



COMP4DRONES



Gianluca Bellocchi, Alessandro Capotondi and Andrea Marongiu
UNIMORE

A RISC-V-based FPGA Overlay to
Simplify Accelerator Deployment
for Unmanned Vehicles

COMP4DRONES will provide a **framework** of key enabling technologies for **safe and autonomous drones** that will leverage their **customization and modularity** for civilian services



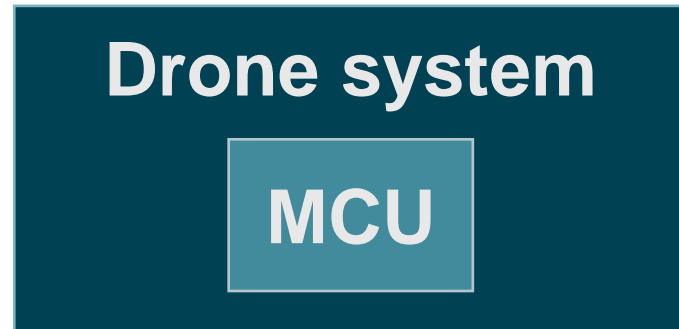
COMP4DRONES

ECSEL JU GA No 826610

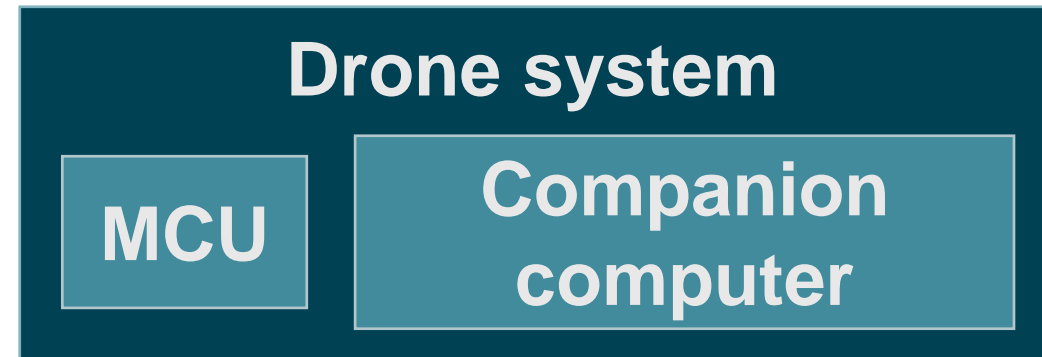
Website: comp4drones.eu

Introduction

Transition to a new paradigm

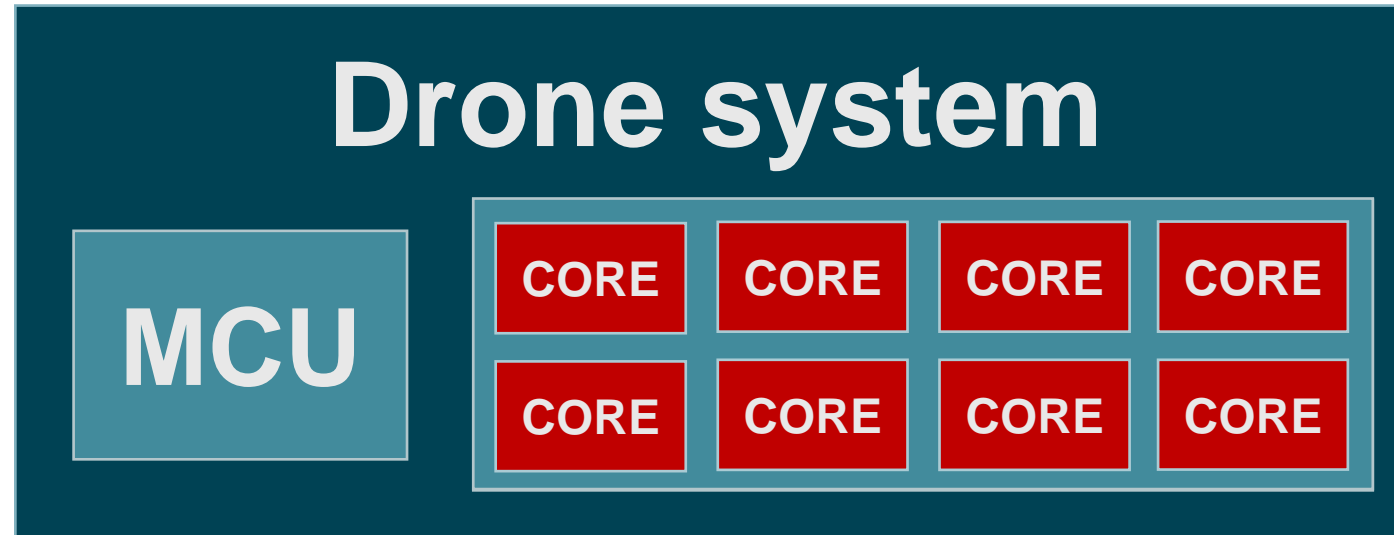


- The “*classic*” set-up comprises a **micro-controller unit (MCU)** that is used for control and actuation



- *Current paradygm* envisions coupling a **MCU** with a **companion computer**
- **Heterogeneous solutions** (Nvidia Tegra TX2, Xilinx Zynq US+, ..) are increasingly used

Transition to a new paradigm




Companion computer typically exhibits:

- Low power envelopes
- High performance



AI workloads become feasible for UAVs

Heterogeneous platform

- There exist many *commercially available platforms* (Nvidia Tegra TX2, Xilinx Zynq US+, ..)
- Our solution  **FPGA-based HeSoC**
- **Why FPGA?**
 - Hardware flexibility
 - Tight power envelopes
 - Design is more predicatable
 - Demonstrated to be highly performant with AI workloads
 - GPUs are closed systems in HW and SW: difficult to design custom extensions




Open-Hardware is the solution!

Using a “closed” platform would not permit to do this.

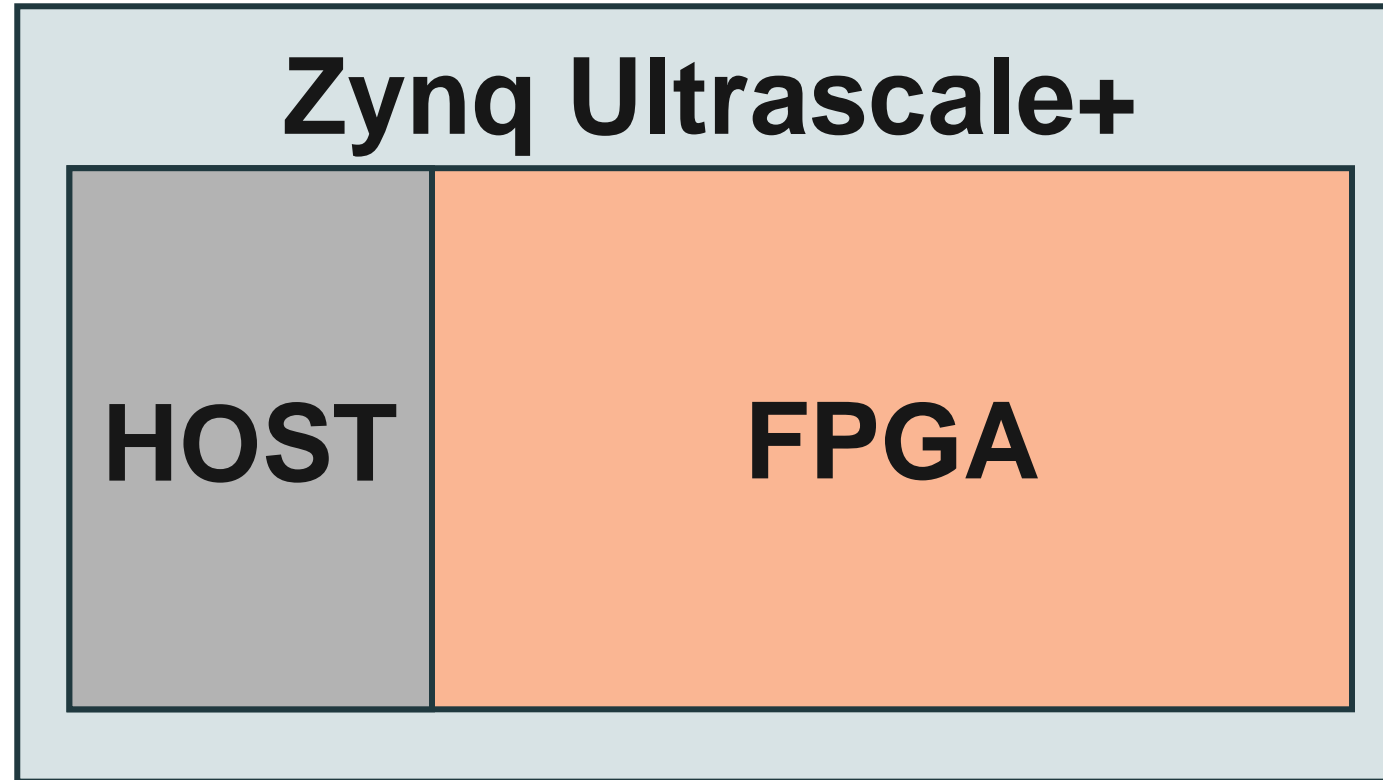


Heterogeneous platform



- There exist many *commercially available platforms* (Nvidia Tegra TX2, Xilinx Zynq US+, ..)
- Our solution  **FPGA-based HeSoC**
- **On the other side..**
 - ***Hard design process*** - RTL, low-level design expertise is often needed
 - ***Long compilation time*** - Productivity is negatively impacted
 - Existing tools are ***not mature enough to permit rapid swapping and modifying of design components***

Xilinx Zynq Ultrascale+



Xilinx Zynq Ultrascale+

- The HeSoC architecture comprises a large set of components

- **Host processor**

- High-performance ARM processors
- Standard deployment of software legacies
 - ❖ **Linux** - Real-Time distros are available
 - ❖ **ROS** – Ideal for robotic applications



No need to replicate these from scratch!

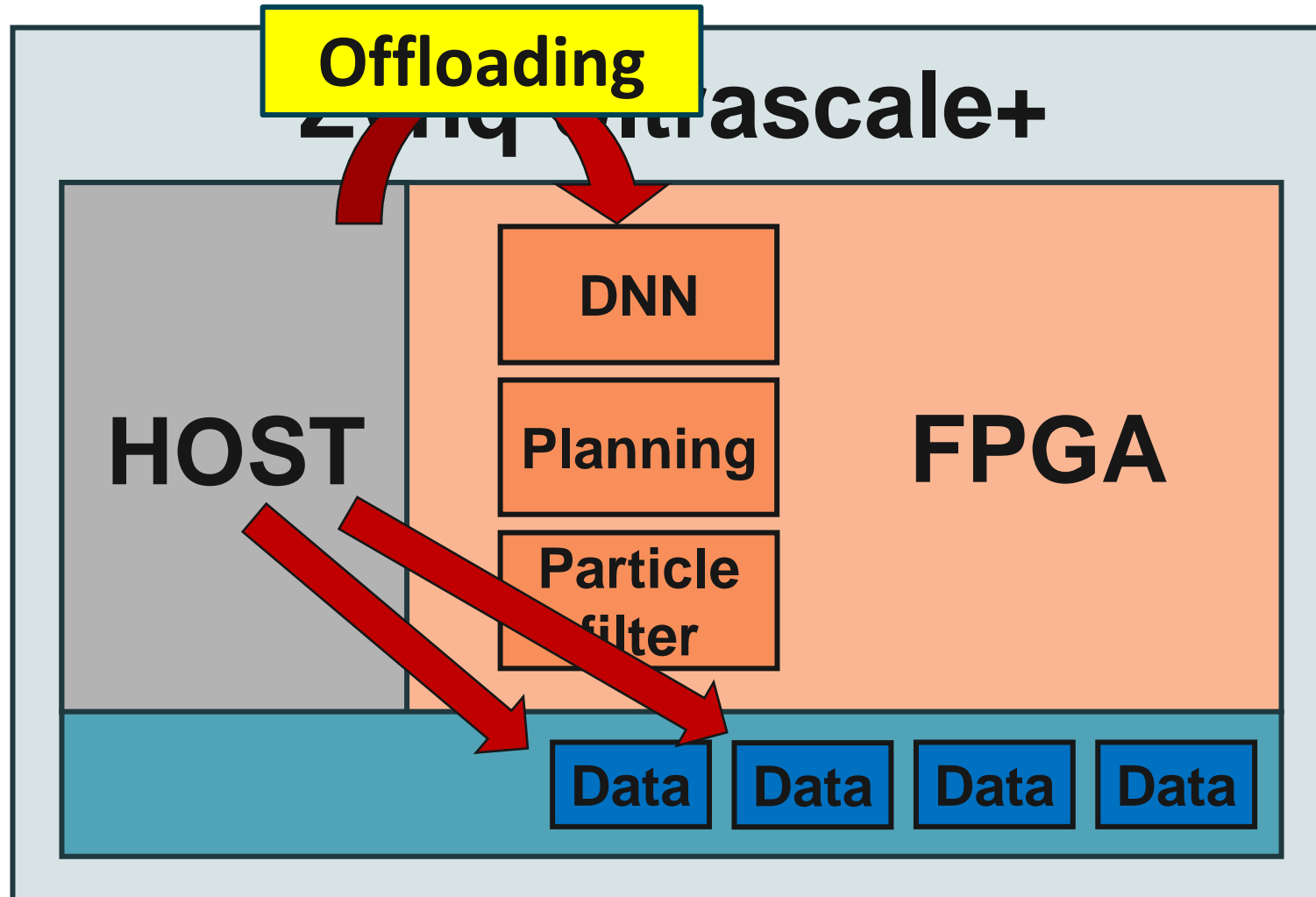
- **Programmable logic**

- **Hardware accelerators** can enhance both *performance* (real-time requirements) and *energy saving* (high power peaks but low execution times)



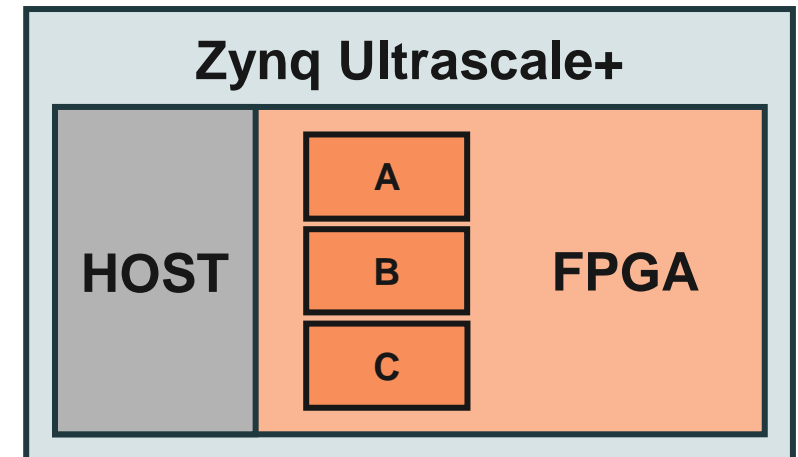
Need to ease the pain of integrating and deploying these IPs

Traditional approach



Traditional approach

- **Performance is limited** because of:
 - Accelerators controlling and data transferring are managed by the HOST (not local and highly expensive)
- **Rapid swapping and modifying of coarse-grained blocks**
 - Still unfeasible
 - Automated design tools (e.g. Xilinx SDSoC) lack the required maturity to efficiently and easily tackle system-level integration of large HW/SW design blocks



Motivation

- **Performance is limited** because of:
 - Accelerators controlling and data transferring are managed by the HOST (not local and highly expensive)
- **Rapid swapping and modifying of coarse-grained blocks**
 - Still unfeasible
 - Automated design tools (e.g. Xilinx SDSoC) lack the required maturity to efficiently and easily tackle system-level integration of large HW/SW design blocks

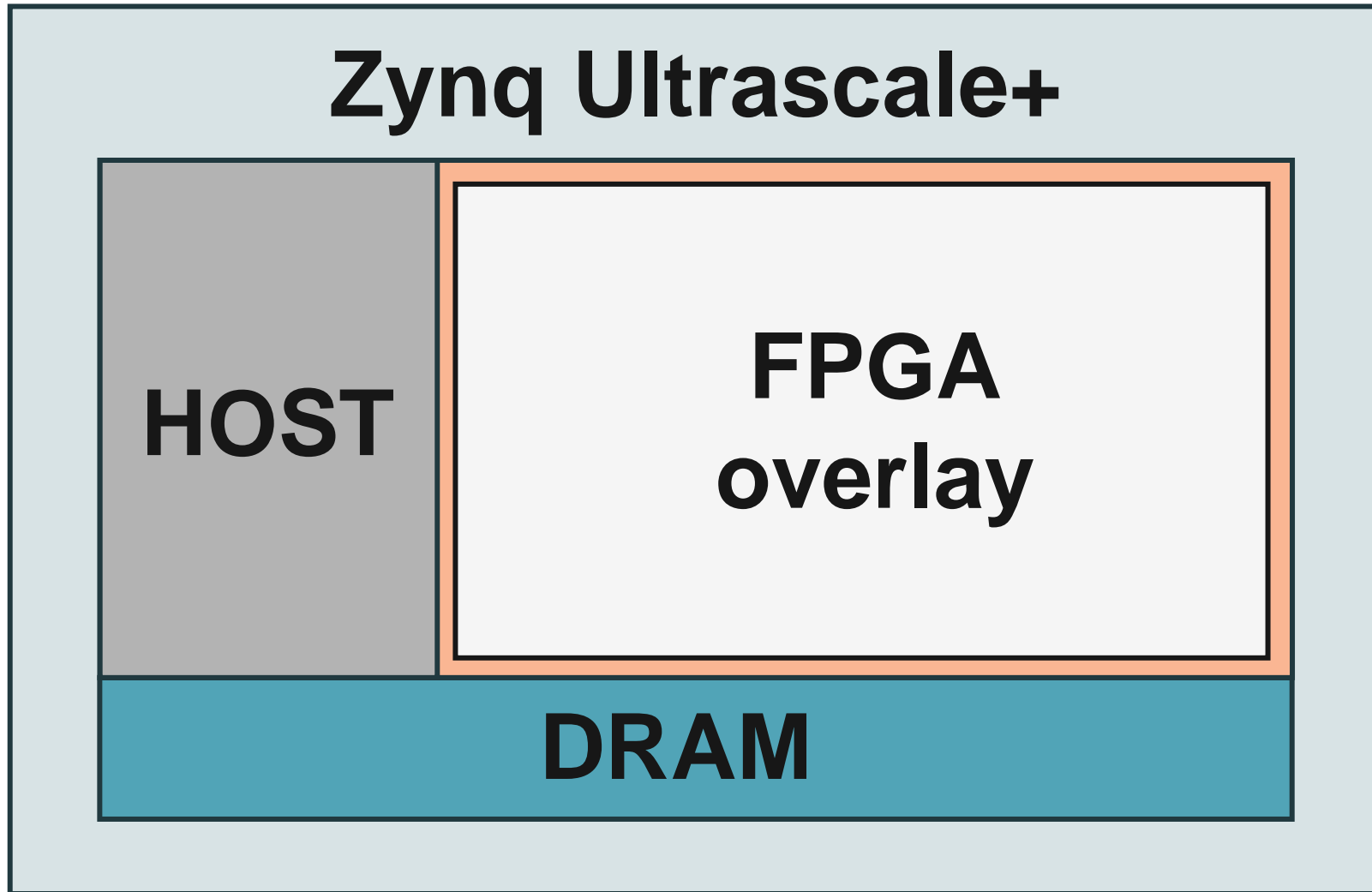


What about when the design includes many of them?

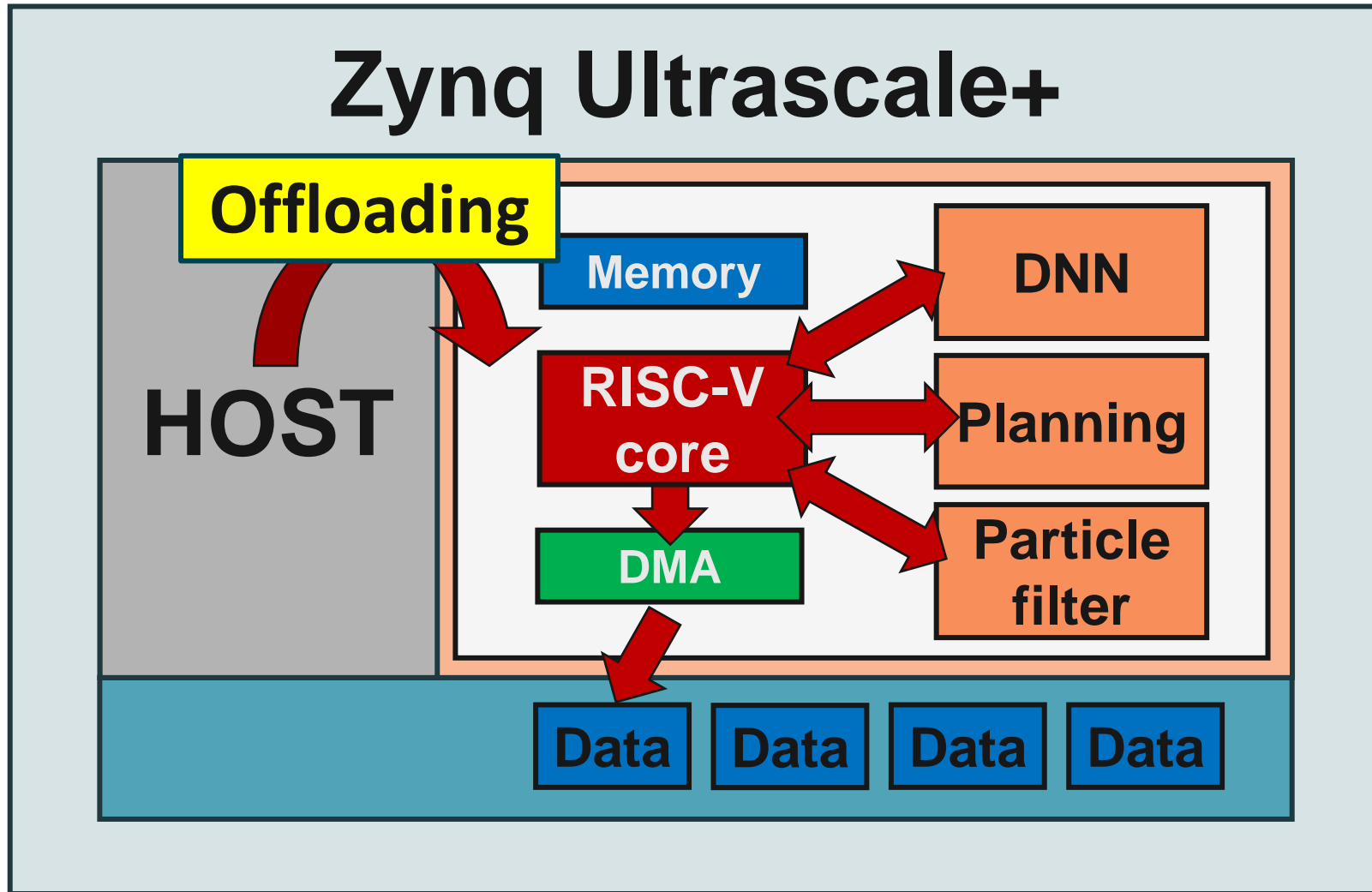


Essential in modern UAV systems

Our proposal



Our proposal



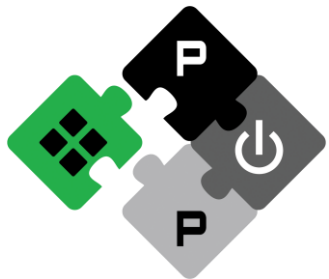
Starting point - HERO

▪ PULP architecture

- Parallel Ultra Low Power
- Open and Scalable HW/SW research and development platform
- Cluster-based architecture
- RISC-V ISA compliant

▪ HERO

- FPGA emulation of heterogeneous and massively parallel PULP systems
- Instantiable with COTS FPGA-based heterogeneous SoCs



Kurth, A., Capotondi, A., Vogel, P., Benini, L., & Marongiu, A. (2018, November). HERO: An open-source research platform for HW/SW exploration of heterogeneous manycore systems.





Overlay architecture

Overlay architecture

- **What is it?**

- Hardware abstraction layer
- Overlays the original FPGA fabric  Hides hardware details

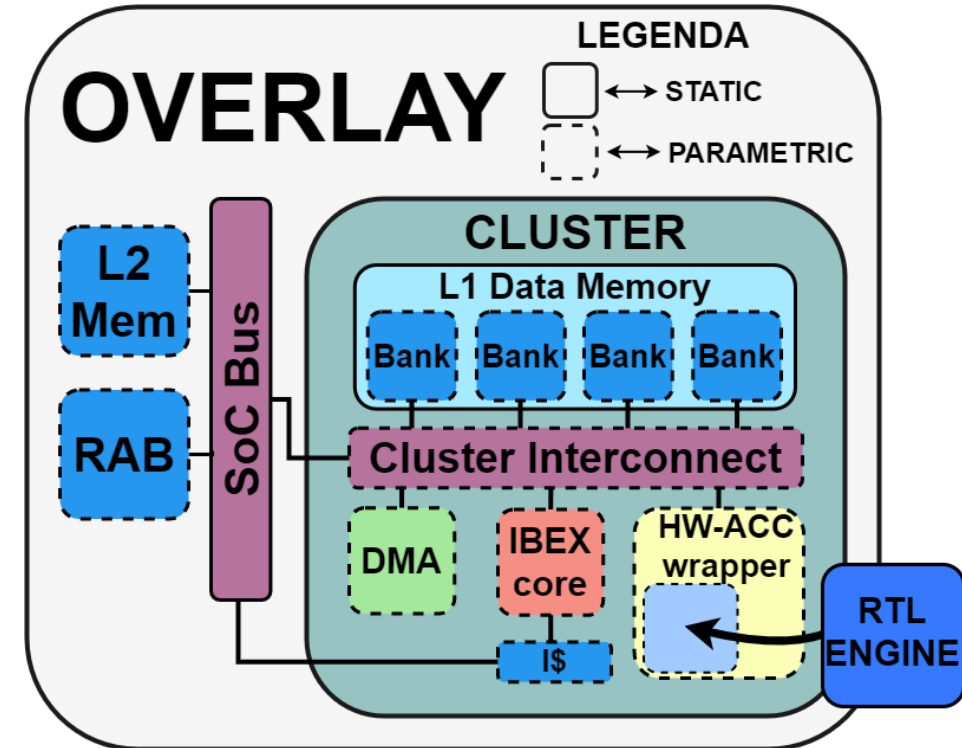
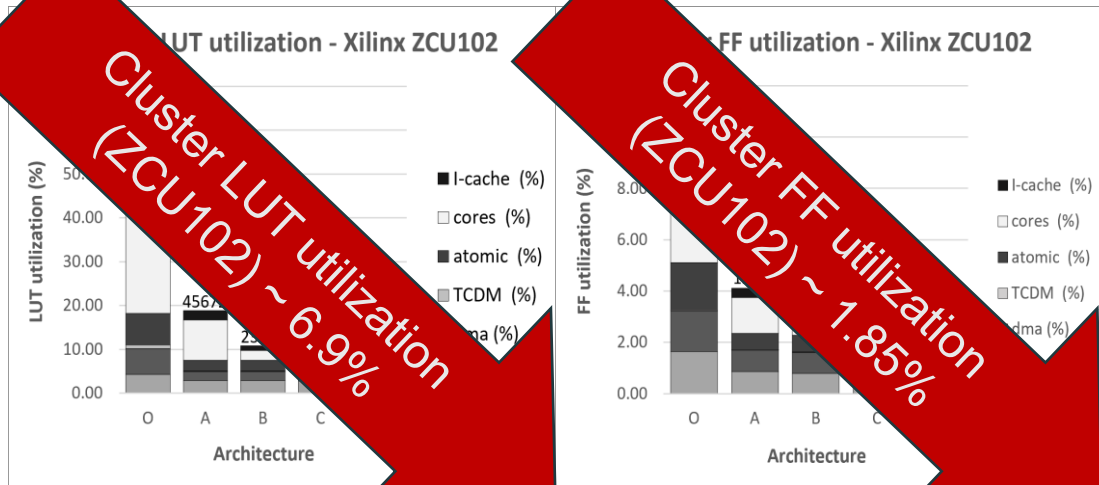
- **Features:**

- Coarse-grained  Rapid swapping of architectural blocks
- Avoid FPGA design flow  Improved design productivity
- Programmable via standard APIs for heterogeneous compute platforms

Our FPGA overlay

- Small resource overhead

- Extensive *characterization of logic resource* (LUT, flip-flop, BRAM, DSP) on different architectural solutions



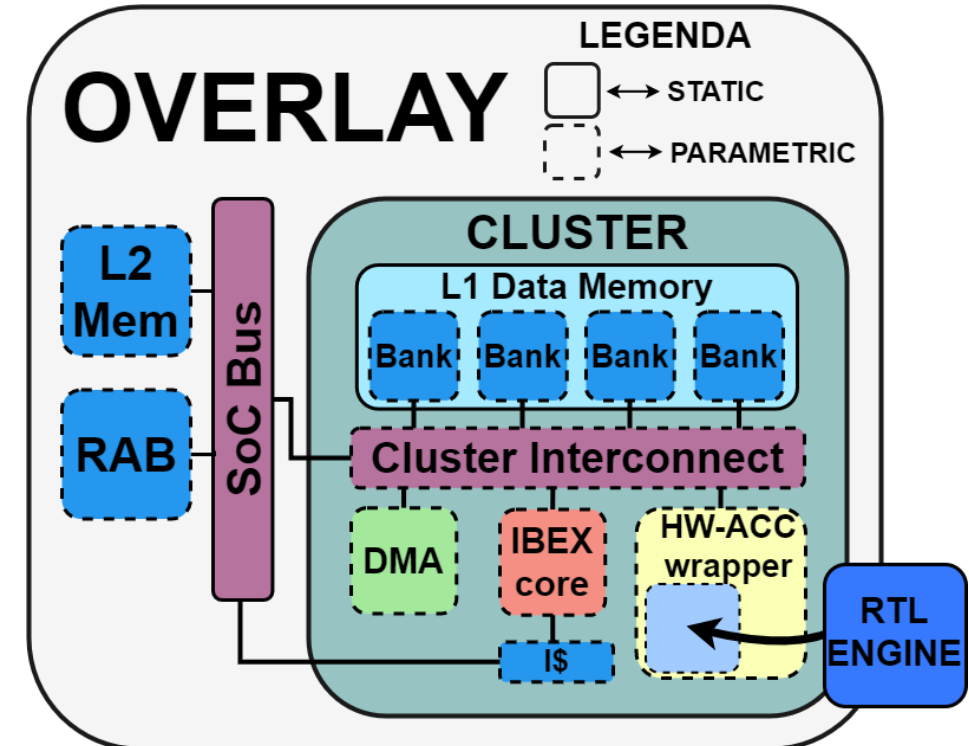
Our FPGA overlay

- **Acceleration**

- Hardware accelerators populate the cluster
- Plug-and-play integration with the aid of a parametric wrapper



- **Soft core(-s)**

- Orchestration of hardware accelerators operation



Accelerator integration methodology

Wrapper-based methodology

- A wrapper supports hardware acceleration encapsulating:
 - **Communication interface**  *Streamer*
 - **Control interface**  *Controller*
- **Wrapper specialization**
 - Simplified via template instantiation
 - User needs to identify the key-features of its accelerator design

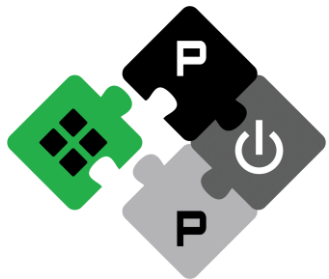
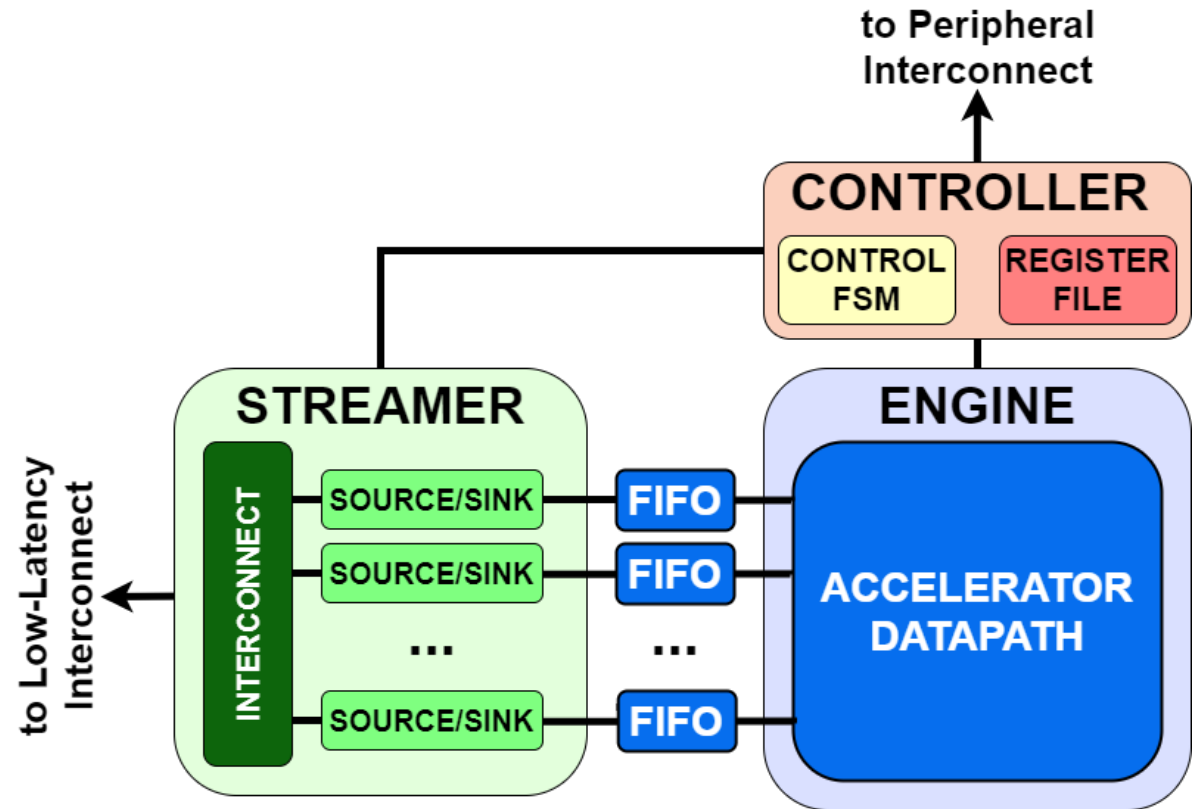
Starting point – HWPE

- **Streamer**

- Specialized DMA controller that transforms streams into memory accesses

- **Controller**

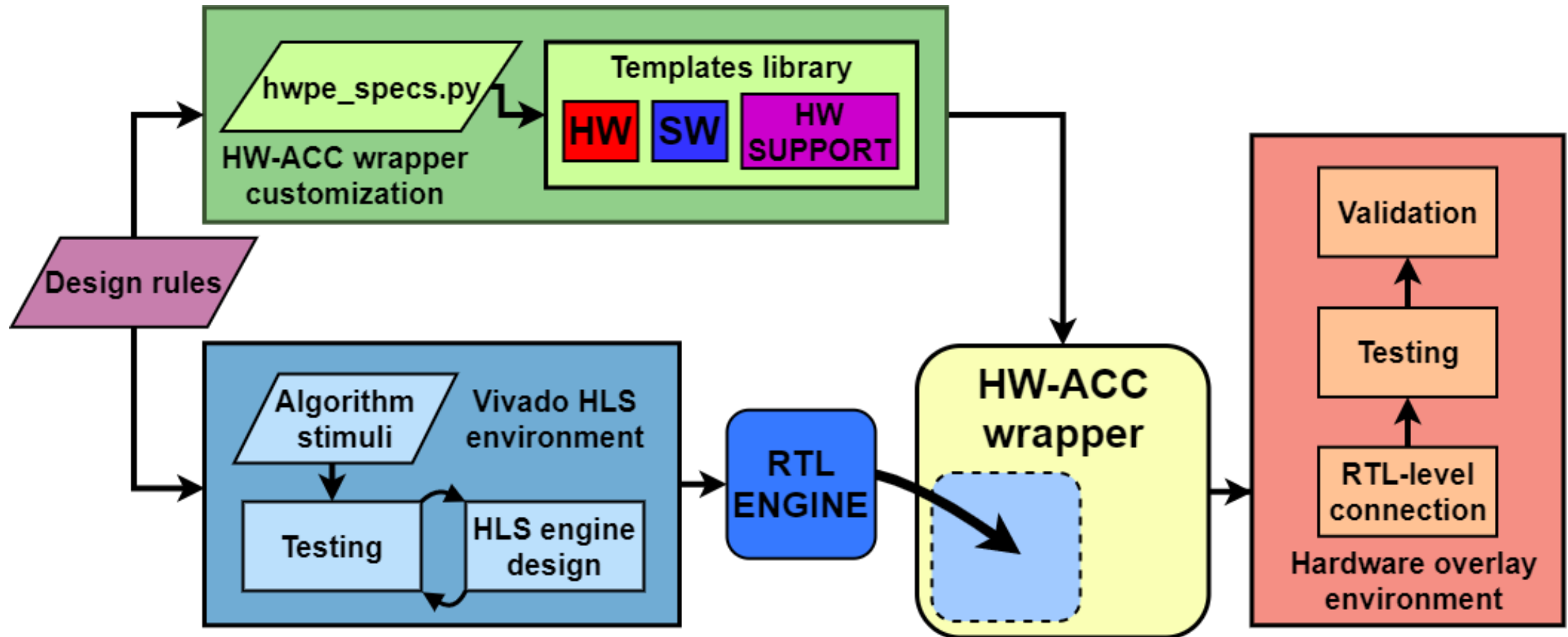
- Register file to host runtime parameters
- FSM for coarse-grained control/(re)-configuration



Final result - Wrapper specialization

1. The user needs to identify the key features of its accelerator design
2. Compile a wrapper spec file
 - ..nothing more than a simple Python class!
 - Describes how the application-dependent components of the wrapper have to be implemented
3. Specs are propagated throughout the wrapper template library
4. Generation of:
 - Hardware wrapper
 - Software for accelerator runtime calls

Accelerator integration methodology



Conclusions and Future work

Conclusions

1. Exploration of a hardware overlay solution to simplify the adoption of FPGA-based HeSoCs
2. Plug-and-play HW/SW integration of hardware accelerators based on a specializable wrapper IP
3. Use of local proxy core(-s) to orchestrate the accelerator operations
4. Ease the development process of optimized accelerator control routines

Future work

1. Benchmarking is essential to:
 - validate the design
 - verify the design entries
2. Extend the wrapper specification tool capabilities
3. Acceleration and deployment of UAV workloads (perception, planning, control)

Thanks for your attention!

