TaPaSCo: Towards a Plug-and-Play FPGA Experience

Torben Kalkhof
kalkhof@esa.tu-darmstadt.de
Technical University of Darmstadt
Darmstadt, Germany

David Volz
volz@esa.tu-darmstadt.de
Technical University of Darmstadt
Darmstadt, Germany

Andreas Koch
koch@esa.tu-darmstadt.de
Technical University of Darmstadt
Darmstadt, Germany

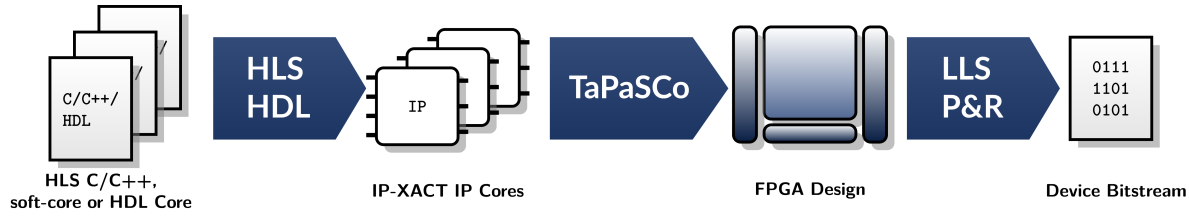

Figure 1. The TaPaSCo bitstream generation toolflow.

1 Introduction

A familiar pattern persists: with every new chip release, hardware grows faster, more powerful, and increasingly complex. Yet, tool support continues to lag behind this rapid advancement. Especially in the FPGA community, we had to get used to this over the past few decades. We are far away from anything plug-and-play-like, and lucky if synthesis tools at least support all features of the newest FPGA generation.

Of course, things have improved compared to the very beginning of FPGAs. AMD has established a robust ecosystem with Vitis [3] and XRT [2], which have supported many of their FPGAs over the years. However, they discontinued XRT for their newest cards, such as the Alveo V80. The replacement is the AVED [1] reference design, which does not provide a corresponding software stack, and does also not cover all hardware features of the card, such as networking. Again, users suffer from lack of comprehensive tool support.

However, the open-source community is here to rescue us. Among other initiatives, our TaPaSCo framework, established in 2015 and continuously updated since, is not only one of the oldest players on the market, but also has the widest range of supported devices and features. In the following, we provide a brief overview of TaPaSCo as well as highlight our most recent advances towards providing a plug-and-play FPGA experience.

2 The TaPaSCo Open-Source Framework

```
/* Initialize TaPaSCo FPGA device */
Tapasco tapasco;
/* data buffer */
std::vector<int> v = {0, 1, ...};
auto buf = makeWrappedPointer(v.data(),
        v.size() * sizeof(int));
/* Launch a TaPaSCo task */
auto myTask = tapasco.launch(PE_TYPE, buf, 42);
/* Wait on myTask for completion */
myTask();
```

Figure 2. Example of a very simple but fully-working host application using the TaPaSCo API.

The Task Parallel System Composer (TaPaSCo) comprises a hardware toolflow to generate FPGA designs and a software API to include custom accelerators into host applications. The hardware toolflow is depicted in Figure 1.

TaPaSCo accepts user-provided accelerators, we call them Processing Element (PE), as HLS C/C++ kernels, or directly as pre-compiled HLS or HDL cores in the IP-XACT format. PEs must use standard AXI interfaces for CSR and memory access, and adhere to the defined CSR layout to be compatible with the TaPaSCo driver and runtime. The toolflow will then generate the FPGA design and place all required infrastructure in a shell around the user-provided PEs before launching the vendor synthesis tool. The entire bitstream generation requires only *one* command on the command line and makes TaPaSCo really easy to use. On top, the Design Space Exploration feature allows optimizing your design in area and frequency.

The TaPaSCo API is available in C++ and Rust, and is kept very simple as well. A handful of lines of code suffice to launch a task on a PE, as shown in the code example in Figure 2. In addition to task dispatch and launch, our API implicitly handles device memory management, required

LATTE'26, March 23, 2026, Pittsburgh, PA, USA

data movements from and to device memory, and passing arguments to the PE. By marking the data vector using `makeWrappedPointer()` in Figure 2, TaPaSCo knows that this buffer should be transferred to and from device memory during `launch()` and `myTask()`. If required for finer control, API calls for manual memory management are available.

Device Support. From the beginning, platform independence has been a key design target of TaPaSCo. Especially, the wide range of supported platforms lets TaPaSCo stand out against other frameworks. On the one hand, we support very small ZYNQ-based devices, e.g., the Ultra96 board, up to large data center acceleration cards, such as the Alveo U280. On the other hand, multiple generations of FPGAs are included, starting with 7-series platforms like the VC709, reaching up to the VCK5000 board as the first-generation Versal.

On the hardware level, we achieve platform independence by hiding all platform-specific components in the generated shell. On ZYNQ devices, we run Linux and the TaPaSCo software stack on the ARM processing system, and use the available AXI ports for communication with the hardware and DDR access. In contrast, the shell for PCIe-based accelerator cards must provide a PCIe bridge for host communication, a DMA engine and DDR controller. In addition, TaPaSCo instantiates an interrupt controller and generates all required interconnect infrastructure.

Also, host applications are fully portable since the runtime and driver take care of any platform-specific parts. The user API is completely agnostic and TaPaSCo applications run on both x86 hosts and ARM-based embedded SoCs. This makes it very easy to move an application to a smaller or larger device, but also to move to a newer generation FPGA without changing anything in your design.

Feature Support. Furthermore, TaPaSCo has seen many extensions over the years, making more features of the supported devices available to the user. Probably the most used extension is the **networking** (SFPPLUS) feature. Via standard AXI4-Stream interfaces, user PEs may use 1G/10G/100G Ethernet and Aurora on devices with respective hardware support. With TaPaSCo **RISC-V** [9], we added easy access of pre packaged soft-cores into any design. Users can choose from small in-order cores to large out-of-order cores with caches and use TaPaSCo to upload, run, and debug their firmware. The **SVM** extension enables Shared Virtual Memory implemented using physical page migrations for the Alveo U280 card [10, 11]. While DDR memory is usually the primary memory used in TaPaSCo (with the exception of the Alveo U50), our **HBM** feature allows using HBM as additional scratch pad memory on UltraScale+ devices.

Direct FPGA-to-NVMe Access. Our newest release added support for streaming-based access to NVMe SSDs located on the same PCIe bus. TaPaSCo ensures proper communication with the SSD, including initialization, queue management, and DMA transfers. The user PE on the other hand needs only to implement standard AXI4-Stream ports to load or store a data stream starting at a specific address. This provides an easy start for hardware designs that require data-sets far exceeding the memory capacity of todays FPGAs.

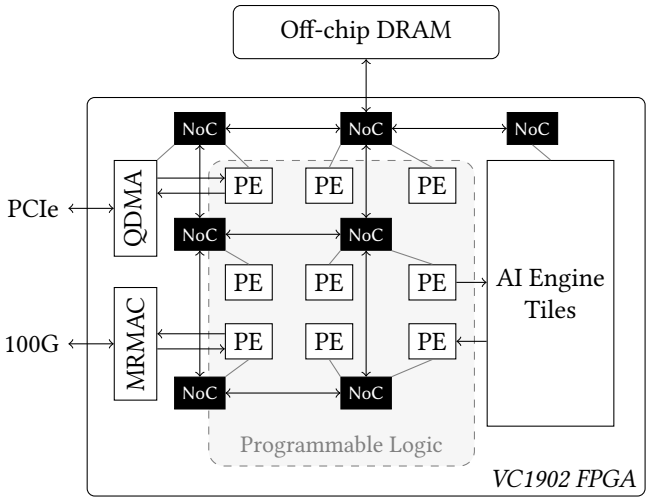

Figure 3. TaPaSCo system overview on a Versal device.

Our publication [12] implements different buffer strategies to move data between FPGA and NVMe SSD. Comparing DMA transfers using URAM, on-board DRAM, and host DRAM as backing memory showed that the easier to implement options resulted in worse performance. Only when using host DRAM – thus bypassing PCIe-P2P issues – are we able to leverage the full bandwidth of the utilized SSD of 6.9 GB/s for read and 6.24 GB/s for write transfers. We plan to extend this work with support for multiple SSDs.

3 TaPaSCo on Versal

Each FPGA device has its own peculiarities, as even within the same generation pin out, available peripherals and resources differ. Supporting a new device in TaPaSCo requires manual work to refine the overall structure of our shell to match these peculiarities. However, adding support for the first device of a new generation is especially challenging. Vendor-provided infrastructure IPs, e.g., for PCIe connectivity or memory controller, often change completely and have to be substituted. The most fundamental changes have been caused by shifting from AMD's UltraScale+ to their Versal generation of FPGAs.

In comparison to previous generations, Versal FPGAs provide important infrastructure components as *hard-wired* IPs. While this allows to maximize the PL space provided for user PEs, it also implies a fundamental restructuring of the TaPaSCo infrastructure shell, as shown in Figure 3. We leverage the hard-wired NoC and DDR controller to provide access to off-chip memory to PEs, and use the QDMA engine for PCIe traffic, replacing the custom BlueDMA engine we

used on previous platforms. 100G Ethernet is enabled using the AMD-provided MRMAC block.

In addition to copy-based data management, we introduced DMA streaming, which bypasses off-chip DRAM completely and streams data directly to and from user PEs. Furthermore, we support the Versal AI Engines (AIEs) by including a pre-compiled *libadf.a* graph in the bitstream and providing means to efficiently connect streams between AIEs and user PEs. Our implementation of a simple 4-layer neural network on the AIEs and achieved comparable or even better performance in combination with DMA streaming and 100G than an NVIDIA A100 GPU [8].

4 Outlook: TaPaSCo V80 Support

The AMD Alveo V80 has the potential to become the next de facto standard FPGA in academia. It doubles the capabilities of its predecessors in terms of FPGA resources as well as PCIe, HBM, and Ethernet bandwidth. However, even more than one year after release, AVED [1] lacks important hardware features, such as 200G Ethernet. In addition, the software stack for host applications previously offered by XRT has been discontinued. Hence, users which relied on the Vitis design flow and XRT for their hardware design and host software are forced to start from scratch when moving from a previous Alveo card to the V80. By integrating V80 support into TaPaSCo, we enable the user using the exact same PE and corresponding host software on previous supported Alveo cards, e.g. the Alveo U280, as well as the Alveo V80, simplifying the migration significantly.

We already have a TaPaSCo prototype running on the V80, but we currently face the challenge of *fully* exploiting the enhanced PCIe bandwidth and networking speeds (200G on the V80, 400G on other Versal devices). Specifically, this involves reworking our DMA subsystem to better utilize the hard-wired QDMA engine. While a single DMA command queue was sufficient to saturate the available PCIe bandwidth on previous Versal device, we now need to use multiple queues. To unlock the full hardware performance in the developer-friendly TaPaSCo environment, we plan to leverage more of the hard-wired on-chip infrastructure, and switch to the use of on-chip HBM as *primary* memory. This will unlock higher memory bandwidth for all PEs by default. For highly optimized accelerators, we plan to offer means allowing the user to specify exactly the desired mapping of PE interfaces to HBM ports, since this is important to fully exploit the bandwidth of HBM.

5 Projects using TaPaSCo

In our group at TU Darmstadt, we have used TaPaSCo for many projects in the past. The TaPaSCo RISCV project [9], already mentioned in Section 2, includes a selection of embedded RISC-V processors, such as VexRISCV or CVA5/6, for fast prototyping and benchmarking on an FPGA. In neoDBMS [4], TaPaSCo is used to prototype a hardware-accelerated non-volatile memory (NVM) based database management system (DBMS). PEs on the FPGA have low latency, high bandwidth access to the NVM, which enables optimization of many database operations. Lastly, RAVEN [7] offers hybrid simulation of SystemC by mapping and emulating parts of the simulation on the FPGA. This project has been commercialized by our industry partner.

6 Conclusion

TaPaSCo provides an easy to use FPGA shell for an ever growing number of devices and peripherals paired with a simple-to-use and platform-independent software API. Even after 15 years, we continue to improve TaPaSCo further and keep up with new platforms, such as the Alveo V80. TaPaSCo is available as open-source [6], with a low barrier of entry due to a growing collection of sample applications [5].

Acknowledgments

This research has been partially funded in multiple projects by the German Federal Ministry for Research, Technology and Space (BMFTR). The authors would like to thank AMD for supporting their work by donations of hard- and software.

References

[1] AMD. 2025. AMD AVED. https://xilinx.github.io/AVED/latest/index.html.

[2] AMD. 2026. AMD Vitis Runtime Library (XRT). https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis/vitis-runtime-library.html.

[3] AMD. 2026. AMD Vitis Unified Software Platform. https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis.html.

[4] Arthur Bernhardt, Sajjad Tamimi, Florian Stock, Andreas Koch, and Ilia Petrov. 2025. Update NDP: On Offloading Modifications to Smart Storage with Transactional Guarantees in Near-Data Processing DBMS. In *ACM Transactions on Database Systems (TODS)*. Association for Computing Machinery. https://doi.org/10.1145/3774753

[5] Embedded Systems and Applications Group, TU Darmstadt. 2026. TaPaSCo-Examples on Github. https://github.com/esa-tu-darmstadt/tapasco-examples.

[6] Embedded Systems and Applications Group, TU Darmstadt. 2026. TaPaSCo on Github. https://github.com/esa-tu-darmstadt/tapasco.

[7] MINRES Technologies GmbH. 2026. RAVEN. https://minres.com/raven.

[8] Carsten Heinz, Torben Kalkhof, Yannick Lavan, and Andreas Koch. 2024. TaPaSCo-AIE: An Open-Source Framework for Streaming-Based Heterogeneous Acceleration Using AMD AI Engines. In *2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 155–161. https://doi.org/10.1109/IPDPSW63119.2024.00041

[9] Carsten Heinz, Yannick Lavan, Jaco Hofmann, and Andreas Koch. 2019. A Catalog and In-Hardware Evaluation of Open-Source Drop-In Compatible RISC-V Softcore Processors. In *IEEE Proc. International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE. https://doi.org/10.1109/ReConFig48160.2019.8994796

[10] Torben Kalkhof and Andreas Koch. 2021. Efficient Physical Page Migrations in Shared Virtual Memory Reconfigurable Computing Systems. In *2021 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, Auckland, New Zealand, 1–10. https://doi.org/10.1109/ICFPT52863.2021.9609831

[11] Torben Kalkhof and Andreas Koch. 2022. Direct Device-to-Device Physical Page Migrations in Multi-FPGA Shared Virtual Memory Systems. In *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*. 225–234. https://doi.org/10.1109/FPL57034.2022.00043

[12] David Volz, Torben Kalkhof, and Andreas Koch. 2025. SNAcc: An Open-Source Framework for Streaming-based Network-to-Storage Accelerators. In *Proceedings of the SC '25 Workshops of the International Conference for High Performance Computing, Netowrking, Storage and Analysis (SC Workshops '25)*. 633–641. https://doi.org/10.1145/3731599.3767412