Enabling the Proxy Computing Paradigm on DPU-based FPGA Acceleration

Gianluca Brilli
University of Modena and Reggio Emilia
Modena, Italy
gianluca.brilli@unimore.it

Alessandro Capotondi
University of Modena and Reggio Emilia
Modena, Italy
alessandro.capotondi@unimore.it

Paolo Burgio
University of Modena and Reggio Emilia
Modena, Italy
paolo.burgio@unimore.it

Andrea Marongiu
University of Modena and Reggio Emilia
Modena, Italy
andrea.marongiu@unimore.it

Abstract

FPGA-based heterogeneous systems are a popular choice for accelerating Deep Neural Networks (DNNs), but efficiently integrating and orchestrating HW and SW tasks remains challenging. FPGA overlay architectures have been proposed to simplify accelerator management, yet state-of-the-art solutions struggle with performance bottlenecks caused by frequent CPU-FPGA interactions. We introduce a novel overlay-based methodology enabling the *Proxy Computing* paradigm, leveraging a local orchestrator and shared memory to (i) reduce accelerator control overhead and (ii) minimize unnecessary data movements. As a case study, we integrate the AMD/Xilinx Deep Learning Processing Unit (DPU) with additional accelerators for unsupported layers. Experiments show that our approach significantly reduces memory transfers, achieving up to 4× speed up in the proposed case study.

CCS Concepts

• **Hardware → Reconfigurable logic and FPGAs**; • **Computer systems organization → Embedded hardware**; • **Computing methodologies** → *Artificial intelligence.*

Keywords

FPGA, Deep Neural Networks, Proxy Computing, Hardware Accelerators, DPU

ACM Reference Format:
Gianluca Brilli, Alessandro Capotondi, Paolo Burgio, and Andrea Marongiu. 2025. Enabling the Proxy Computing Paradigm on DPU-based FPGA Acceleration. In *22nd ACM International Conference on Computing Frontiers (CF '25), May 28–30, 2025, Cagliari, Italy*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3719276.3725192

1 Introduction

Deep Neural Networks (DNNs) are fundamental in Cyber-Physical Systems (CPS) design, excelling in tasks like perception for autonomous navigation [9]. These systems require low latency, high throughput, and energy efficiency for real-time control, sensor processing, and decision-making. FPGAs are well-suited for DNN acceleration due to their customizable hardware, optimized performance, and adaptability to evolving applications [14]. However, DNNs are typically part of larger applications involving multiple HW and SW tasks [11]. The adoption of FPGAs has been facilitated by heterogeneous systems integrating FPGA logic with multi-core CPUs and memory hierarchies, supporting standard OS and applications [14]. For acceleration logic design, established solutions rely on high-level flows such as High Level Synthesis (HLS), OpenCL, or pre-configured engines from major System on Chip (SoC) vendors. The AMD/Xilinx Deep Learning Processing Unit (DPU), for instance, is a commercial DNN accelerator deployable on FPGAs in a plug-and-play manner [8, 12, 13, 15]. Despite these advancements, integrating DNN engines with other accelerators and processing cores remains a challenge. FPGA *overlays* simplify accelerator-rich system design [1–3], but performance bottlenecks persist due to orchestration overhead and memory transfer inefficiencies. Prior works explored integration solutions, such as coupling the NVDLA engine with RISC-V cores [5, 6], or leveraging small RISC-V cores for co-processor control [1–3, 10]. Bernardi et al. [2] demonstrated that using a soft-core to manage a localization engine reduces memory transactions and end-to-end latency, but their results also highlight that the soft-core is not capable of sustaining the compute throughput required by an accelerator. For this reason, unlike our approach, we strictly employ the soft-core for orchestration.

We propose an *overlay*-based methodology enabling the *Proxy Computing* paradigm, which provides *local* control of accelerators within an FPGA *cluster*, including DMA engines managing data movement from DRAM to shared memory. This reduces CPU-dependent memory copies, improving computation locality. As a case study, we integrate the AMD/Xilinx DPU with additional accelerators for unsupported layers. The proposed approach encourages accelerating all compute-intensive components using established methodologies such as HLS while centralizing their control within the overlay. Our experiments demonstrate that *Proxy Computing* achieves up to 4× performance improvement in multi-accelerator scenarios compared to vendor-provided workflows.

2 The Proxy Computing Approach

FPGA-based heterogeneous systems combine a multi-core CPU with an FPGA fabric, both sharing access to DRAM for data exchange. This architecture leverages the general-purpose processing

CF '25, Cagliari, Italy

ACM ISBN 979-8-4007-1528-0/25/05
https://doi.org/10.1145/3719276.3725192

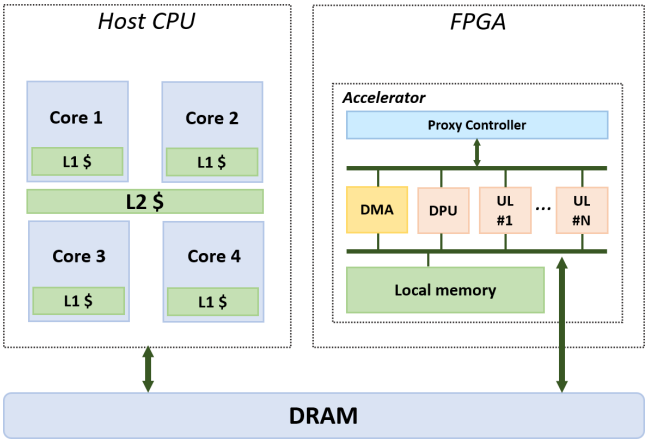

Figure 1: FPGA-based HeSoC, integrating DPU and Unsupported Layers (UL).

of the CPU and the hardware acceleration capabilities of the FPGA. A common approach to integrate high-performance accelerators involves deploying multiple accelerator clusters on the FPGA, interconnected via DRAM. Each cluster typically includes an accelerator and a DMA engine, with the CPU handling configuration, computation setup, execution, and task monitoring. However, this *remote control* mechanism introduces significant overhead due to CPU-based register accesses, adding latency that becomes a major bottleneck in systems requiring frequent CPU-accelerator interactions. Furthermore, accelerators lack a shared memory space, relying on DRAM for data exchange. This necessitates frequent and costly memory transfers, significantly impacting performance. To address these inefficiencies, prior works have explored integrating soft-processors for accelerator control [1, 2, 10]. Some approaches even offload computations to soft-cores [2], reducing data movement but failing to improve performance due to their limited computational power compared to the *host* CPU.

To overcome these limitations, our methodology introduces an accelerator *cluster* with a shared local memory and a programmable *Proxy-Controller*, tightly coupled with accelerators, DMAs, and other IPs. This enables the *Proxy Computing* paradigm, maintaining both data and control locally within the cluster. The *Proxy-Controller* orchestrates accelerator operations, minimizing costly CPU interactions and optimizing data flow within shared memory, reducing unnecessary transfers.

2.1 Integrating the AMD/Xilinx DPU

As a representative embodiment of the *Proxy Computing* design paradigm, we describe how to integrate the DPU engine in our overlay cluster. To enable the local orchestration of the *overlay*, we start analyzing the main DPU registers, that are: *1) DPU machine code*: memory address of the machine code that contains the instructions to be executed on the DPU engine; *2) Input/Output buffers*: two different registers, one to keep the memory address of the source buffer and one for the destination address; *3) Neural network parameters*: memory address of the quantized parameters of the target neural network, including weights, biases and activations; *4) DPU workspace*: starting address of a memory region that is used by the DPU engine as a temporary workspace, to store intermediate results.

```
1 // notify proxy controller to start
2 *_pcore_end   = SIG_NOT_ENDED;
3 *_pcore_start = SIG_START;
4 // busy wait for DPU execution completion
5 while (*_pcore_end == SIG_NOT_ENDED)
6     ;
```

Listing 1: Host code to trigger Proxy-Controller.

In addition to the DPU engine, several other accelerators have been integrated in our cluster, with the aim of accelerating the operations that are not directly supported by the DPU. These accelerators, illustrated as {*UL#1*, ..., *UL#N*} in the figure, have been simply designed using a standard HLS flow.

Listing 1 shows how to trigger DNN execution from the host CPU, by offloading control to the *Proxy-Controller*. Lines 2-3 trigger the execution of the *Proxy-Controller*, then the `while` loop at line 5, allows to wait for the task completion, that will be notified by the *Proxy-Controller* to the host CPU.

```
1  while (1) {
2      // wait the Cortex A53 start signal
3      while(*_pcore_start != SIG_START)
4          ;
5      // program DPU registers
6      xdpu_prog(wks, par, src, dst, instr);
7      // start the DPU engine
8      xdpu_start(SIG_START);
9      // wait DPU to complete
10     while (!xdpu_ended())
11         ;
12     xdpu_int_clear();
13     // notify the host
14     *_pcore_end = SIG_ENDED
15 }
```

Listing 2: Proxy-Controller firmware to control the DPU.

Listing 2 shows the firmware code executed by the *Proxy-Controller*. First, it waits for the trigger coming from the host CPU (line 3). After that, the DPU registers are initialized in line 6 with parameters coming from the host CPU. Line 8 is the flag to trigger the execution of the DPU engine, while in line 10, the *Proxy-Controller* polls the completion register of the DPU, indicating that the neural network execution is ended. Finally, in line 14, the host CPU is notified for the completion and the busy wait (line 5 in Listing 1) is unlocked.

3 Experimental Results

The reference platform that we used to conduct the experiments is the AMD/Xilinx ZCU102, the most widespread high-end platform to the Zynq UltraScale+ family. It features dual processor CPU complex, composed of a quad-core ARM Cortex A53 for the applications and a dual-core ARM Cortex R5 for real-time applications. To validate the proposed ideas we used the Vitis AI 2.0[1], a commercial framework from AMD/Xilinx that contains the FPGA

[1]https://github.com/Xilinx/Vitis-AI/tree/2.0

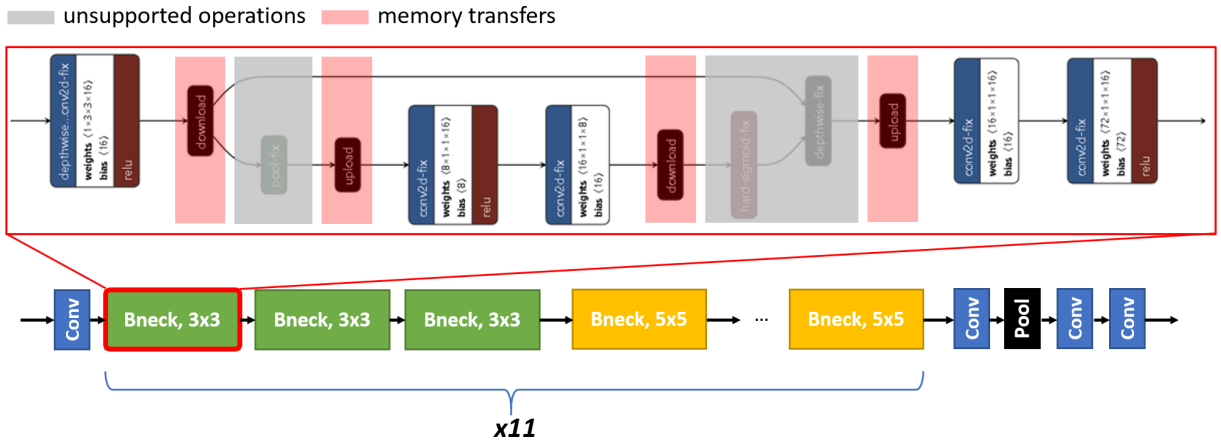

Figure 2: Structure of the *MobileNetv3 Small*, with unsupported operations and memory transfers involving DRAM and FPGA.

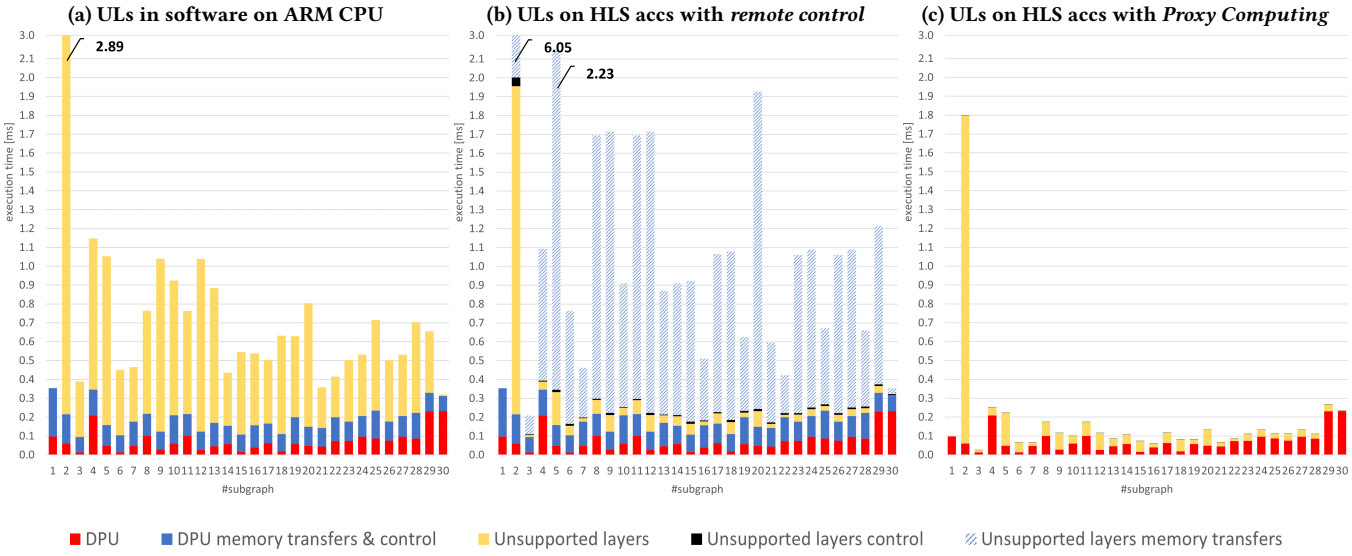

Figure 3: Comparison of Unsupported Layers (ULs) execution approaches

hardware design containing the DPU engine and all the software, libraries and applications needed to accelerate neural networks on the FPGA. AMD/Xilinx also provides the Vitis AI Model Zoo, which is a repository that contains pre-trained neural network models. To simplify the deployment of the DPU engine we implemented a hardware design containing only one DPU core synthesized with 250MHz for the FPGA logic and 500MHz for the DSPs. The four ARM Cortex A53 run a minimal Linux distribution, while the *Proxy-Controller*, implemented using the AMD MicroBlaze[2], executes in bare-metal mode. As reported in Fig. 1 we deployed local memory banks, implemented using Block RAM (BRAM). The local memory can be accessed by all the PEs.

3.1 Case study — MobileNet implementation

It is common that a custom-made neural network model can integrate layers that are not currently supported by the DPU engine, and that must be executed outside of the engine itself. This typically entails executing such layers in software on the host CPU, which implies transferring the execution between the host CPU and the DPU engine multiple times, consuming a significant fraction of the execution time and dimming the benefits reaped from the hardware acceleration.

To conduct our experiments, we analyzed the *MobileNetv3 Small* [7], a widely used neural network for image classification trained on the Imagenet dataset [4]. This model consists of repeated layer structures, as shown in Fig. 2. Its backbone includes eleven instances of the *Bneck* block, composed of *convolutions*, *ReLUs*, *sigmoids*, *depthwises*, and *global poolings*. The latter three operations (highlighted in gray) are unsupported by the DPU engine, requiring memory copies between FPGA and DRAM (red box). Consequently, the DPU compiler generates a binary with multiple neural network graphs (30 in this case), that represent the portions of the neural network that can be executed on the DPU engine. Fig. 3 presents the execution time breakdown for *MobileNetv3 Small* across three scenarios.

[2]https://www.xilinx.com/products/intellectual-property/microblazecore.html

Plot 3a shows the baseline approach, where the host CPU controls the DPU and executes unsupported layers. The red bars indicate DPU computation time, blue represents memory transfers (including control overhead such as Vitis AI Runtime (VART) library calls and AXI transactions), and yellow denotes CPU execution of unsupported layers, that is the majority of total execution time. A natural solution is to synthesize dedicated accelerators. However, traditional methodologies instantiate separate accelerators for each unsupported layer, leading to inefficient remote control and excessive memory transfers.

Plot 3b shows execution with HLS-generated accelerators, created by directly synthesizing the same C++ code used in software execution. These accelerators use local BRAM for input/output storage, with data transfers handled by the CPU. While this approach significantly accelerates unsupported layers (yellow bars), remote control overhead (black bars) and memory transfer delays (dashed blue) remain. The latter could be mitigated with manual overlap of computation and data movement, allowing an ideal best-case estimation by ignoring these delays.

Plot 3c illustrates execution with our *Proxy Computing* paradigm. While DPU computation remains unchanged, memory copies are entirely eliminated (zeroed blue bars). Dedicated HLS accelerators benefit from local execution advantages, but additionally, *local control* via the *Proxy-Controller* removes remote CPU overhead and avoids unnecessary memory transfers by leveraging shared local memory.

We evaluate the impact of our methodology on the end-to-end execution of the *MobileNetv3 Small* neural network, observing significant speedups. When using HLS accelerators with remote control and considering worst-case memory copy time, the execution is actually slower than the baseline, achieving only 0.58× speedup. In the ideal best-case scenario, where memory copy time is completely neglected, the speedup improves to 2.36×. Finally, with our *Proxy Computing* approach, we achieve a significantly higher speedup of 4.09×, demonstrating the efficiency of our methodology in reducing overhead and optimizing execution.

4 Conclusion

This work studied how our overlay infrastructure enabling the *Proxy Computing* approach targeting FPGA HeSoCs, can be effectively adopted to optimize the inference latency of AI models using the Deep Learning Processing Unit (DPU). Our proposed mechanism allows: i) to reduce costly software interactions of the typical *remote* form of accelerator control, between host CPU and FPGA; ii) to reduce unnecessary data movements involving distinct memory spaces. We showed that if the neural network model is composed of multiple sub-graphs, as in the case of the *MobileNetv3 Small* neural network, using our *Proxy Computing* paradigm, we can achieve up to 4× of speedup compared to the state of the art.

Acknowledgments

This work has received funding from the European Union's Horizon 2020 programme dAIEDGE (G.A. No 101120726). The project is supported by the Chips joint Undertaking and its members, including the top-up funding by the national Authorities of Germany, Belgium, Spain, Finland, Netherlands, Austria, Italy, Greece, Latvia, Lithuania and Turkey, under grant agreement number 101139996-2. Co-funded by the European Union. This work was supported by the European Union under the NextGenerationEU Programme within the Plan "PNRR - Missione 4 "Istruzione e Ricerca" - Componente C2 Investimento 1.1 "Fondo per il Programma Nazionale di Ricerca e Progetti di Rilevante Interesse Nazionale (PRIN)" by the Italian Ministry of University and Research (MUR)". Project Title: "Simplifying Predictable and energy-efficient Acceleration from Cloud to Edge (SPACE)", Project code: 202254AM7H, CUP: E53D23007810006, MUR D.D. financing decree n. 959, 30th June 2023.

References

[1] Gianluca Bellocchi, Alessandro Capotondi, Francesco Conti, and Andrea Marongiu. 2021. A RISC-V-based FPGA Overlay to Simplify Embedded Accelerator Deployment. In *2021 24th Euromicro Conference on Digital System Design (DSD)*. IEEE, Vienna, Austria, 9–17. doi:10.1109/DSD53832.2021.00011

[2] Andrea Bernardi, Gianluca Brilli, Alessandro Capotondi, Andrea Marongiu, and Paolo Burgio. 2022. An FPGA Overlay for Efficient Real-Time Localization in 1/10th Scale Autonomous Vehicles. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, Antwerp, Belgio, 915–920. doi:10.23919/DATE54114.2022.9774517

[3] G. Brilli, G. Valente, A. Capotondi, P. Burgio, T. Di Mascio, P. Valente, and A. Marongiu. 2023. Fine-Grained QoS Control via Tightly-Coupled Bandwidth Monitoring and Regulation for FPGA-based Heterogeneous SoCs. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, San Francisco, California, USA, 1–6. doi:10.1109/DAC56929.2023.10247840

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, Miami, Florida, USA, 248–255. doi:10.1109/CVPR.2009.5206848

[5] Farzad Farshchi, Qijing Huang, and Heechul Yun. 2019. Integrating NVIDIA Deep Learning Accelerator (NVDLA) with RISC-V SoC on FireSim. In *2019 2nd Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*. IEEE, Virtual Conference, 21–25. doi:10.1109/EMC249363.2019.00012

[6] Davide Giri, Kuan-Lin Chiu, Guy Eichler, Paolo Mantovani, Nandhini Chandramoorthy, and Luca Carloni. 2019. Ariane + NVDLA: Seamless Third-Party IP Integration with ESP. In *Fifth Workshop on Computer Architecture Research Directions. CARD 2019*. ACM, Phoenix, Arizona, USA, 1–10.

[7] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. 2019. Searching for MobileNetV3. arXiv:1905.02244 [cs.CV] https://arxiv.org/abs/1905.02244

[8] Yutian Lei, Qingyong Deng, Saiqin Long, Shaohui Liu, and Sangyoon Oh. 2020. An Effective Design to Improve the Efficiency of DPUs on FPGA. In *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE Computer Society, Hong Kong, 206–213. doi:10.1109/ICPADS51040.2020.00036

[9] Antonio Loquercio, Ana I. Maqueda, Carlos R. del Blanco, and Davide Scaramuzza. 2018. DroNet: Learning to Fly by Driving. *IEEE Robotics and Automation Letters* 3, 2 (2018), 1088–1095. doi:10.1109/LRA.2018.2795643

[10] Simone Machetti, Pasquale Davide Schiavone, Thomas Christoph Müller, Miguel Peón-Quirós, and David Atienza. 2024. X-HEEP: An Open-Source, Configurable and Extendible RISC-V Microcontroller for the Exploration of Ultra-Low-Power Edge Accelerators. arXiv:2401.05548 [cs.AR] https://arxiv.org/abs/2401.05548

[11] Sparsh Mittal. 2019. A Survey on optimized implementation of deep learning models on the NVIDIA Jetson platform. *Journal of Systems Architecture* 97 (2019), 428–442. doi:10.1016/j.sysarc.2019.01.011

[12] Francesco Restuccia and Alessandro Biondi. 2021. Time-Predictable Acceleration of Deep Neural Networks on FPGA SoC Platforms. In *2021 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, Virtual Conference, 441–454. doi:10.1109/RTSS52674.2021.00047

[13] Gerlando Sciangula, Francesco Restuccia, Alessandro Biondi, and Giorgio Buttazzo. 2022. Hardware Acceleration of Deep Neural Networks for Autonomous Driving on FPGA-based SoC. In *2022 25th Euromicro Conference on Digital System Design (DSD)*. IEEE, Maspalomas, Gran Canaria, Spain, 406–414. doi:10.1109/DSD57027.2022.00061

[14] Xilinx Inc. 2020. *Zynq UltraScale+ MPSoC Overview*. Xilinx Inc. https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html.

[15] Xilinx, Inc. 2024. *Xilinx Deep Learning Processing Unit (DPU)*. Xilinx, Inc. Available at: https://www.xilinx.com/products/intellectual-property/dpu.html.