A DNN-based Background Segmentation Accelerator for FPGA-equipped satellites

Invited Paper

Michele Fiorito
Politecnico di Milano
Milano, Italy
michele.fiorito@polimi.it

Serena Curzel
Politecnico di Milano
Milano, Italy
serena.curzel@polimi.it

Giovanni Gozzi
Politecnico di Milano
Milano, Italy
giovanni.gozzi@polimi.it

Fabrizio Ferrandi
Politecnico di Milano
Milano, Italy
fabrizio.ferrandi@polimi.it

ABSTRACT

The computational requirements of complex vision-based navigation algorithms influence the design of on-board processors for satellites, often leading to the adoption of solutions based on field programmable gate arrays (FPGAs). Following this trend, the HERMES project (qualification of High pErformance pRogrammable Microprocessor and dEvelopment of Software ecosystem) aims at providing both radiation-hardened FPGAs for aerospace applications and dedicated tools to program them at a high level of abstraction. In this paper, we demonstrate the use of design automation methods developed within HERMES to implement a deep neural network model on a space-grade FPGA, optimizing it to fit memory footprint and performance constraints.

CCS CONCEPTS

• **Hardware → High-level and register-transfer level synthesis**; **Reconfigurable logic and FPGAs**.

KEYWORDS

FPGA, aerospace, High-Level Synthesis, Deep Learning

ACM Reference Format:
Michele Fiorito, Serena Curzel, Giovanni Gozzi, and Fabrizio Ferrandi. 2024. A DNN-based Background Segmentation Accelerator for FPGA-equipped satellites: Invited Paper. In *Proceedings of the 21st ACM International Conference on Computing Frontiers Workshops and Special Sessions (CF '24 Companion), May 7–9, 2024, Ischia, Italy*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3637543.3652979

1 INTRODUCTION

Aerospace applications increasingly rely on the processing of data coming from imaging sensors, for both payload operations and vision-based navigation. While routine spacecraft operations such as decoding commands from a ground station, logging status reports, management of subsystems, and attitude control can be executed by simple microcontrollers, computer vision algorithms require more processing power, often significantly higher than what is available on space-grade, radiation-hardened CPUs. In this context, commercial-off-the-shelf (COTS) devices and heterogeneous platforms can provide the required performance, and devices based on field programmable gate arrays (FPGAs) are particularly suited because of their low overhead in size, power consumption, and cost.

Alongside performance and power consumption, the reliability of accelerators and soft-cores implemented on FPGA is a major concern when they are deployed in harsh environments. For this reason, European institutions funded several efforts to develop a new generation of radiation-hardened FPGAs, and the supply from European industries is quickly improving to meet current and future demands. HERMES - qualification of High pErformance pRogrammable Microprocessor and dEvelopment of Software ecosystem [11] is one of such efforts, with the dual target of developing next-generation FPGAs for aerospace applications and providing tools that facilitate their usage. In fact, one obstacle to the adoption of FPGAs is often the required expertise in low-level hardware programming, which is not common among software engineers. HERMES thus relies on High-Level Synthesis (HLS), and in particular on the open-source Bambu HLS tool [7], to automatically generate optimized FPGA implementations starting from high-level software descriptions. Bambu has been extended during HERMES to support three space-grade FPGAs from NanoXplore (NG-MEDIUM, NG-LARGE, and NG-ULTRA) through the integration of the NanoXplore logic synthesis tool and through a process of characterization, i.e., by collecting latency and resources consumption of functional units on each of the boards to inform the allocation and scheduling steps during the HLS process. Other improvements to Bambu included support for industry-standard AXI protocol interfaces, and the introduction of caches to reduce the memory access latency [8].

The HERMES use cases included computer vision kernels to be accelerated on space-grade FPGAs. Vision-based navigation algorithms that could benefit from hardware acceleration also include machine learning (ML) techniques used for image classification, segmentation, and other compute-intensive tasks. In this paper, we use a deep neural network (DNN) model designed to analyze images and distinguish a satellite from the background (background segmentation) to describe a synthesis and optimization flow from the training of the DNN in a high-level ML framework to the generation of a bitstream to program the FPGA. We start from the combination of HLS and the Multi-Level Intermediate Representation (MLIR) that was proposed in the SODA framework [1] to automatically synthesize DNNs, and we customize the compilation pipeline to fit the memory footprint and resource constraints of a NanoXplore FPGA. In fact, by default SODA targets an ASIC design with the highest possible performance: the frontend optimization pipeline preferably applies techniques that sacrifice area to reduce latency

CF '24 Companion, May 7–9, 2024, Ischia, Italy

ACM ISBN 979-8-4007-0492-5/24/05.
https://doi.org/10.1145/3637543.3652979

(e.g., loop unrolling), which is undesirable when targeting a space-grade FPGA with a fixed number of logic and memory resources. Preliminary results help us to plan a roadmap of improvements to Bambu and to the MLIR frontend that will provide users with more flexibility to synthesize diverse DNN models and obtain high performance at a reasonable cost in terms of area.

The rest of the paper is structured as follows:

- We briefly introduce necessary background concepts and the state of the art in Section 2;
- We describe our design and optimization flow in Section 3;
- We present preliminary results obtained synthesizing DNN models in Section 4;
- We conclude the paper with some final remarks in Section 5.

2 STATE OF THE ART

FPGAs are extensively used in computing systems for space missions [9, 14], where performance, power consumption, and reliability are equally critical. Vision-based navigation algorithms for satellites, in particular, require significantly faster processing than what space-grade CPUs can provide, as they need to process high-definition images at high frame rates performing feature extraction, matching, object tracking, and other compute-intensive tasks. In this domain, FPGAs can reach the highest performance per Watt ratio among other COTS platforms [13]. The NG-ULTRA platform from NanoXplore, which is the main target for HERMES [11], is a system-on-chip (SoC) integrating a quad-core CPU and a radiation-hardened FPGA, prioritizing high reliability through hardening techniques in the manufacturing process and a design featuring triple modular redundancy, error correction, and memory integrity checks; in cases where the application to be accelerated is not safety-critical, a commercial FPGA not designed for use in space may be used instead, sacrificing reliability for performance.

A possible downside of FPGA-based platforms is that they require a bigger programming effort when designing accelerators manually with a hardware description language (HDL) such as Verilog or VHDL. However, many HLS tools exist that can efficiently translate high-level software code into an HDL representation [4], simplifying the design and the verification of FPGA accelerators. The generated HDL code can be integrated into a larger design as an IP block, translated into a bitstream through commercial logic synthesis and implementation tools, and deployed on the FPGA. Increasing the degree of automation in the design flow through HLS allows software developers with limited hardware design expertise to exploit the increased performance provided by FPGAs. Most HLS tools are part of commercial design suites that support target platforms from a single vendor, while Bambu [7] is an open-source HLS tool supporting FPGAs from AMD/Xilinx, Intel, Lattice, and NanoXplore. Accelerators synthesized through Bambu can be interfaced with a host CPU, external memory, or other accelerators through the AXI protocol, and they can include a custom cache to mitigate the effect of external memory accesses [8].

The vision-based application considered in this study includes a DNN model for background segmentation, i.e., a neural network trained to distinguish pixels belonging to an object in the foreground (in this case, a satellite or other spacecraft) from pixels belonging to the background, as shown in Figure 1. Training such models can be difficult because of the absence of a reference dataset with labeled images specific to the domain; synthetic datasets have been proposed where both the background and the spacecraft are rendered through a simulator with varying conditions of lighting, spacecraft position and orientation, Earth atmospheric conditions, and so on [15]. Another option is to apply transfer learning or other post-training techniques to existing convolutional neural network (CNN) models [10]; these models, however, tend to be too large to fit on the limited memory resources of an FPGA.

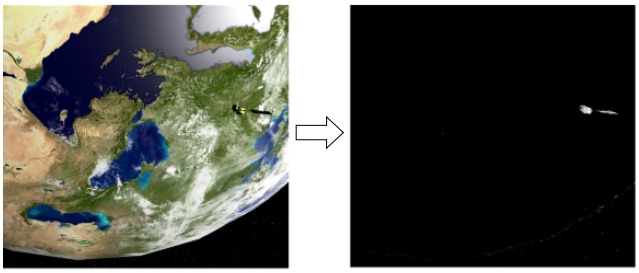


Figure 1: Background segmentation task.

Traditionally, the input to HLS tools is a program written in C/C++, together with timing and resource utilization constraints. When the application to be accelerated is a pre-trained DNN, however, its designers would have to manually translate a higher-level representation (usually Python-based) to C/C++, which is highly impractical. Tools like hls4ml [6] or FINN [2] cover such an abstraction gap by parsing DNN models and replacing operators with corresponding C/C++ functions, taken from a library of templates that already contain HLS optimization directives. Domain-specific compilation frameworks provide more flexible solutions that do not depend on pre-optimized libraries; in particular, the MLIR framework [12] allows to interface with popular DNN frameworks and progressively lower models through multiple levels of abstraction (called dialects). ScaleHLS [16] exploits MLIR to analyze and transform input code from C or PyTorch, generating annotated C++ code for the AMD/Xilinx HLS tool through a design space exploration engine that automatically identifies the best combination of optimization directives. Code generated through ScaleHLS is only useful when targeting AMD/Xilinx FPGAs, and because it relies on early HLS estimates it may lead to underestimating resource consumption. The SODA framework [1] integrates an MLIR-based frontend [3] and Bambu, obtaining an open-source, end-to-end design automation flow from DNN frameworks to hardware design that can target any FPGA supported by Bambu. The SODA-OPT optimization pipeline in the frontend applies several transformations with the aim of exposing instruction-level parallelism so that Bambu can later schedule multiple operations in parallel; such a strategy proved to be very effective for polyhedral benchmarks and isolated DNN layers translated into ASIC designs, but it causes excessive resource consumption when considering a complete DNN and when the target is a space-grade FPGA.

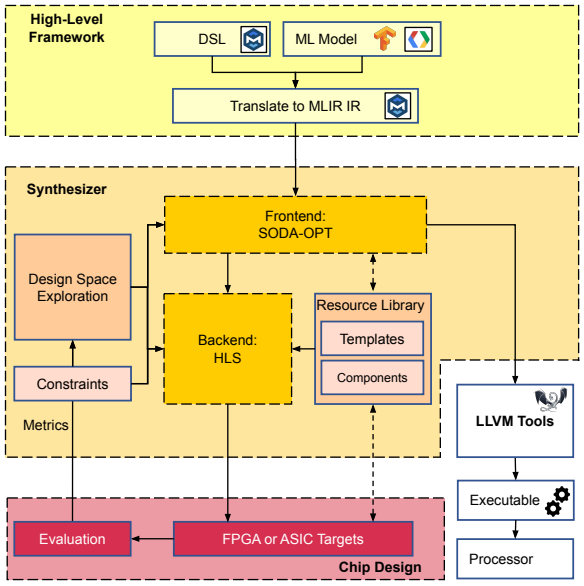


Figure 2: The SODA framework design flow [1].

3 ACCELERATOR DESIGN

The starting point for this work is the SODA framework and design flow [1] (Figure 2), an end-to-end compiler-based design automation framework able to synthesize DNN models into hardware accelerators. In its original version, SODA takes as input a model designed and trained in TensorFlow. The model is serialized in the Protobuf format and translated into an MLIR representation (`tf` dialect), which is then lowered through subsequent dialects in the SODA-OPT frontend and eventually translated into an LLVM IR that can be synthesized with Bambu for one of the supported FPGA/ASIC targets. SODA-OPT also performs system-level design: in fact, it is not the whole MLIR IR that undergoes progressive lowering and synthesis, but only the kernels marked by the user through the custom `soda` dialect, while the rest of the application is compiled into a host executable. During the lowering process, a default optimization pipeline is applied that unrolls loops to increase instruction-level parallelism and prepares the kernels for HLS.

The goal of this work is to synthesize an accelerator targeting a space-grade FPGA from NanoXplore, and the basic SODA flow has a few limitations that make it unsuitable for this purpose. Table 1 lists the type and quantity of available resources on the three FPGA platforms considered in the HERMES project. (Changing the synthesis target to one of them simply requires adjusting Bambu options, as they have already been characterized and downstream logic synthesis with NanoXplore tools is already supported in Bambu [8].) Their limited capacity in terms of digital signal processing elements (DSPs), registers, look-up tables (LUTs), and memory elements requires careful consideration of the trade-off between performance and resources consumption, whereas by default, SODA focuses on extracting the maximum performance even if it incurs a higher cost in terms of resources.

The first choice that reduces resources consumption, and at the same time improves performance, is to start from a *quantized* DNN

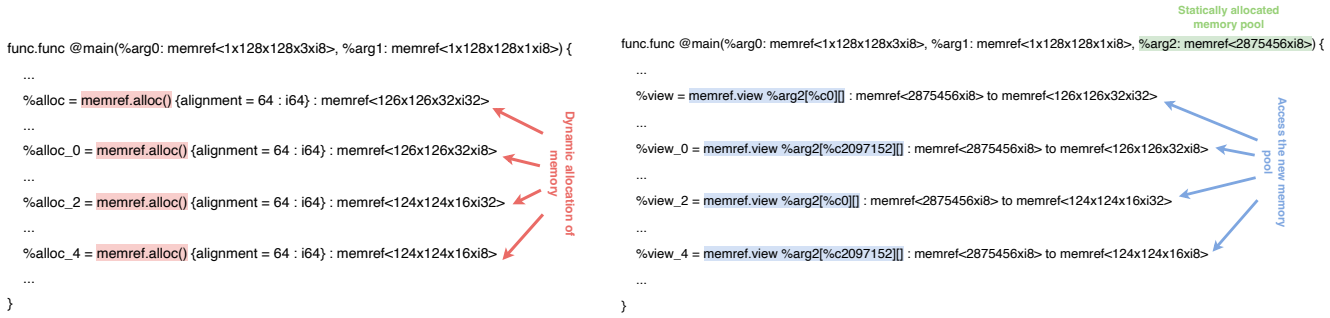
Table 1: Main available resources in space-grade FPGAs from NanoXplore.

Device	Registers	LUTs	DSPs	Memory blocks
NG-MEDIUM	32256	34272	112	56
NG-LARGE	129024	137088	384	192
NG-ULTRA	505344	536928	1344	672

model. TensorFlow models are usually designed and trained in single- or double-precision, but it is possible to exploit the TensorFlow Lite converter to quantize weights and activations into 8-bit integer numbers, resulting in a smaller and faster model which is more suitable for inference on edge devices. After conversion, the model can be serialized in the Flatbuffer format and translated into an MLIR IR in the `tflite` dialect, which is then lowered with a very similar pipeline to the one used for `tf` to obtain a synthesizable LLVM IR. Because integer functional units can be implemented more efficiently on FPGA, such a quantized model will result in lower resources consumption and shorter execution time. The accuracy of the model results is likely going to be lower than the one obtained with the original floating-point model; however, if it falls below mission requirements, post-quantization fine-tuning and re-training techniques in TensorFlow Lite can be applied.

The default transformation and optimization pipeline in the SODA-OPT frontend also needs to be modified. We don't use the outlining feature, as the whole DNN needs to be accelerated, and host code is not provided in the input MLIR IR. We then disable the three full loop unrolling commands in the `affine` dialect, as they would result in unacceptable area consumption. The memory footprint of a DNN model is also a concern for space-grade FPGAs, since if the weights cannot fit the limited on-chip memory blocks, each load and store access to an external memory will have a significant cost in terms of latency. The default SODA-OPT pipeline optimizes memory usage only at the layer level, as it was designed with single-layer accelerators in mind, leaving buffers between one DNN layer and the next to be lowered with the default MLIR strategy. In this way, however, each buffer becomes a dynamic memory allocation operation in the `memref` dialect. We introduced a new MLIR pass that allows instead to generate optimized, statically allocated memory pools, mitigating resource constraints and improving overall performance.

Taking inspiration from similar efforts targeting embedded devices [5], our pass starts from the analysis of def-use chains to determine liveness information for existing buffers, indicating whether two or more of them are used concurrently within the program. We use liveness information to build a Memory Exclusion Graph (MEG) where nodes represent buffers and edges represent conflicts between them; the pass then has to solve a graph coloring problem on the MEG to generate an optimized set of memory pools where multiple buffers can be allocated, sharing the same memory region at different times during execution. Graph coloring is known to be NP-complete, so we use a simple heuristic to solve it in a timely way. The current heuristic provides satisfactory results, but more sophisticated algorithms could be implemented in the future to enhance the accuracy and efficiency of the memory optimization process, leading to even better resource utilization and performance in the



(a) Default `memref` buffers between DNN layers. **(b) Statically allocated, optimally sized memory pool.**

Figure 3: MLIR IR before and after applying our static memory allocation pass.

generated hardware accelerators. With this optimization pass we reduce the memory footprint of the accelerators and we solve the memory allocation problem statically, avoiding the generation of runtime calls to dynamic memory allocation primitives which are not synthesizable on FPGA. Figure 3 illustrates the effect of our pass on the MLIR IR: on the left the default MLIR buffer generation strategy results in frequent `memref.alloc` operations between DNN layers, on the right a single memory pool has been statically sized and added to the function arguments, and buffers are represented through `memref.view` operations that access it.

Finally, one important aspect that is missing in the current design flow concerns vectorization, which is another possible way to balance resources consumption and performance. Vectorization is possible through standard passes within the `linalg` and `affine` MLIR dialects, so integrating appropriate options in the SODA-OPT pipeline would immediately result in the generation of a vectorized LLVM IR, possibly with a customizable vector size. Future efforts will focus on design space exploration in the MLIR frontend, and on supporting the synthesis of vectorized functional units in Bambu.

4 EARLY RESULTS

The final target of our study is a DNN accelerator that performs background segmentation on images captured by a satellite that depict another spacecraft. Traditional DNNs for image segmentation are composed by concatenating a feature extraction model with a series of upsampling layers, both containing mostly convolution operations. Our model has a similar structure, requiring several million multiplications and roughly 13.000 floating-point weights, although we cannot disclose more details about its design and training due to industrial secrecy. Because synthesis and simulation times are substantially long for a hardware design containing such a high amount of operations, we first tested our design flow on a smaller DNN, i.e., a simple digit classifier trained on the MNIST dataset, composed of a convolutional layer followed by a fully connected one. The DNN has been designed and trained in TensorFlow, translated into MLIR, and synthesized for a NanoXplore NG-ULTRA FPGA with a target clock period of 20ns following the design flow described in Section 3.

Table 2 shows performance and area metrics for different design and synthesis configurations. The first configuration we tested (fp32-external) was to synthesize the model as it was trained, with floating-point weights and calculations. Because Impulse, the NanoXplore logic synthesis tool, struggled to meet the requested clock period in the place and route phase, we aimed for low area consumption by disabling loop unrolling and constraining Bambu to prioritize resource sharing. We also had to place all storage in external memory because of an unknown error during logic synthesis, resulting in additional clock cycles for every load and store operation. Subsequently, we added integer quantization in TensorFlow Lite keeping the rest of the synthesis options unchanged (int8-external), placing storage in internal BRAMs (int8-internal), and removing the resource sharing constraints (int8-internal-opt). Looking at the results in the table, this last configuration achieves the highest performance in terms of execution time: Bambu manages to exploit a higher degree of parallelism by using more FPGA resources, integer functional units are smaller and faster than their floating-point counterparts, and memory access times are reduced by keeping all storage on the FPGA. The critical path, however, exceeds the requested 20ns.

Considering the results obtained on a smaller model, we immediately quantized the larger DNN model for background segmentation to 8-bit integer values, as the floating-point version would not likely fit the resource constraints of the target FPGA since the digit classifier model is at least one order of magnitude smaller in terms of number of multiplications. The effect of our memory optimization pass was not significant on the digit classifier model as it was composed of only two layers, but it was instead extremely relevant when we moved to the larger DNN model for background segmentation. The input model, in fact, contained 9.54MB of allocated memory buffers which were reduced to 2.74MB after applying the pass. Early synthesis results are reported in Table 3: the achieved performance is quite low, to the point that the simulation process reached its timeout set at 200 million clock cycles, while only about 5% of LUTs and registers are occupied. This means that there is still ample room to explore optimization strategies that better exploit parallelism, although the inability of Impulse to reach the desired target period is a concern, as well as the increased synthesis times that slow down the exploration process. Implementing vectorization passes in the frontend and corresponding vectorized functional units in Bambu will undoubtedly prove helpful to increase performance; it

Table 2: Synthesis results for a simple digit classifier on NG-ULTRA.

Configuration	Latency	Clock Cycles	Frequency	Memory blocks	LUTs	Registers	DSPs
fp32-external	51.729 ms	2114052	40.9 MHz	0	30078	24410	4
int8-external	16.844 ms	844744	50.1 MHz	0	3470	3976	12
int8-internal	11.350 ms	670316	59.1 MHz	37	2523	2963	9
int8-internal-opt	3.951 ms	178436	45.2 MHz	40	4625	6779	27

Table 3: Synthesis result for the DNN backround segmentation model on NG-ULTRA.

Configuration	Latency	Clock cycles	Frequency	Memory blocks	LUTs	Registers	DSPs
int8-internal-opt	>5s	>200M	37.2 MHz	47	24312	26814	59

will also likely introduce less complicated logic in the accelerator datapath with respect to achieving the same level of parallelism through loop unrolling and replication of scalar functional units, simplifying bitstream generation in Impulse after the accelerator has been synthesized.

5 CONCLUSION

The HERMES project and the SODA framework provide application developers with a set of tools that simplify the generation of accelerators for space-grade FPGAs, enabling automated synthesis of DNN models into custom hardware. We have applied such tools and introduced new features to generate an accelerator for a convolutional neural network performing background segmentation, obtaining a baseline design that will serve as a starting point for the exploration of new optimizations, with vectorization as our next target feature. In the future, it will also be fundamental to collaborate with the application experts who designed and trained the model, to verify that our modifications (in particular quantization) do not break the model's functionality and maintain acceptable accuracy on a relevant dataset.

ACKNOWLEDGMENTS

This research was partially supported by the HERMES project funded by the EU Horizon 2020 Program under grant agreement No 101004203.

REFERENCES

[1] Nicolas Bohm Agostini, Serena Curzel, Jeff Jun Zhang, Ankur Limaye, Cheng Tan, Vinay Amatya, Marco Minutoli, Vito Giovanni Castellana, Joseph Manzano, David Brooks, Gu-Yeon Wei, and Antonino Tumeo. 2022. Bridging Python to Silicon: The SODA Toolchain. *IEEE Micro* 42, 5 (2022), 78–88.

[2] Michaela Blott, Thomas B Preußer, Nicholas J Fraser, Giulio Gambardella, Kenneth O'brien, Yaman Umuroglu, et al. 2018. FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks. *ACM Transactions on Reconfigurable Technology and Systems* 11, 3 (2018), 1–23.

[3] Nicolas Bohm Agostini, Serena Curzel, Vinay Amatya, Cheng Tan, Marco Minutoli, Vito Giovanni Castellana, Joseph Manzano, David Kaeli, and Antonino Tumeo. 2022. An MLIR-based Compiler Flow for System-Level Design and Hardware Acceleration. In *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–9.

[4] J Cong, J Lau, G Liu, S Neuendorffer, P Pan, K Vissers, and Z Zhang. 2022. FPGA HLS Today: Successes, Challenges, and Opportunities. *ACM Transactions on Reconfigurable Technology and Systems* 15, 4 (2022), 1–42.

[5] Karol Desnos, Maxime Pelcat, Jean-François Nezan, and Slaheddine Aridhi. 2015. Memory Analysis and Optimized Allocation of Dataflow Applications on Shared-Memory MPSoCs. *Journal of Signal Processing Systems* 80, 1 (July 2015), 19–37. https://doi.org/10.1007/s11265-014-0952-6

[6] Javier Duarte, Song Han, Philip Harris, Sergo Jindariani, Edward Kreinar, Benjamin Kreis, J Ngadiuba, M Pierini, N Tran, and Z Wu. 2018. Fast inference of deep neural networks in FPGAs for particle physics. *Journal of Instrumentation* 13, 07 (2018), P07027.

[7] F. Ferrandi, V. G. Castellana, S. Curzel, P. Fezzardi, M. Fiorito, et al. 2021. Bambu: an Open-Source Research Framework for the High-Level Synthesis of Complex Applications. In *Proceedings of the 58th ACM/IEEE Design Automation Conference (DAC)*. 1327–1330.

[8] Fabrizio Ferrandi, Michele Fiorito, Claudio Barone, Giovanni Gozzi, and Serena Curzel. 2024. High-Level Synthesis Developments in the Context of European Space Technology Research. In *15th Workshop on Parallel Programming and Run-Time Management Techniques for Many-Core Architectures and 13th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms (PARMA-DITAM 2024) (Open Access Series in Informatics (OASIcs), Vol. 116)*. 1:1–1:12.

[9] Alan D. George and Christopher M. Wilson. 2018. Onboard Processing With Hybrid and Reconfigurable Computing on Small Satellites. *Proc. IEEE* 106, 3 (2018), 458–470.

[10] Yefei Huang, Tianlai Xu, Zexu Zhang, Hutao Cui, and Yu Su. 2022. Satellite Segmentation with Pre-trained CNN Models. *Journal of Physics: Conference Series* 2171, 1 (jan 2022), 012003.

[11] Nadia Ibellaatti, Edouard Lepape, Alp Kilic, Kaya Akyel, Kassem Chouayakh, Fabrizio Ferrandi, Claudio Barone, Serena Curzel, Michele Fiorito, Giovanni Gozzi, et al. 2023. HERMES: qualification of High pErformance pRogrammable Microprocessor and dEvelopment of Software ecosystem. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–5.

[12] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2021. MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In *IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. 2–14.

[13] George Lentaris, Konstantinos Maragos, Ioannis Stratakos, Lazaros Papadopoulos, Odysseas Papanikolaou, Dimitrios Soudris, Manolis Lourakis, Xenophon Zabulis, David Gonzalez-Arjona, and Gianluca Furano. 2018. High-Performance Embedded Computing in Space: Evaluation of Platforms for Vision-Based Navigation. *Journal of Aerospace Information Systems* 15, 4 (2018), 178–192.

[14] N Montealegre, D Merodio, A Fernández, and P Armbruster. 2015. In-flight reconfigurable FPGA-based space systems. In *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. 1–8.

[15] Pedro F. Proença and Yang Gao. 2020. Deep Learning for Spacecraft Pose Estimation from Photorealistic Rendering. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 6007–6013. https://doi.org/10.1109/ICRA40945.2020.9197244

[16] Hanchen Ye, HyeGang Jun, Hyunmin Jeong, Stephen Neuendorffer, and Deming Chen. 2022. ScaleHLS: A Scalable High-Level Synthesis Framework with Multi-Level Transformations and Optimizations. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*. 1355–1358.