blocks incorporating a dilated layer are employed. Given that the pooling process can result in a loss of detail, applying dilated convolution provides more relevant results for extracting deep features than standard convolution and subsequent pooling processes. After the deep feature extraction, up-sampling is performed to restore the spatial resolution. In the processing methodology, hints about objects provided by the encoder properties are integrated into the decoder side to delineate the boundaries with more precision. Feature summation is used in place of element accumulation to enhance inference accuracy.

4) Software optimization

FPGAs and SoCs implement domain-specific architectures to optimize CNN in applications, including inference rate, latency, and hardware utilization. Our ADAS multi-task learning model runs on ZYNQ Ultrascale+ MPSoC architecture as a co-design where DPU and ARM are used together. The DPU accelerates the computing workloads of DL inference algorithms commonly used in various computer vision applications. Therefore, the DPU performs the CNN computations here, while the ARM performs the pre-and post-processing of the input signal.
Furthermore, depth-wise separable convolution (DWSC) is adapted to the on-chip pipeline method to process efficiently in parallel, thereby reducing off-chip memory access. Thus, the DPU core significantly improves run-time scheduling efficiency during the computation of layers. DWSC decreases computational intensity [39] approximated to normal convolution operations. DWSC has the fundamental principle of separating two procedures dept-wise convolution (dwc) and point-wise convolution (pwc). Pwc has a 1x1 standard structure that follows deep convolution (see Fig. 2). It collects feature information from different channels in the exact spatial location, thus reducing the computational cost and memory footprint of separable convolutional networks.
In an effort to optimize memory utilization on the ARM (Quad-Core Cortex A-53) processing engines (PEs), careful data allocation was enacted to prevent unnecessary rereading. This facilitated the creation of an adaptable infrastructure for diverse neural network models. Similarly, multithreading and a pipeline architecture were implemented to leverage the DPU and Quad-Core PEs' full potential on the Kria KV260 development board. As a result, high efficiency was attained by mitigating delays during data communication between the PL and the PS. The processing duration of the PL, observable during data transactions via AXI-DMA, underscores this improvement.
A significant software-side enhancement was the quantization of the model, as depicted in Fig. 3. This modification resulted in a model with a memory footprint well-suited to resource-constrained devices, thereby considerably reducing computational density. Although model quantization may slightly impair inference accuracy, the benefits are sufficiently significant to overlook this minor setback. A detailed account of the improvements and inference results obtained are comprehensively discussed in Section III.

C. HARDWARE (OVERLAY) DESIGN

The ZYNQ architecture comprises a customizable MPSoC incorporating a quad-core ARM Cortex-A53, dual-core ARM Cortex RF53, ARM Mali 400MP and a conventional PL integrated circuit (IC). In addition, MPSoC is equipped with fast and efficient connections and supports the AXI standard. The system's design phase requires tasks to be allocated between the processor and the FPGA sections, known as PS and PL, based on the system requirements. This allocation is a critical step as the overall speed and functionality of the program depend on how tasks are distributed between PS and PL sections. The entire system's performance is influenced by the tasks assigned to each team. Our research proposal allocated the high-speed and computationally intensive parts to PL while assigning the remaining roles to PS.
Identifying the functional blocks in the hardware design, we integrated them as IPs to establish the necessary AXI-DMA interfaces between PL and PS, as depicted in Fig. 4. Digital hardware development was carried out using the Vivado 2022.1 integrated development environment, while high-level synthesis integrated system design was conducted in PL. The hardware accelerator design was prepared in the PL environment and comprised the overlay comprising IP blocks.
The ARM is utilized for pre-processing and post-processing tasks in this study, exploiting the capabilities of high-level languages such as Python and C++ and the OpenCV library. As part of the methodology, the decode thread is programmed to carry out resize functions on images of 1920x1080 and 320x512 resolutions. During the ML inference phase, scale and mean value subtraction operations are performed on the ARM, leveraging the capabilities of the Vitis-AI library. The remainder of the task is executed within the PL, specifically in the DPU. Due to the convolution process's intricate nature and heavy processing demands, the high-performance computational abilities of the DPU are employed to carry out this process successfully.
The execution of tasks in this study specifically involved the use of the AXI4 stream interface, a decision influenced by the superior functionalities of the Direct Memory Access (DMA) feature, which facilitated a notably rapid transfer of image pixel values. Strategic adjustments to the master port were carried out to facilitate seamless interaction between the PS and the PL. These modifications included setting the bit width for the AXI HPM0 FPD at 32, the AXI HPM1 FPD at 128, and the AXI HPM0 LPD at 32. Further configurations were made to the data width of the slave interface, with the AXI HPC0 and HPC1 explicitly set at 128 and the AXI LPD at 32. These configurations proved essential in

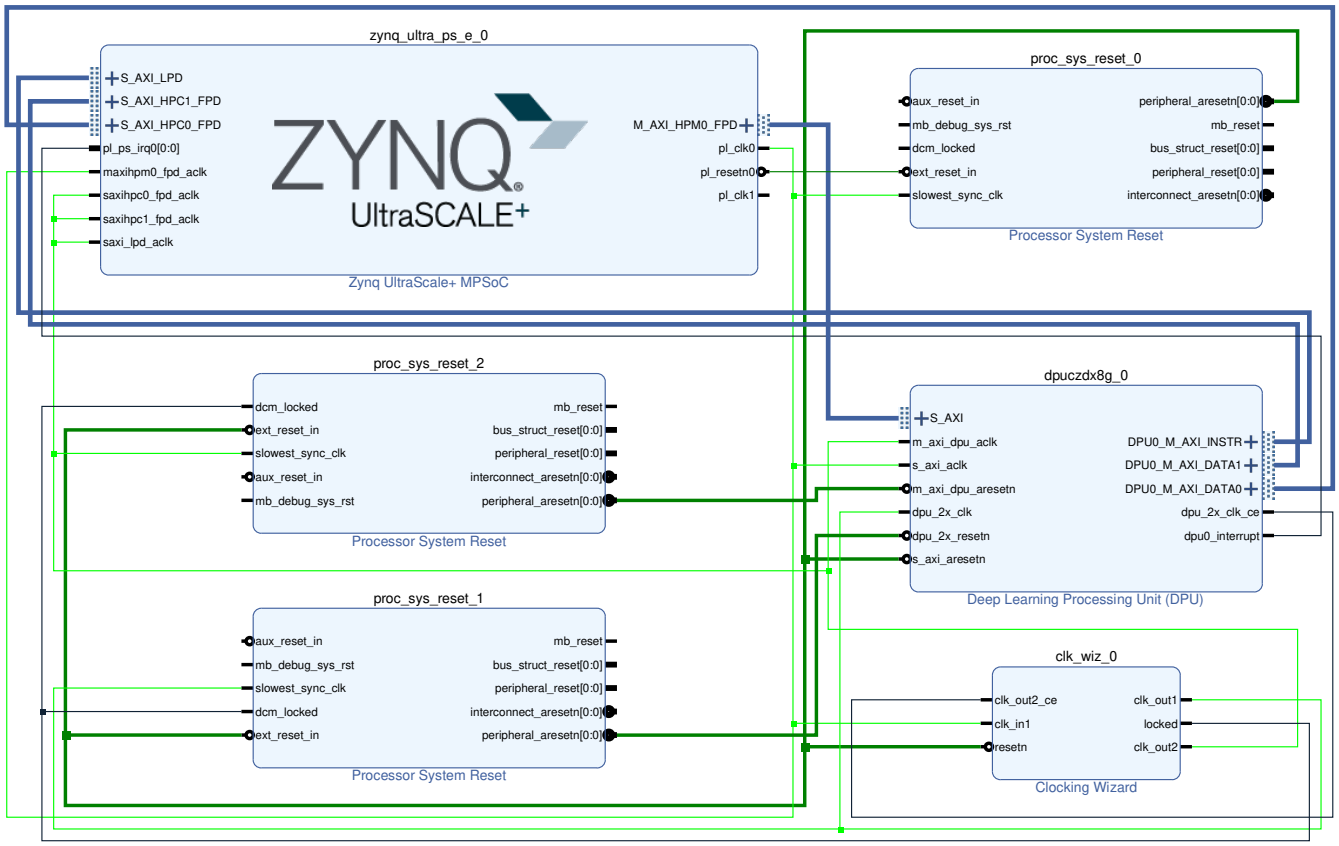


FIGURE 4. Overlay design of ADAS multi-task learning hardware.

maintaining the AXI4 protocol for the DPU architecture, a B4096 model. Such adaptations, aimed at enhancing the system's design, improved the interaction between hardware and software components while optimizing task execution within the system.

1) Hardware optimization

DL models exist in various types, and within resource-constrained environments, it is unlikely to create a single hardware parameter that caters to all models. Typically, designs with fewer hardware resources lead to improved FPGA timing, higher clock frequency, increased throughput, and reduced power consumption. For applications similar to ADAS, the presented design provides a viable compromise with the opportunity for customization through fine-tuned parameters. The DPU retrieves instructions from off-chip memory to guide the computing engine's procedure, with the Vitis-AI compiler generating these instructions, which include layer-fusion and optimizations.

This setup uses on-chip memory for input activation, feature maps, and output metadata buffering to maximize efficiency. Further, software optimization is employed to reuse data as much as possible, minimizing external memory bandwidth requirements. The DPU architecture's computing engine leverages a deep pipeline, and the PEs fully utilize fine-grained building blocks such as multipliers, adders, and accumulators. This methodology maximizes the benefits of the hardware accelerator structure.

It is possible to configure the convolution architectures available within the DPU IP architecture to align with the parallelism of the convolution unit. Architectures vary in PL resource consumption. Larger architectures consume more resources and thus show higher performance. On the other hand, we prefer smaller architectures for our resource-constrained device. When we use minor architecture, we experience a decrease in our device performance. To avoid dropping the performance, we used a Double Data Rate (DDR) approach to improve the performance we get with the DPU. In this formatting, we have specified 1x clock input for general logic and 2x for Digital Signal Processing (DSP) slices.

The cascade length's usage presents a crucial aspect of leveraging DSP. A trade-off between resource usage and timing performance always exists when determining the DSP cascade's size. For instance, deploying a more significant cascade length reduces resource consumption but delivers subpar timing performance. Conversely, opting for a shorter cascade length reduces resource usage and significantly improves timing performance. Hence, in smaller devices with limited logic resources, it is advisable to employ more exten-

sive cascade lengths.

In light of resource usage and timing performance, the study deduced that the DSP's ideal maximum cascade length is four. The DPU IP core's use of DSP elements forms the basis for whether DSP usage is high or low. In instances where DSP element usage is low, multiplication only is performed, whereas high DSP element usage involves both multiplication and accumulation. By setting DSP usage as high in the hardware design, a reduction in the computational density of the quantized DWSC model led to increased timing performance. Moreover, enabling UltraRAM memory in the hardware design allowed for the use of the more significant DPU architecture, given the device's lack of sufficient BRAM, effectively reducing the resource constraint of the development board. Fine-tuning performed on DSP, BRAM, and UltraRAM culminated in a more optimized PL process, yielding timing requirement values at an optimal level. As a consequence, the timing summary revealed values of 1.135ns for $Worst\ Negative\ Slack$, 0.001ns for $Worst\ Hold\ Slack$, 2.000ns for $Worst\ Pulse\ Width\ Slack$ and finally, 0.009ns for $Total\ Negative\ Slack$.

D. QUANTIZATION AWARE TRAINING (QAT)

Designers have been tailoring more comprehensive architectures to enhance the performance of CNN models. Such adaptations, including broader and more profound CNN architecture, have successfully reduced the classification error rate for specific problems. Authors of a particular study [40], utilizing various CNN models, exemplified the relationship between computational density and memory requirements in ImageNet classification. They observed a decrease in the ImageNet classification error rate from 17% to 2.9%. Consequently, expanding the network model incurs an increase in computational complexity. This escalation, in turn, leads to a considerable surge in memory requirements. Additionally, bandwidth concerns arise due to the millions of parameters found in CNN models.

Techniques such as model pruning, weight quantization, and activation function quantization [19], [41] aid in reducing computational complexity. Misuse of the quantization method can diminish inference accuracy, while the pruning method can prevent network over-fitting during training. Aiming for a goal-oriented model, the designer needs to balance these trade-offs. A potential pitfall of quantizing CNN model weights and activation functions is data loss, attributed to the inability to restore the floating point after quantization and de-quantization fully. To articulate this issue in mathematical terms;

$$x = f_d(f_q(x, s_x, z_x), s_x, z_x) + \Delta_x \tag{1}$$

where;

f_d and f_q are de-quantization and quantization functions, respectively. Δ_x is an undetermined small value. Suppose $\Delta_x = 0$, the quantized integer models' inference accuracies are the same as those of the floating point models. Unfortunately, this is not the case. The model performs well after training when the model parameters are in FP32 (32 bits floating-point arithmetic). However, setting the precision to int-8 (8 bits integer) or lower can lead to standard inference even if the network is well-trained. In contrast, the quantized network has a much lower memory requirement than the floating point counterpart, resulting in less energy consumption by the system. As a result, the quantized model is more suitable for battery-powered embedded devices.

This study delves into the implementation of real-time ADAS for an FPGA-based MPSoC hardware accelerator by quantizing the ResNet18 model with SSD assets. The weights and activation functions of the model were quantified as int-8 bit low precision integers and a performance comparison of the network was carried out. The PyTorch framework was utilized to construct the model, and the $vai\_q\_pytorch$ library was used to quantize the weights and activation functions. It is noteworthy that $vai\_q\_pytorch$ is a Vitis AI quantizer-supported library that operates on the PyTorch framework.

The Vitis AI [21] quantizer takes in the floating point model, conducts pre-processing, and then quantizes the weights and activation/biases at the specified bit-width. The pre-processing performed by Vitis AI folds the batch normalization and eliminates nodes from the model that are not necessary for inference. Thanks to batch normalization, simultaneous learning is possible across layers in the network. Without batch normalization, the use of a high learning rate could lead to the issue of disappearing gradients. However, with batch norms, a higher learning rate can be used since alterations in one layer do not impact the others.

Only the initial value is set as QAT is employed since the learning rate value will undergo automatic updates during training. The authors' extensive investigation and explanation of QAT are presented in their work [43], examining its various components and mechanisms. This article is recommended for those desiring a more complete and in-depth understanding of QAT. The detailed information in this work can provide further illumination and enrichment to the reader's comprehension of this specific area of study.

The QAT technique in neural networks strives to minimize the effect of data loss during training, with the inference accuracy of the model experiencing only minimal impact. Given that the weight and activation tensors change during neural network training, a quantization and de-quantization layer can be added for each varying tensor in QAT. Differing from (1), (2) and (3) can be defined in the following manner;

$$\hat{x} = f_d(f_q(x, s_x, z_x), s_x, z_x) \tag{2}$$

$$\hat{x} = s_x(clip(round(\frac{1}{s_x}x + z_x), \alpha_q, \beta_q) - z_x) \tag{3}$$

Data types for quantized tensors are still floating-point tensors. Therefore, we need to train as if there were no quantization layers. In addition, the main problem with QAT is that such quantization layers cannot be differentiated [42]. On the other hand, the straight-through estimation (STE) [44]

derivative strategy excels when used for QAT. The identity function in the clipping range $[\alpha, \beta]$ and the constant function outside of the clipping range $[\alpha, \beta]$ are how STE handles the quantization and de-quantization functions. Thus, the resulting derivatives are 1 if $[\alpha, \beta]$ is in the clipping range and 0 if outside of the field. We can define symmetric quantization mathematically as in 4.

$$\frac{\partial \hat{x}}{\partial x} = \begin{cases} 1 & if\ \alpha \leq x \leq \beta \\ 0 & else \end{cases} \tag{4}$$

Scaling factors can be discovered during QAT thanks to STE. For instance, the Learned Step-Size Quantization (LSQ) [45] is obtained from the scaling elements' gradient quantization function. Starting from 1, we can get 5 to 8 as follows;

$$\frac{\partial \hat{x}}{\partial s_x} = \frac{\partial s_x}{\partial s_x}\left(clip(round\left(\frac{1}{s_x}x, \alpha_q, \beta_q\right)\right) + \tag{5}$$
$$s_x \frac{\partial\left(clip\left(round\left(\frac{1}{s_x}x\right), \alpha_q, \beta_q\right)\right)}{\partial s_x}$$
$$= clip\left(round\left(\frac{1}{s_x}x\right), \alpha_q, \beta_q\right) + \tag{6}$$
$$s_x \frac{\partial\left(clip\left(round\left(\frac{1}{s_x}x\right), \alpha_q, \beta_q\right)\right)}{\partial s_x}$$

If we define the numerator part of (5) as any variable (θ) in order not to rewrite it at length; $\theta = clip\left(round\left(\frac{1}{s_x}x\right), \alpha_q, \beta_q\right)$

$$\cong \begin{cases} \theta + s_x \frac{\partial\left(\frac{1}{s_x}\right)x}{\partial s_x} & if\ \alpha \leq x \leq \beta \\ a_q + s_x \frac{\partial(b_q)}{\partial s_x} & if\ x < \alpha \\ b_q + s_x \frac{\partial(b_q)}{\partial s_x} & if x > \beta \end{cases} \tag{7}$$

$$= \begin{cases} round\left(\frac{x}{s_x}\right) - \frac{x}{s_x} & if\ \alpha \leq x \leq \beta \\ a_q & if\ x < \alpha \\ b_q & if\ x > \beta \end{cases} \tag{8}$$

Here, it is possible to learn or adaptively select different bit widths for each layer in a model or a uniform bit width for the entire model in this case. The $vai\_q\_pytorch$ library may quantize the activation functions of the model according to the following equations.

$$QuantReLU_{(x,z_x,y_x,k)} = \begin{cases} z_y & if\ x < z_x \\ z_y + k(x - z_x) & if\ x \geq z_x \end{cases} \tag{9}$$

When $z_x = 0$, $z_y = 0$ and $k = 1$, the generally utilised ReLU in DL models is a particular case of the above description.

$$ReLU_{(x,0,0,1)} = \begin{cases} 0 & if\ x < z_x \\ 1 & if\ x \geq z_x \end{cases} \tag{10}$$

Here, we have given the Mathematically analysis steps of the QuantReLU function.

$$y = ReLU(x, 0, 0, 1) \tag{11}$$
$$= \begin{cases} 0 & if\ x < z_x \\ 1 & if\ x \geq z_x \end{cases} \tag{12}$$
$$= s_y(y_q - z_y) \tag{13}$$
$$= ReLU(s_x(x_q - z_x), 0, 0, 1)$$
$$= \begin{cases} 0 & if\ s_x(x_q - z_x) < 0 \\ (s_x(x_q - z_x) & if\ s_x(x_q - z_x) \geq 0 \end{cases}$$
$$= \begin{cases} 0 & if\ x_q < z_x \\ s_x(x_q - z_x) & if\ x_q \geq z_x \end{cases} \tag{14}$$

Consequently;

$$y_q = \begin{cases} z_y & if\ (x_q < z_x \\ z_y + \frac{s_x}{s_y}(x_q - z_x) & if\ x_q \geq z_x \end{cases} \tag{15}$$
$$= ReLU(x_q, z_x, z_y, \frac{s_x}{s_y})$$

Hereby, to achieve the QuantReLU corresponding to the floating-point $y_q = ReLU(x, 0, 0, 1)$, we require to serve;

$$y_q = ReLU(x_q, z_x, z_y, \frac{s_x}{s_y}) \tag{16}$$

Where; z_x and z_y are zero points, s is a positive floating-point scale element and x_q is quantized matrices,

E. EXPERIMENTAL SETUP

This section details the experimental setup required for the real-time execution of ADAS multi-task learning. The discussion initiates with an explanation of the fundamental operation of the model and the significance of pipeline design. Following this, insights about the datasets used for this study will be shared. The next segment will dive into the process of model training. Concluding the section, an overview of the hardware-software co-design involved in this study will provide a comprehensive understanding of the overall process.

1) System setup

Our research leverages the capabilities of the AMD Xilinx Zynq UltraScale+ MPSoCs, uniting an FPGA with PL and an ARM processor inside a PS into a cohesive entity. The chosen experimental setup utilizes the Zynq UltraScale+ MPSoC Kria KV260 Vision AI development board from AMD Xilinx, offering 4GB DDR memory due to its compatibility

TABLE 2. Use of datasets

Datasets	Train	Validation	Test
BDD100K	70000	10000	20000
KITTI	7480	-	7517
CityScapes	2975	500	1525

and efficiency in executing the investigated algorithm.
The focus is primarily on the ARM Cortex A53 (PS) and PL. The PS coordinates multiple operations, encompassing monitor connections, pre-processing and post-processing task management, the interface functions oversight, USB interface regulation, and operating system activity control. Simultaneously, PL develops optimized on-chip and off-chip memory access techniques, formulates pipeline strategies, and supports hardware acceleration functions.
As depicted in Figure 5, multiple threads are established as pipelines and operated in parallel to maximize efficiency. This pipeline design strategy yields considerable benefits, contributing to a roughly 50% throughput increase, and diminishing design complexity and resource usage, as illustrated in Fig. 6. With each implemented FPGA kernel embodying a single thread, the inherent parallelism within this thread can be fully exploited.

2) Datasets

We are concentrating our research on improving autonomous driving in driver-operated and driverless vehicles by combining object detection and segmentation in a multi-task learning approach. We trained our model on three publicly available datasets: BDD100K, KITTI, and CityScapes. We mostly used the BDD100K dataset, known for its various autonomous driving scenes, and KITTI, offering object detection in three separate classes for both object detection and segmentation tasks. For semantic segmentation across 19 categories, we turned to the CityScapes dataset. Table 2 outlines the dataset distribution we used for training and inference.
We merged datasets into common categories to further enhance inference accuracy and efficiency. In particular, we merged the CityScapes and BDD100K datasets for segmentation tasks, while the BDD100K dataset was utilized for object detection. The data was portioned for model training, testing, and validation, resulting in highly promising outcomes for multi-task learning. The resulting data subsets have demonstrated highly favourable outcomes for multi-task learning. Evaluating the performance of our model, we employed standard mAP (17) and mIoU (18) criteria. The mAP was used to measure object recognition, while the mIoU was used to evaluate segmentation. The results suggest that consolidating datasets into shared categories significantly improved the inference's accuracy and efficiency in multi-task learning.

$$mAP = \frac{1}{n}\sum_{k=1}^{k=n} AP_k \tag{17}$$

Here, AP_k is the AP of class k, n is the number of classes,

$$mIoU = \frac{\left(\frac{1}{n_{cl}}\right)\sum_i n_{ii}}{\left(t_i + \sum_j n_{ji} - n_{ii}\right)} \tag{18}$$

Where n_{cl} represents the number of classes, t_i is the total number of pixel in class i, n_{ii} represents true positives, n_{ji} false negatives.

3) Model training

The construction of the sophisticated multi-task model necessitated a meticulous selection of loss functions. Furthermore, the extensive capacity of the model and the management of sizable datasets required the use of a high-performance GPU. The model was segmented into several subnets subjected to independent training to alleviate the computational burdens associated with end-to-end training.
The initialization of the shared backbone subnet was carried out during the pre-training phase, which capitalized on the extensive versatility of the ImageNet dataset, recognized for its proficiency in large-scale image classification assignments. This crucial step provided the backbone with meaningful representations for both tasks. The training protocol encompassed several stages. Initially, the semantic segmentation and backbone encoder subnets were rendered passive, followed by the training of the multi-object detection subnet. Afterwards, training was initiated for the semantic segmentation and backbone encoder subnets, achieved by temporarily turning off the object detection subnet. Each subnet was subjected to 100,000 training iterations to ensure thorough learning.
The commencement of the training phase was centred on setting up the general contextual information within the images, designated as weights. This stage was sustained until the loss function could indicate convergence towards a global minimum value. During the training phase, the quantization of weight and activation functions was expedited by applying QAT, aiming to curtail unnecessary quantization tasks. Additional steps included pre-processing measures, such as adapting the size of the input image, required explicitly for multi-object detection, to align directly with the mesh input size.
The accomplishment of the pre-training phase led to the fine-tuning of the entire multi-tasking model, with task-relevant labelled data incorporated into the process. The parameters of the shared backbone network were updated in parallel with those of the task-specific subnets. The optimization algorithms previously described were utilized to minimize the weighted sum of the loss functions. A series of tests on various hyperparameters, encompassing learning rate, weight decay, and batch size, were conducted to achieve the model's most effective configuration. A validation set was employed to monitor the model's performance.
The objective classification function for the object detection subnet was defined as $focal_loss2d$ and a soft L_1 loss was employed for bounding box regression, thereby tailoring

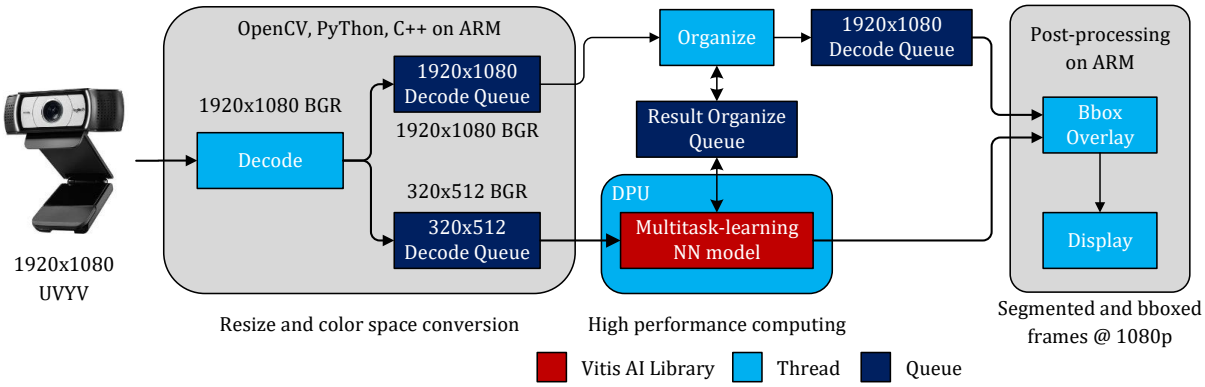


FIGURE 5. Fundamental execution of the ADAS multi-task learning

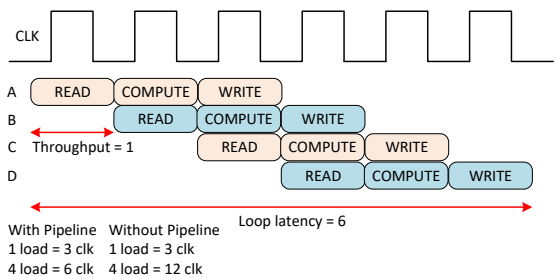


FIGURE 6. Kernel pipeline design

the model specifically for object detection tasks. Notably, $focal_loss2d$ effectively countered the influence of class imbalance during the training phase, whereas the soft L_1 loss function played a role in diminishing the effect of outliers in bounding box regression. Stochastic gradient descent (SGD) was subsequently applied to refine the model, setting a learning rate 1e-5 and a momentum value of 0.9.

In the SSD multi-box configuration, a batch-size ratio of 16 was set, and binary cross-entropy (BCELoss) was utilized as the loss function, guaranteeing a proficient training process. Encoder weights were initialized via a pre-trained ImageNet model for the segmentation subnet. The choice of $LovaszSoftmaxLoss$ [46] facilitated pixel-level classification and semantic segmentation tasks. The model's optimization relied on the SGD optimizer, assigning a learning rate of 1-e2. Notably, during training, the batch size for the segmentation subnet was designated as 2.

After training and fine-tuning procedures, the ADAS multi-task learning network was evaluated on the test set, employing relevant metrics for each task. Specifically, mAP was used for object detection, while IoU served the segmentation task. Such assessments aided in approximating the model's effectiveness and efficiency, subsequently offering critical insights for potential enhancements and refinements.

SGD and LovaszSoftmaxLoss can be specified mathematically as in 19, 20 and 21, respectively.

$$\theta = \theta - \eta.\nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{19}$$

$$\nabla = \frac{\partial}{\partial x}\hat{\imath} + \frac{\partial}{\partial y}\hat{\jmath} + \frac{\partial}{\partial k}\hat{z} \tag{20}$$

Here, ∇ is the gradient operator, η is the learning rate, $x^{(i)}$ is the training sample, and $y^{(i)}$ denotes the label, respectively.

$$loss(f) = \frac{1}{|C|}\sum_{c\in C}\overline{\Delta_{J_c}}(m(c)) \tag{21}$$

Where Δ_{J_c} is Jaccard loss extension, $m(c)$ is the vector of errors and C represents class.

You can find the features of the workstation we use for training in Table 3.

TABLE 3. Specifications of the computer used in evaluation

Parameters	Value	Unit
CPU Manufacturer	INTEL	-
CPU Variant	i7-7700HQ	-
CPU Clock Frequency	3.8	GHz
CPU Core Size	8	-
Cache Size	6	MB
RAM Size	32	GB
GPU Manufacturer	NVIDIA	-
GPU Chipset	GTX1080	-
GPU RAM Size	6	GB

4) Hardware - software co-design

This research emphasizes the creation of a versatile system input compatible with data from camera or sensor fusion. Such flexibility supports modifications in input modes via

adjustments to the kernel or root file format of the real-time operating system. This requirement is crucial given the pivotal role that machine vision systems serve in the automotive industry, particularly in object detection and semantic segmentation. Consequently, optimizing hardware and software co-designs is critical to meet the demands of throughput and power consumption.

The adopted approach in this study entails a meticulous evaluation of the algorithm, arranging its components based on the time consumption profiles for each process. The formation of this hierarchy was directed by the criteria established within this research. This process facilitated the identification of specific segments of the algorithm where single-instruction multiple-data (SIMD) operations were prevalent. An observed increase in power consumption in certain algorithmic sections was attributed to an escalation in algorithmic latency and the frequency of memory access. Given that, each memory access operation requires energy, an escalation in these operations' frequency directly affects the system's energy consumption. Consequently, power consumption elevates with an increase in the frequency of these accesses, as demonstrated in certain parts of our algorithm. This insight proved vital in identifying the algorithm's energy-intensive areas, optimising our overall system design for enhanced energy efficiency.

Modifications were made to several settings to circumvent potential resource constraints and amplify overall performance. ALU parallelism was set to 8, RAM usage to High, channel augmentation was enabled, and the DSP cascade length was extended to 4. Channel augmentation is optional to boost DPU efficiency, especially when the number of input channels is considerably less than the available channel parallelism. This scenario is frequently seen in numerous CNNs where the input channel of the first layer typically comprises three, failing to utilize the hardware channels optimally. However, even when the number of input channels surpasses channel parallelism, channel augmentation can be advantageous, albeit requiring more logic resources. Despite the related costs, this feature could enhance the efficiency of most CNNs. These optimizations' importance lies in achieving a highly efficient system design.

Table 4 outlines two unique DPU configurations and their respective utilization methods. Both configurations pose credible options for inference tasks, with this research choosing the configuration presented in case 1. Each variable emphasized in the table affects inference and memory consumption in distinctive ways. For instance, activating channel augmentation can improve the overall efficiency for numerous CNNs, although it may result in elevated LUTs consumption, thereby creating obstacles for devices with restricted memory. Moreover, it is critical to acknowledge that LUTs consumption may also vary among different DPU architectures, such as B1152, B3136, and B4096. Performance and memory prerequisites are further influenced by the types of ReLU used in convolution and ALU. A suggested setting for ALU parallelism is 4 for devices with memory restrictions. Nevertheless, this study chose a setting of 8 to cater to performance requirements. This modification, while leading to extra consumption of LUTs, FFs BlockRAMs, and DSPs resources, is often considered negligible when prioritizing performance.

TABLE 4. Performance comparison of two distinct DPU configurations for hardware-software co-design

DPU Architecture (B4096)	Case 1	Case 2
Channel Augmentation	Enable	Disable
ALU Parallel	8	4
DSP48 Maximal Cascade Length	4	4
RAM & DSP48 Usage	High	High
Convolution ReLU Type	ReLU+ReLU6	ReLU+LeakyReLU+ReLU6
ALU ReLU Type	ReLU+ReLU6	ReLU+ReLU6
Memory Consumption (Average)	Higher	Lower

III. RESULTS AND DISCUSSION

This research involves the Kria KV260 development board and the Logitech c930e camera, as illustrated in Fig. 7. The development board obtains a real-time video stream via USB for processing on ARM. DPU enables the PL environment to conduct complex computation and convolution operations, which are conveyed to the monitor through an HDMI connection. Performance evaluation was conducted via various ADAS use cases, including object detection, segmentation, line detection, and detection in drivable areas. The real-time results of these specified ADAS tasks are demonstrated in Fig. 8, showing proficient input video data processing.

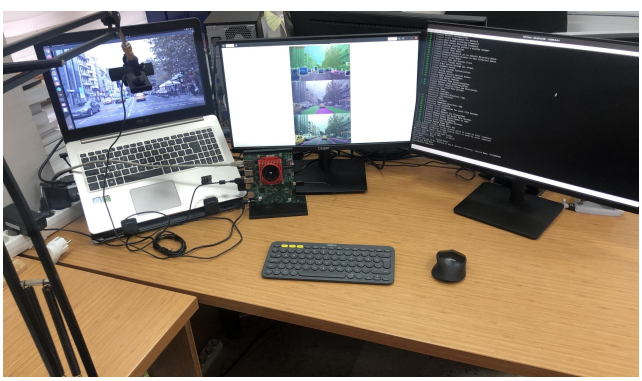
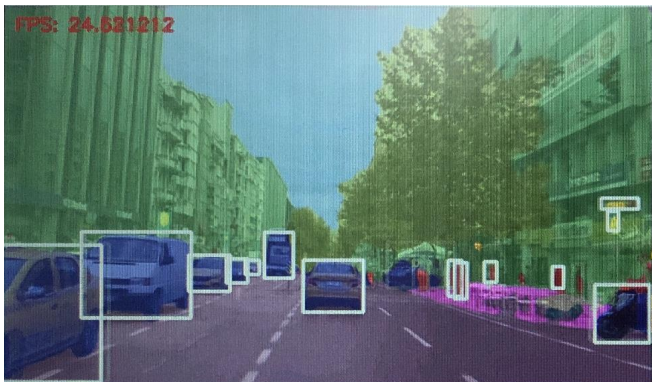


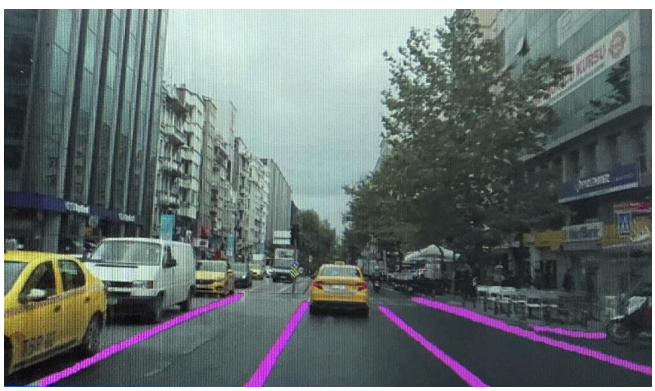
FIGURE 7. A Comprehensive examination of the real-time implementation of ADAS multi-task learning.

The platform's accuracy was gauged through the mAP value for object detection and the mIoU value for segmentation and line detection. In contrast, the platform's throughput was evaluated using the FPS value. Analogous investigations employing mAP and mIoU metrics have been presented for reader comprehension. A literature review identified a



(a) Multiple object detection

(b) Drivable area detection



(c) Line detection

FIGURE 8. Real-time implementation of ADAS multi-task learning.

demand for multi-task ADAS research deploying MPSoC FPGA.

The methodologies discussed presently might appear designated for diverse applications; however, their future incorporation in numerous sectors, notably those involving autonomous vehicle technology and ADAS, is virtually inevitable. Notably, within the MPSoC-FPGA environment, employing hardware-software co-design is poised to yield enhanced outcomes, especially when software-accelerated techniques are reinforced by hardware. Integrating hardware and software in such a harmonized approach can significantly leverage system performance, fostering advancements in the forthcoming era of autonomous and assisted driving technologies.

In Table 5, we have reviewed the literature encompassing our specified criteria. As discernible from the Table 5, the power consumption, the number of tasks, and the performance (GOPs) we recommend surpass those of the other studies. Notably, although operation at [28] appears optimal regarding power consumption, it only carries out the multi-object detection task. The study closest to ours was conducted at [48], where the researchers undertook multiple object detection and segmentation tasks. Our investigation indicates that their study's mAP and mIoU values are satisfactory. However, their power consumption exceeds ours, and they perform fewer tasks. Thus, as demonstrated in the table, our ADAS multi-task learning stands out in terms of both the number of functions and the evaluation outcomes.

The performance evaluation results for object detection using the specified dataset showed 51% mAP, indicating that the Kria KV260 Vision AI platform can accurately detect objects in real time. For segmentation, the platform achieved 56.62% mIoU, demonstrating its ability to segment objects accurately in complex scenarios. In line detection, the platform reached 43.86% IoU, indicating its ability to detect lines in the environment accurately. In seeing the derivable area, the platform also achieved 81.56% mIoU, demonstrating its ability to detect the derivable location accurately. Furthermore, the platform reached a throughput of 25.4 FPS at the optimized + pipeline design, indicating its real-time ability to process multiple ADAS use cases.

To assess the performance of ADAS multi-task learning on the Kria KV260 Vision AI Starter Kit Board, we conducted a comparative study between two precision, FP32 and QAT-INT8, as illustrated in Tab. 6. This comparison aimed to gauge how precision influences model performance concerning computational metrics (FLOPs) and task-specific outcomes. The FP32 model delivered 6.36 GFLOPs for the computational metrics, while the INT8 model attained nearly 25 GFLOPs. This boost in computational speed for the INT8 model is anticipated due to the reduced numerical precision, which leads to a lesser computational complexity and memory usage. Consequently, it enables more operations to be performed every second, resulting in a higher FLOPs value for the INT8 model. Alongside this, a minor trade-off between precision and task-specific performance outcomes is noticed, with the FP32 model showing slightly improved performance in segmentation and derivable area detection tasks, owing to its higher numerical precision.

Conversely, the INT8 model showed comparable or slightly superior performance in object and line detection tasks. Thus, the selection between FP32 and INT8 hinges on the application's specific requirements. FP32 may be preferable when the highest accuracy is a priority and computational resources are not a limiting factor. However, the INT8 model is a viable alternative for scenarios prioritizing computational ef-

TABLE 5. Performance comparison of the studies in terms of specified metrics

Ref.	Methods	Parameters	Power Consumption (w)	mAP (%)	mIoU / IoU(%)	GFLOPs
[13]	Multi-task learning	N	12.1	N	57.59	13.6
[28]	Multiple object det.	0.115M	3.118	N	67	N
[48]	Multi-task learning	N	14.3	62.8	57.6	9
[49]	Autonomous Driving	65.2M	9.8	N	N	N
[50]	Multi-task learning	65.2M	7.34	N	N	N
ADAS Multi-task Learning	Multi-task learning	11.4M	7.19	51	Segmentation: 56.62 Drivable: 81.56 Line: 43.86	25

TABLE 6. Performance comparison of the study in INT8 and FP32 model types

ADAS Multi-task Learning	Floating Point (FP32)	Quantization Aware Training (INT8)
Segmentation mIoU (%)	57.95	56.62
Multiple object detection mAP (%)	51.29	51
Drivable Area Detection mIoU (%)	82.63	81.56
Line Detection IoU (%)	43.65	43.86
GFLOPs	6.36	25

ficiency and speed, still delivering competitive performance. The optimal balance depends on the specific constraints and requirements of the application.

As can be seen from Table 6, we preferred QAT-INT8 and FP32 for comparison. There are methods for quantization, including post-training quantization (PTQ) (also called direct-quantization) and QAT. The memory footprint after quantization is similar in both methods. The main difference lies in the performance of the model after quantization. Our model that we trained with QAT was typically more resilient to the effects of quantization and outperformed a model we quantized with PTQ (i.e., it produced more accurate predictions) given the same amount of memory. QAT aims to reduce the accuracy disruption caused by the quantization process, but the process is more time-consuming and computationally expensive than PTQ.

We utilized multi-threading to improve the study's throughput and observed a significant performance improvement as in Table 7. We followed the performance variation of the DPU across varying thread sizes. It is well-known that the DPU we utilized has a maximum thread limit of 4. While employing a vast number of threads enhances the performance, it also significantly increases the DPU run-time value. Thus, we can elaborate on a trade-off between the thread size and run time. We also incorporated a DPU pipeline to enhance the performance of CNNs processing on the FPGA fabric, resulting in further progress in the platform's throughput. We use the Vitis AI analyzer tool to measure the platform's performance inference for all allocation threads and tasks, as shown in Fig. 9. In this context, every color denotes the speeds of read and write operations in megabytes per second (MB/s) for five distinct DDR ports. Additionally, the platform had a low memory footprint, indicating its efficiency in memory utilization. As a result, the Kria KV260 Vision AI platform delivers high accuracy and throughput while maintaining low power consumption and memory footprint. Furthermore, the platform's multi-task implementation and multi-class object detection capabilities allow it to process complex ADAS use cases. Our ADAS multi-task learning successfully integrated a complex and large model into a development board thanks to the hardware and software optimizations. Table 8 depicts the resource usage of similar analyses. We offered a glimmer of hope for resource-constrained devices by multi-tasking on a single development board. Our study incorporated the B4096 DPU architecture provided by AMD Xilinx, resulting in maximum efficiency at low frequencies and with limited resource usage. Our inference success and resource usage are commendable compared to the other two studies. Our hardware and software optimizations allowed for optimal utilization of the development board, creating sufficient resource space to include various ADAS tasks. Overall, our study is a cost-effective and efficient ADAS research solution that can be deployed in real-world applications with minimal modifications.

It is imperative to note that future investigations will not remain restricted to CNNs and DNNs. Despite the extensive applicability of these networks, there is an evolving trajectory towards more simplified structures in artificial intelligence, which could yield superior results compared to the highly effective structures comprising convolutional layers, such as DNNs and CNNs.

Indeed, relentless technological advancement has enabled current models to offer plausible solutions to contemporary problems. However, the escalating complexity of these problems necessitates formulating novel, diverse structures. This concept is well-exemplified in the research indicated in [51]. Contrary to the traditional CNN methodology, this study introduces the Physics Informed DL model. This paradigm

TABLE 7. Performance comparison of DPU with and without optimized

Thread size	Optimized + Pipeline			None-Optimized		
	DPU run-time (ms)	ARM (PEs) average run-time (ms)	FPS	DPU run-time (ms)	ARM (PEs) average run-time (ms)	FPS
t - 1	33.551	146.874	10.5	33.901	153.784	8.6
t - 2	34.095	156.342	19	34.194	165.864	15
t - 3	34.788	176.1	22	34.491	186.78	18.86
t - 4	34.839	220.5	25.4	34.766	209.687	21
Memory footprint	62.86 MB			69 MB		

TABLE 8. Resource consumption of hardware architectures used in similar studies

ADAS multi-task learning				[48]			[49]		
Resource	Available	Utilization	Utilization (%)	Available	Utilization	Utilization (%)	Available	Utilization	Utilization (%)
LUTs	117120	63445	54.17	70560	39772	56.37	274080	75000	27.3
LUTRAM	57600	7180	12.47	28800	3650	12.67	144000	N	N
FFs	234240	112272	47.93	141120	59045	41.84	548160	146000	26.7
BRAM	144	135	93.75	216	123	56.94	912	280	30.7
URAM	64	48	75	N			0	N	
DSP	1248	774	62.02	360	211	58.61	2520	N	
BUFG	352	6	1.7	196	3	1.53	NA	N	
Platform	XCK26			XCZU3EG			ZU9		
DPU Architecture	1 x B4096			1 x B1152F			2 x B4096		
Frequency (MHz)	300			525			600		

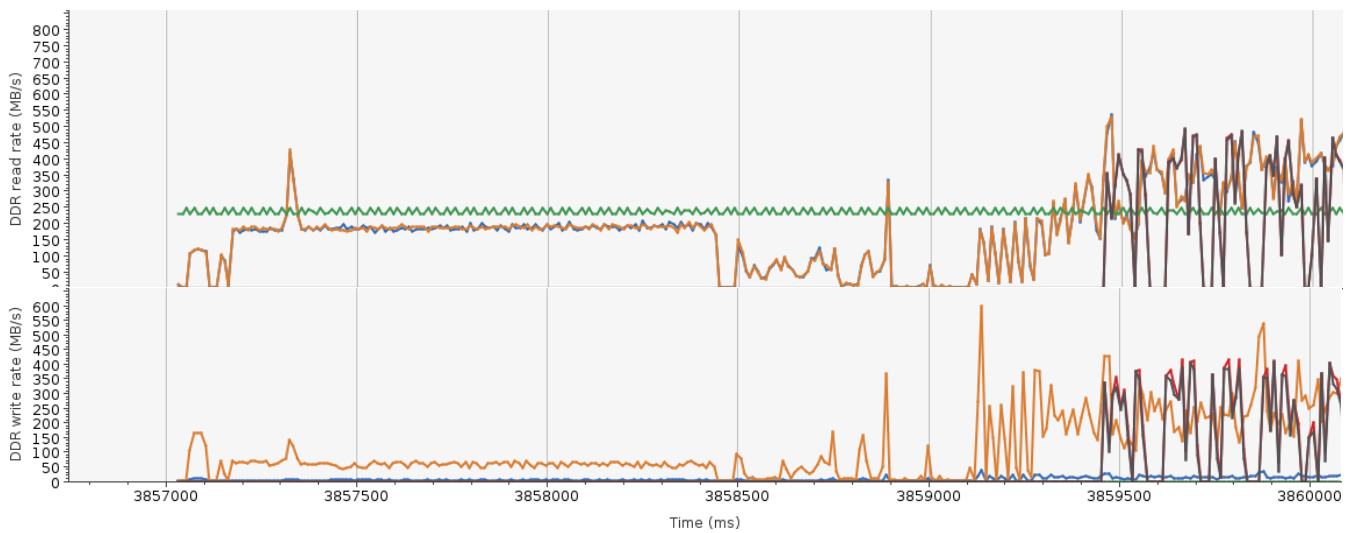


FIGURE 9. Performance evaluation of DDR ports read/write in Vitis AI analyzer

can be characterized as a physics-knowledge-informed deep learning framework, wherein physics-based domain knowledge is assimilated into the data-driven model as soft constraints. These constraints serve to guide and adjust the data-driven model.

In parallel, the research in [52] elucidates Extreme Learning Machines (ELMs) as an alternative to traditional deep learning methodologies, which typically encompass Deep Belief Networks and Constrained Boltzmann Machines. This approach streamlines the training process, a phase typically protracted by the intricate fine-tuning of numerous parameters and the complexity of the hierarchical structure. ELMs achieve this through a non-iterative, rapid training process facilitated by a random feature-matching mechanism.

IV. CONCLUSION AND FUTURE WORK

This study proposes a resourceful and efficient solution for the execution of multi-task ADAS on an MPSoC-FPGA,

focusing on detecting multiple objects, lane identification, drivable area detection, and semantic segmentation. Our strategy provides an effective and efficient development pathway for embedded systems while ensuring minimal power consumption. As part of our comprehensive methodology, we have incorporated a variety of adjustments encompassing both software and hardware enhancements. On the software aspect, we amalgamated several models and implemented a unified learning algorithm, leading to a consequent quantification of the model.

This software-level modification resulted in a notable reduction in memory consumption by approximately 9%. We exploited parallelization techniques using variable threads and pipeline architecture on the hardware aspect. The simultaneous software and hardware components design ensured that the algorithm performed with improved efficiency, compatibility, and speed. As a direct consequence of these comprehensive optimizations, the system exhibited increased accuracy, enhanced performance, and minimized energy consumption.

Notably, our research approach employed a single B4096 DPU, a measure that substantially reduced resource consumption compared to prior research endeavours. This strategy culminated in our system achieving an energy consumption rate of 7.19w, an FPS value of 25.4, a memory footprint of nearly 62.86MB, a multi-object detection rate of 51% mAP, a segmentation rate of 56.62% mIoU, a drivable area detection rate of 81.56% mIoU, and a line detection rate of 43.86% IoU. Given the data and results, ADAS multi-task learning offers an effective, efficient, sustainable, and precise system design for real-time ADAS applications. Furthermore, with its low power consumption, cost-effectiveness, and compact design, this system presents a compelling solution for real-world applications, as the experimental results demonstrate the proposed method's feasibility for conducting real-time processing in low-power embedded devices for on-road testing.

The main hurdle for real-time systems is to supply drivers with instant data based on object detection. Further, a hardware setup that ensures fast and accurate output is crucial. In response to these challenges, a state-of-the-art real-time deep learning configuration has been developed and assessed that synergizes with embedded systems and a computing environment, guaranteeing high detection accuracy. Owing to its remarkable accuracy and speed, this research acts as a beacon for academics, showcasing the effectiveness of real-time road object detection, segmentation, and line and drivable area identification on mobile platforms, all while consuming minimal power.

Addressing impending challenges, ADAS has become indispensable in the modern automotive industry, significantly enhancing driver safety and convenience. The successful deployment of ADAS necessitates the integration of an array of sensors, complex algorithms, and sophisticated computing resources. These elements must collaborate to interpret environmental data and deliver real-time decisions. A solution that has seen considerable attention in recent years involves the utilization of MPSoC FPGAs, owing to their capabilities in parallel computing and reconfigurable logic.

The critical elements of MPSoC FPGAs, namely parallel computing and pipeline architectures, have the potential to amplify the performance of ADAS applications significantly. For instance, the opportunity to create custom hardware accelerators like DPUs using programmable logic and processing engines can enable rapid and low-latency processing of image and video data. Further advancements in software frameworks and libraries, such as OpenCV, Python, and C++, have streamlined the deployment of intricate algorithms necessary for successful ADAS applications.

Multi-task learning presents another approach to enhance the efficacy of ADAS systems. This technique trains a single model to perform multiple tasks, including object and lane detection, allowing it to recognize and interpret varied environmental features simultaneously. Nevertheless, this strategy introduces its challenges, particularly in memory allocation and management of processing resources.

A potential resolution to these constraints of memory and resources is the application of quantized aware training. This approach facilitates the creation of compact and efficient models, ensuring minimal performance degradation. Nevertheless, optimizing these quantized models demands a delicate balance, considering the potential trade-offs between accuracy, performance, and memory consumption.

While substantial strides have been made in developing ADAS systems utilizing MPSoC FPGAs and other computing resources, several hurdles remain. A principal challenge lies in integrating ADAS with other advanced methods, like autonomous driving, which demands heightened reliability, safety, and security. Furthermore, ADAS systems operating in harsh environments, such as extreme temperatures or adverse weather conditions, require specialized hardware and software architectures. To conclude, MPSoC FPGAs, parallel computing architectures, and software frameworks like OpenCV and Python provide promising avenues for developing efficient and high-performance ADAS systems. However, it is imperative to address the significant challenges to ensure the widespread adoption and successful deployment of ADAS across diverse applications.

Our future research is directed towards investigating how innovative technologies, such as advanced LiDAR or radar systems, can be incorporated into the multi-task learning approach of ADAS. We also consider integrating refined computer vision and machine learning algorithms to enhance object detection and tracking accuracy. The thought of advancing the decision-making process in ADAS using Vehicle-to-Everything (V2X) data for coordinating activities across multiple vehicles, thereby augmenting road safety, is also contemplated. Furthermore, the advent of technologies like quantum computing and neuromorphic computing may revolutionize ADAS. These innovative concepts underpin our future research plans as we investigate these technologies' potential impacts and integrative possibilities in ADAS sys-

tems.

REFERENCES

[1] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 2014.

[2] C. Szegedy et al., "Going deeper with convolutions," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 2015, pp. 1-9, doi: 10.1109/CVPR.2015.7298594.

[3] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

[4] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137-1149, 1 June 2017, doi: 10.1109/TPAMI.2016.2577031.

[5] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016, October). Ssd: Single shot multibox detector. In European conference on computer vision (pp. 21-37). Springer, Cham.

[6] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

[7] H. Noh, S. Hong and B. Han, "Learning Deconvolution Network for Semantic Segmentation," 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 2015, pp. 1520-1528, doi: 10.1109/ICCV.2015.178.

[8] V. Badrinarayanan, A. Kendall and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 12, pp. 2481-2495, 1 Dec. 2017, doi: 10.1109/TPAMI.2016.2644615.

[9] D. Neven, B. D. Brabandere, S. Georgoulis, M. Proesmans and L. V. Gool, "Towards End-to-End Lane Detection: an Instance Segmentation Approach," 2018 IEEE Intelligent Vehicles Symposium (IV), Changshu, China, 2018, pp. 286-291, doi: 10.1109/IVS.2018.8500547.

[10] S. Lee et al., "VPGNet: Vanishing Point Guided Network for Lane and Road Marking Detection and Recognition," 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 2017, pp. 1965-1973, doi: 10.1109/ICCV.2017.215.

[11] Neven, D., De Brabandere, B., Georgoulis, S., Proesmans, M., & Van Gool, L. (2017). Fast scene understanding for autonomous driving. arXiv preprint arXiv:1708.02550.

[12] M. Teichmann, M. Weber, M. Zöllner, R. Cipolla and R. Urtasun, "MultiNet: Real-time Joint Semantic Reasoning for Autonomous Driving," 2018 IEEE Intelligent Vehicles Symposium (IV), Changshu, China, 2018, pp. 1013-1020, doi: 10.1109/IVS.2018.8500504.

[13] J. Peng et al., "Multi-task ADAS system on FPGA," 2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), Hsinchu, Taiwan, 2019, pp. 171-174, doi: 10.1109/AICAS.2019.8771615.

[14] Sanjay Basu, P. D. (2022, August 7). Deep learning part 3/4. Medium. Retrieved January 9, 2023, from https://medium.com/my-aiml/deep-learning-part-3-4-5c1392ecbc17

[15] Jawandhiya, P. (2018). Hardware design for machine learning. Int. J. Artif. Intell. Appl, 9(1), 63-84.

[16] J. Borrego-Carazo, D. Castells-Rufas, E. Biempica and J. Carrabina, "Resource-Constrained Machine Learning for ADAS: A Systematic Review," in IEEE Access, vol. 8, pp. 40573-40598, 2020, doi: 10.1109/ACCESS.2020.2976513.

[17] K. S. Zaman, M. B. I. Reaz, S. H. Md Ali, A. A. A. Bakar and M. E. H. Chowdhury, "Custom Hardware Architectures for Deep Learning on Portable Devices: A Review," in IEEE Transactions on Neural Networks and Learning Systems, vol. 33, no. 11, pp. 6068-6088, Nov. 2022, doi: 10.1109/TNNLS.2021.3082304.

[18] M. Lebedev and P. Belecky, "A Survey of Open-source Tools for FPGA-based Inference of Artificial Neural Networks," 2021 Ivannikov Memorial Workshop (IVMEM), Nizhny Novgorod, Russian Federation, 2021, pp. 50-56, doi: 10.1109/IVMEM53963.2021.00015.

[19] G. Tatar, S. Bayar and I. Cicek, "Hardware Acceleration of FIR Filter Implementation on ZYNQ SoC," 2022 IEEE 16th International Conference on Application of Information and Communication Technologies (AICT), Washington DC, DC, USA, 2022, pp. 1-6, doi: 10.1109/AICT55583.2022.10013522.

[20] G. Tatar, S. Bayar and I. Cicek, "Performance Evaluation of Low-Precision Quantized LeNet and ConvNet Neural Networks," 2022 International Conference on INnovations in Intelligent SysTems and Applications (INISTA), Biarritz, France, 2022, pp. 1-6, doi: 10.1109/INISTA55318.2022.9894261.

[21] Vitis AI. (2021). Xilinx. Retrieved November 7, 2022, from https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html

[22] Rani, M. R., Mustafar, M. Z. C., Ismail, N. H. F., Mansor, M. S. F., & Zainuddin, Z. (2021, March). Road peculiarities detection using deep learning for vehicle vision system. In IOP Conference Series: Materials Science and Engineering (Vol. 1068, No. 1, p. 012001). IOP Publishing.

[23] Almeida, T., Lourenço, B., & Santos, V. (2020). Road detection based on simultaneous deep learning approaches. Robotics and Autonomous Systems, 133, 103605.

[24] A. Hernández et al., "3D-DEEP: 3-Dimensional Deep-learning based on elevation patterns for road scene interpretation," 2020 IEEE Intelligent Vehicles Symposium (IV), Las Vegas, NV, USA, 2020, pp. 892-898, doi: 10.1109/IV47402.2020.9304601.

[25] Andrei, M. A., Boiangiu, C. A., Tarbă, N., & Voncilă, M. L. (2022). Robust lane detection and tracking algorithm for steering assist systems. Machines, 10(1), 10.

[26] Chen, Y., Xiang, Z., & Du, W. (2022). Improving lane detection with adaptive homography prediction. The Visual Computer, 1-15.

[27] Ghorbel, A., Ben Amor, N., & Abid, M. (2022). GPGPU-Based Parallel Computing of Viola and Jones Eyes Detection Algorithm to Drive an Intelligent Wheelchair. Journal of Signal Processing Systems, 94(12), 1365-1379.

[28] Machura, M., Danilowicz, M., & Kryjak, T. (2022). Embedded Object Detection with Custom LittleNet, FINN and Vitis AI DCNN Accelerators. Journal of Low Power Electronics and Applications, 12(2), 30.

[29] Sharma, N., & Garg, R. D. (2022). Cost reduction for advanced driver assistance systems through hardware downscaling and deep learning. Systems Engineering, 25(2), 133-143.

[30] E. Güney, C. Bayilmiş and B. Çakan, "An Implementation of Real-Time Traffic Signs and Road Objects Detection Based on Mobile GPU Platforms," in IEEE Access, vol. 10, pp. 86191-86203, 2022, doi: 10.1109/ACCESS.2022.3198954.

[31] H. -Y. Han, Y. -C. Chen, P. -Y. Hsiao and L. -C. Fu, "Using Channel-Wise Attention for Deep CNN Based Real-Time Semantic Segmentation With Class-Aware Edge Information," in IEEE Transactions on Intelligent Transportation Systems, vol. 22, no. 2, pp. 1041-1051, Feb. 2021, doi: 10.1109/TITS.2019.2962094.

[32] M. A. Farooq, P. Corcoran, C. Rotariu and W. Shariff, "Object Detection in Thermal Spectrum for Advanced Driver-Assistance Systems (ADAS)," in IEEE Access, vol. 9, pp. 156465-156481, 2021, doi: 10.1109/ACCESS.2021.3129150.

[33] J. Cho, Y. Kim, H. Jung, C. Oh, J. Youn and K. Sohn, "Multi-Task Self-Supervised Visual Representation Learning for Monocular Road Segmentation," 2018 IEEE International Conference on Multimedia and Expo (ICME), San Diego, CA, USA, 2018, pp. 1-6, doi: 10.1109/ICME.2018.8486472.

[34] S. Krishnan et al., "Automatic Domain-Specific SoC Design for Autonomous Unmanned Aerial Vehicles," 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO), Chicago, IL, USA, 2022, pp. 300-317, doi: 10.1109/MICRO56248.2022.00033.

[35] C. -Y. Lai, B. -X. Wu, V. M. Shivanna and J. -I. Guo, "MTSAN: Multi-Task Semantic Attention Network for ADAS Applications," in IEEE Access, vol. 9, pp. 50700-50714, 2021, doi: 10.1109/ACCESS.2021.3068991.

[36] Gaurav Nakhare. Hardware options for machine/deep learning. https://mse238blog.stanford.edu/2017/07/gnakhare/ hardware-options-for-machinedeep-learning/

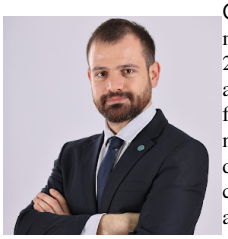
[37] P. Jawandhiya. Hardware design for machine learning. InternationalJournal of Artificial Intelligence & Applications, 9:63–84, 2018.

[38] Chen, R., Wu, T., Zheng, Y., & Ling, M. (2022). Mlof: Machine learning accelerators for the low-cost fpga platforms. Applied Sciences, 12(1), 89.

[39] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 1800-1807, doi: 10.1109/CVPR.2017.195.

[40] Wu, Chen, et al. "Low-precision Floating-point Arithmetic for High-performance FPGA-based CNN Acceleration." ACM Transactions on Reconfigurable Technology and Systems (TRETS) 15.1 (2021): 1-21.

[41] Véstias, Mário P., et al. "A fast and scalable architecture to run convolutional neural networks in low density FPGAs." Microprocessors and Microsystems 77 (2020): 103136.
[42] Lei Mao, "Quantization for Neural Networks," https://leimao.github.io/article/Neural-Networks-Quantization/ (accessed: Feb. 18, 2023).
[43] NOVAC, Pierre-Emmanuel, et al., "Quantization and Deployment of Deep Neural Networks on Microcontrollers" Sensors 21, 2021. no. 9: 2984. https://doi.org/10.3390/s21092984
[44] Bengio, Y., Léonard, N., & Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432.
[45] Esser, S. K., McKinstry, J. L., Bablani, D., Appuswamy, R., & Modha, D. S. (2019). Learned step size quantization. arXiv preprint arXiv:1902.08153.
[46] M. Berman, A. R. Triki and M. B. Blaschko, "The Lovasz-Softmax Loss: A Tractable Surrogate for the Optimization of the Intersection-Over-Union Measure in Neural Networks," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 2018, pp. 4413-4421, doi: 10.1109/CVPR.2018.00464.
[47] J. Wang and S. Gu, "FPGA Implementation of Object Detection Accelerator Based on Vitis-AI," 2021 11th International Conference on Information Science and Technology (ICIST), Chengdu, China, 2021, pp. 571-577, doi: 10.1109/ICIST52614.2021.9440554.
[48] S. Fang et al., "Real-Time Object Detection and Semantic Segmentation Hardware System with Deep Learning Networks," 2018 International Conference on Field-Programmable Technology (FPT), Naha, Japan, 2018, pp. 389-392, doi: 10.1109/FPT.2018.00081.
[49] A. Kojima and Y. Osawa, "Design and Implementation of Autonomous Driving Robot Car Using SoC FPGA," 2019 International Conference on Field-Programmable Technology (ICFPT), Tianjin, China, 2019, pp. 441-444, doi: 10.1109/ICFPT47387.2019.00088.
[50] Kalapothas, Stavros, Georgios Flamis, and Paris Kitsos. 2022. "Efficient Edge-AI Application Deployment for FPGAs" Information 13, no. 6: 279. https://doi.org/10.3390/info13060279.
[51] J. Zhang et al., "Physics-Informed Deep Learning for Musculoskeletal Modeling: Predicting Muscle Forces and Joint Kinematics From Surface EMG," in IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 31, pp. 484-493, 2023, doi: 10.1109/TNSRE.2022.3226860.
[52] J. Zhang et al., "Non-iterative and fast deep learning: Multilayer extreme learning machines," Journal of the Franklin Institute, vol. 357, pp. 8925-8955, 2020, doi: 10.1016/j.jfranklin.2020.04.033.



GUNER TATAR (M'21) was born in Kahramanmaras/Elbistan and completed high school in 2007. In 2009, He enrolled in Marmara University and spent a year studying English preparation before graduating from the Electronics and Communications department in 2014. That same year, He decided to pursue a master's degree and was accepted into Marmara University's Institute of Pure and Applied Science in EEE while simultaneously studying for the second bachelor's degree in EEE at Inonu University in Malatya. Following graduation, He worked for a year as a scholarship student in the development of biomedical imaging and diagnostic systems infrastructure, which was financially supported by the Ministry of Development in 2017. From October 2, He has been working as a Research Assistant in the Department of EEE at Fatih Sultan Mehmet Vakf University while also pursuing a PhD in EEE (English) at Marmara University. His main interests are including Reconfigurable Computing, Dynamic and Partial Reconfiguration of AMD Xilinx FPGA, Multiprocessors, Embedded Multicore Architecture, Deep Learning and Driver Assistant Systems.

SALIH BAYAR received his BS degree in electronics and communication engineering from Yıldız Technical University, Istanbul, Turkey, in 2003. He has received his MS degree in Electrical Engineering and Information Technology in specialization Systems Engineering from Karlsruhe Institute of Technology, Karlsruhe, Germany, in 2007. He was a research assistant in the Department of Computer Engineering at Bogazici University, Istanbul, Turkey, between 2007 and 2013. He received his PhD degree from the Department of Computer Engineering at Bogazici University. He worked as a Research and Development Engineer and Manager from 2013 to 2017 in a leading software company, Istanbul, Turkey. Since 2017 he has been an Assistant Professor in the Electrical and Electronics Department at Marmara University, Istanbul, Turkey. His main research interests are parallel computing, Machine Learning, Image Processing, FPGAs, multi- processor and embedded multi-core architectures.