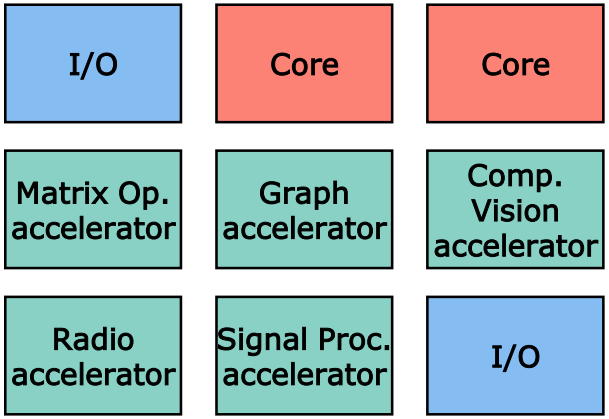
Open-Source Hardware for Heterogeneous Computing with ESP and RISC-V

Luca P. Carloni

Columbia University
In the City of New York

Spring 2022 RISC-V Week
Paris - May 3, 2022

The Age of Heterogeneous Computing



- **State-of-the-art SoC architectures integrate increasingly diverse sets of components**
  - different CPUs, GPUs, hardware accelerators, memory hierarchies, I/O peripherals, sensors, reconfigurable engines, analog blocks…
- **The migration towards heterogeneous SoC architectures will accelerate, across almost all computing domains**
  - IoT devices, mobile devices, embedded systems, automotive electronics, avionics, data centers and even supercomputers
- **The set of heterogeneous SoCs in production in any given year will be itself heterogeneous!**
  - no single SoC architecture will dominate all the markets!

©Luca Carloni

Heterogeneity Increases Design Complexity

- Heterogeneous architectures produce higher energy-efficient performance, but make more difficult the tasks of design, verification and programming
  - at design time, diminished regularity in the system structure, chip layout
  - at runtime, more complex hardware/software and management of shared resources
- With each SoC generation, the addition of new capabilities is increasingly limited by engineering effort and team sizes
- The biggest challenges are (and will increasingly be) found in the complexity of system integration

©Luca Carloni

Open-Source Hardware (OSH)

- An opportunity to reenergize the innovation in the semiconductor and electronic design automation industries
- The OSH community is gaining momentum
  - many diverse contributions from both academia and industry
  - multi-institution organizations
  - government programs

Image Sources:
https://chipsalliance.org/
https://github.com/nvdla
https://www.openhwgroup.org/
https://parallel.princeton.edu/openpiton/
https://pulp-platform.org/
https://riscv.org/

©Luca Carloni

The Open Challenge of Open-Source Hardware

- To date, however, most OSH projects are focused on the development of individual SoC components, such as a processor core or an accelerator
- This leaves open a critical challenge:

  How can we realize a complete SoC for a given target application domain by efficiently reusing and combining a variety of independently developed, heterogeneous, OSH components, especially if these components are designed by separate organizations for separate purposes?

©Luca Carloni

The Concept of Platform

- Innovation in SoC architectures and their design methodologies is needed to promote design reuse and collaboration
  - Architectures and methodologies must be developed together
- *Platform = architecture + methodology*
  - An SoC architecture enables design reuse when it simplifies the integration of many components that are independently developed
  - An SoC methodology enables design collaboration when it allows designers to choose the preferred specification languages and design flows for the various components
- An effective combination of architecture and methodology is a platform that maximizes the potential of open-source hardware
  - by scaling-up the number of components that can be integrated in an SoC and by enhancing the productivity of the designers who develop and use them

©Luca Carloni

ESP : An Open-Source Platform for SoC Design

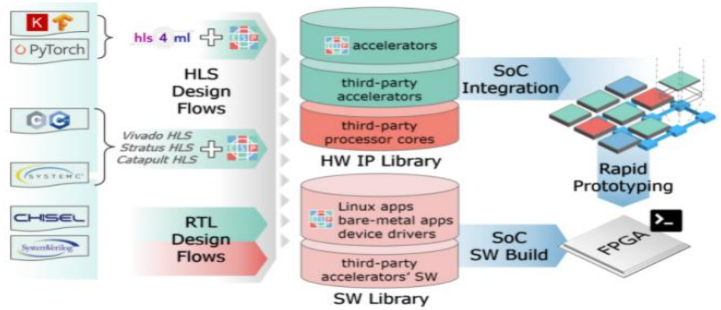
Home Release Resources News Press Team Contact

ESP
the open-source SoC platform

www.esp.cs.columbia.edu

The ESP Vision

ESP is an open-source research platform for heterogeneous system-on-chip design that combines a scalable tile-based architecture and a flexible system-level design methodology.

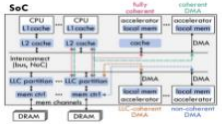


ESP provides three accelerator flows: RTL, high-level synthesis (HLS), machine learning frameworks. All three design flows converge to the ESP automated SoC integration flow that generates the necessary hardware and software interfaces to rapidly enable full-system prototyping on FPGA.

Overview

- Architecture

Latest Posts



Paper accepted at MICRO 2021

Our paper *"Cohmeleon: Learning-Based Orchestration of Accelerator Coherence in Heterogeneous SoCs"* will be presented in October at MICRO 2021.

Read more

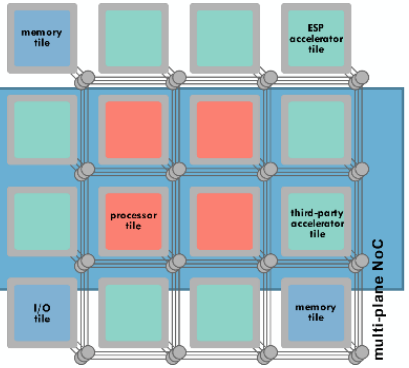
Published: Sep 14, 2021

Paper published in

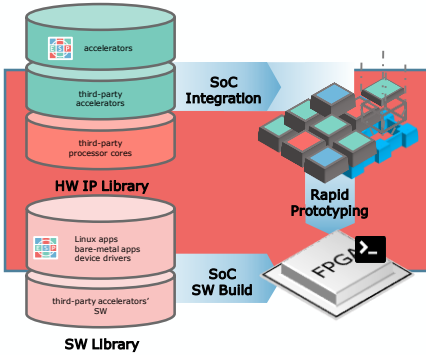
©Luca Carloni

Outline

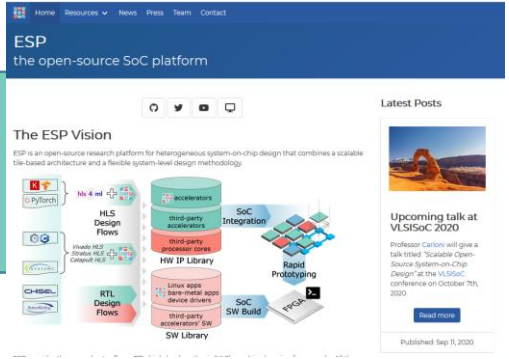
The ESP Architecture



The ESP Methodology



Research & Teaching with ESP

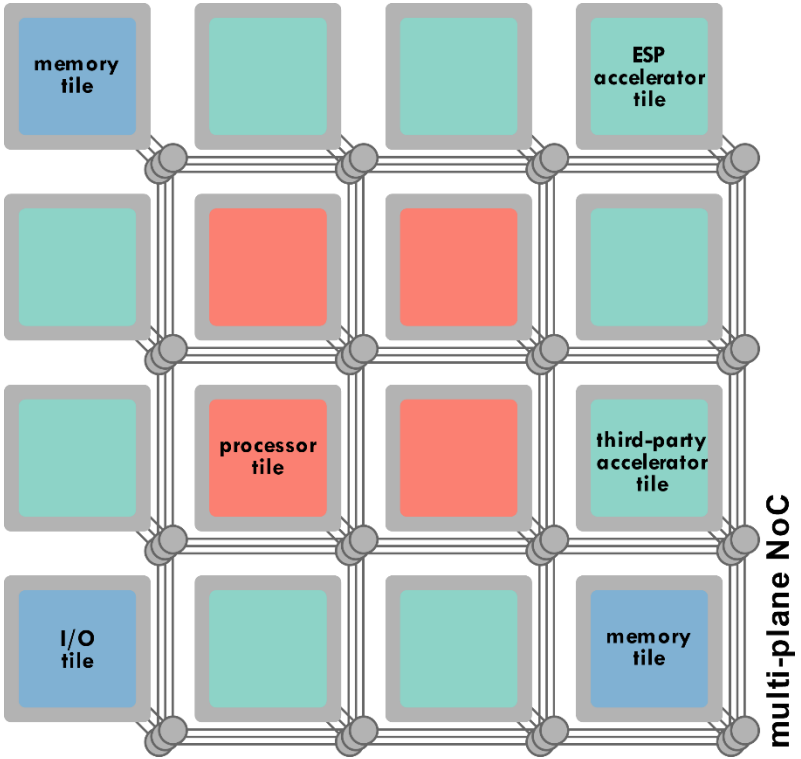


©Luca Carloni

ESP Architecture

- RISC-V Processors
- Many-Accelerator
- Distributed Memory
- Multi-Plane NoC

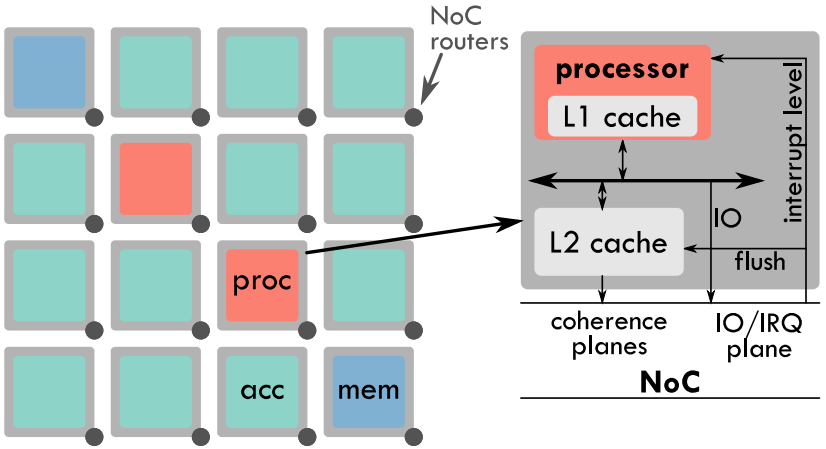
The ESP architecture implements a **distributed** system, which is **scalable**, **modular** and **heterogeneous**, giving processors and accelerators similar weight in the SoC



©Luca Carloni

ESP Architecture: Processor Tile

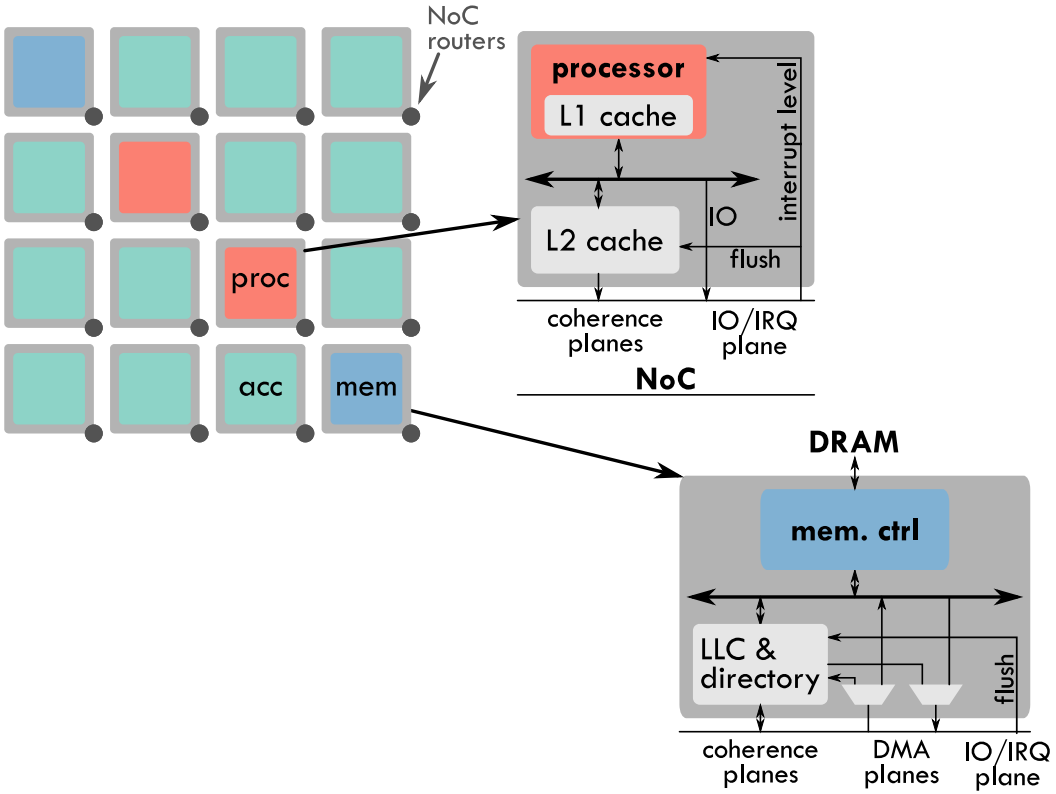
- Processor off-the-shelf
  - RISC-V CVA6-Ariane (64 bit)
    SPARC V8 Leon3 (32 bit)
  - RISC-V IBEX (32 bit)
  - L1 private cache
- L2 private cache
  - Configurable size
  - MESI protocol
- IO/IRQ channel
  - Un-cached
  - Accelerator config. registers, interrupts, flush, UART, …



©Luca Carloni

ESP Architecture: Memory Tile

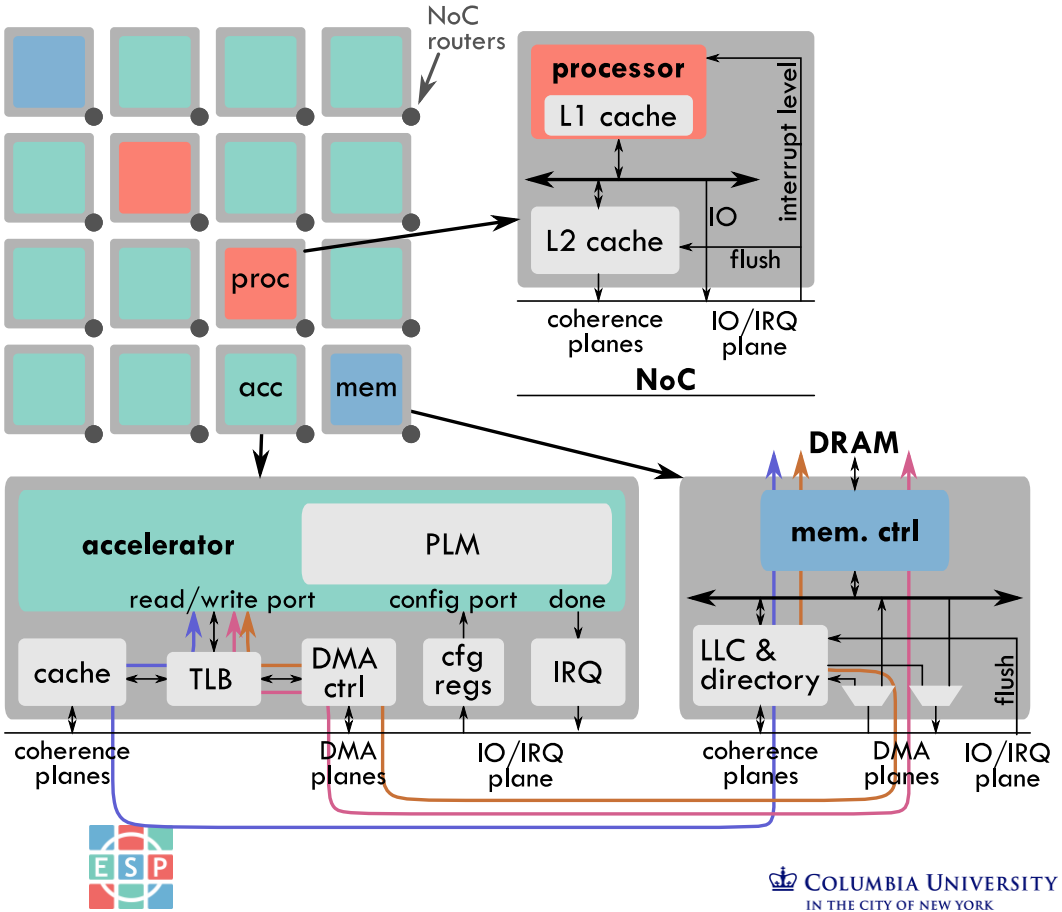
- External Memory Channel
- LLC and directory partition
  - Configurable size
  - Extended MESI protocol
  - Supports coherent-DMA for accelerators
- DMA channels
- IO/IRQ channel



©Luca Carloni

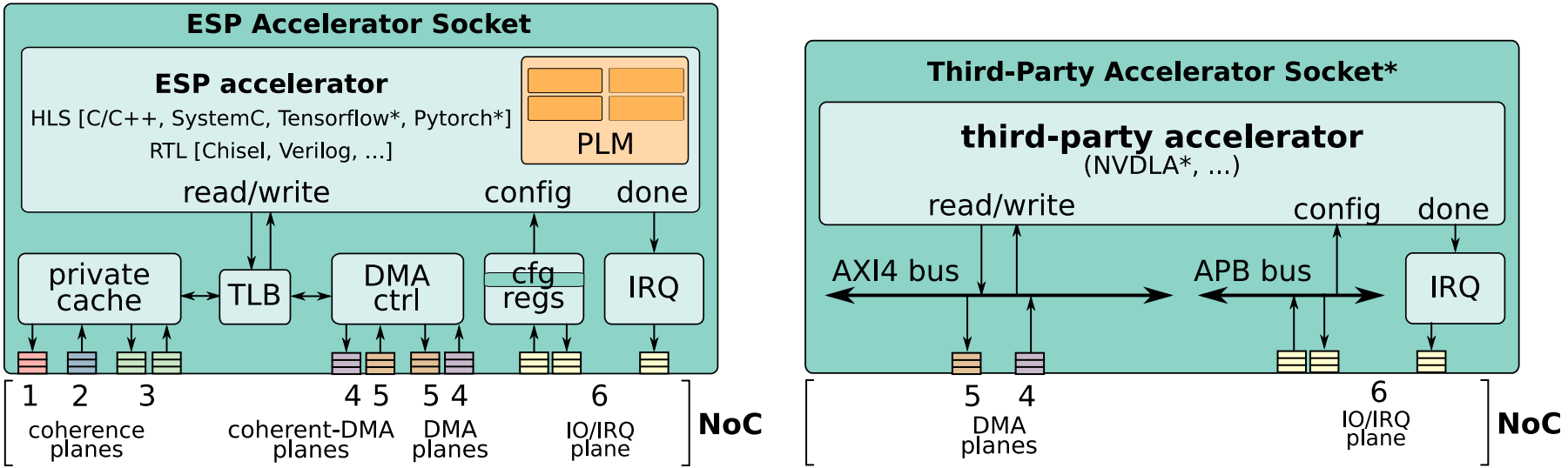
ESP Architecture: Accelerator Tile

- Accelerator Socket w/ Platform Services
  - Direct-memory-access
  - Run-time selection of coherence model:
    - Fully coherent
    - LLC coherent
    - Non coherent
  - User-defined registers
  - Distributed interrupt



©Luca Carloni

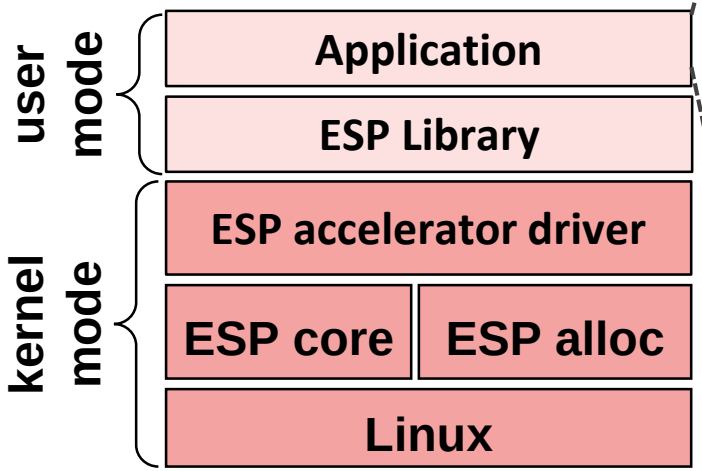
ESP Accelerator Socket



©Luca Carloni

ESP Software Socket

- **ESP accelerator API**
  - Generation of device driver and unit-test application
  - Seamless shared memory




```
/*
 * Example of existing C application with ESP
 * accelerators that replace software kernels 2, 3,
 * and 5. The cfg_k# contains buffer and the
 * accelerator configuration.
 */
{
  int *buffer = esp_alloc(size);

  for (...) {

    kernel_1(buffer,...); /* existing software  */

    esp_run(cfg_k2);      /* run accelerator(s) */
    esp_run(cfg_k3);

    kernel_4(buffer,...); /* existing software  */

    esp_run(cfg_k5);
  }

  validate(buffer);       /* existing checks    */

  esp_free();           /* memory free        */
}
```

©Luca Carloni

ESP Platform Services

Accelerator tile
- DMA
- Reconfigurable coherence
- Point-to-point
- ESP or AXI interface
- DVFS controller

Processor Tile
- Coherence
- I/O and un-cached memory
- Distributed interrupts
- DVFS controller

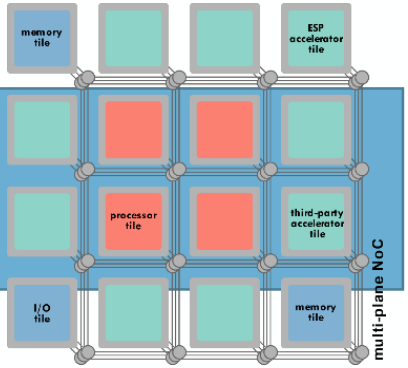
Miscellaneous Tile
- Debug interface
- Performance counters access
- Coherent DMA
- Shared peripherals (UART, ETH, …)

Memory Tile
- Independent DDR Channel
- LLC Slice
- DMA Handler

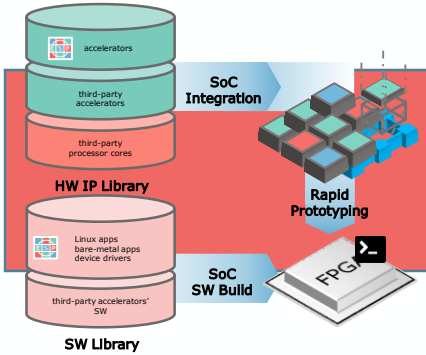
©Luca Carloni

Outline

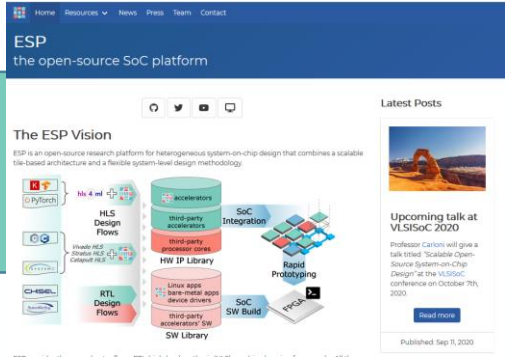
The ESP Architecture



The ESP Methodology



Research & Teaching with ESP



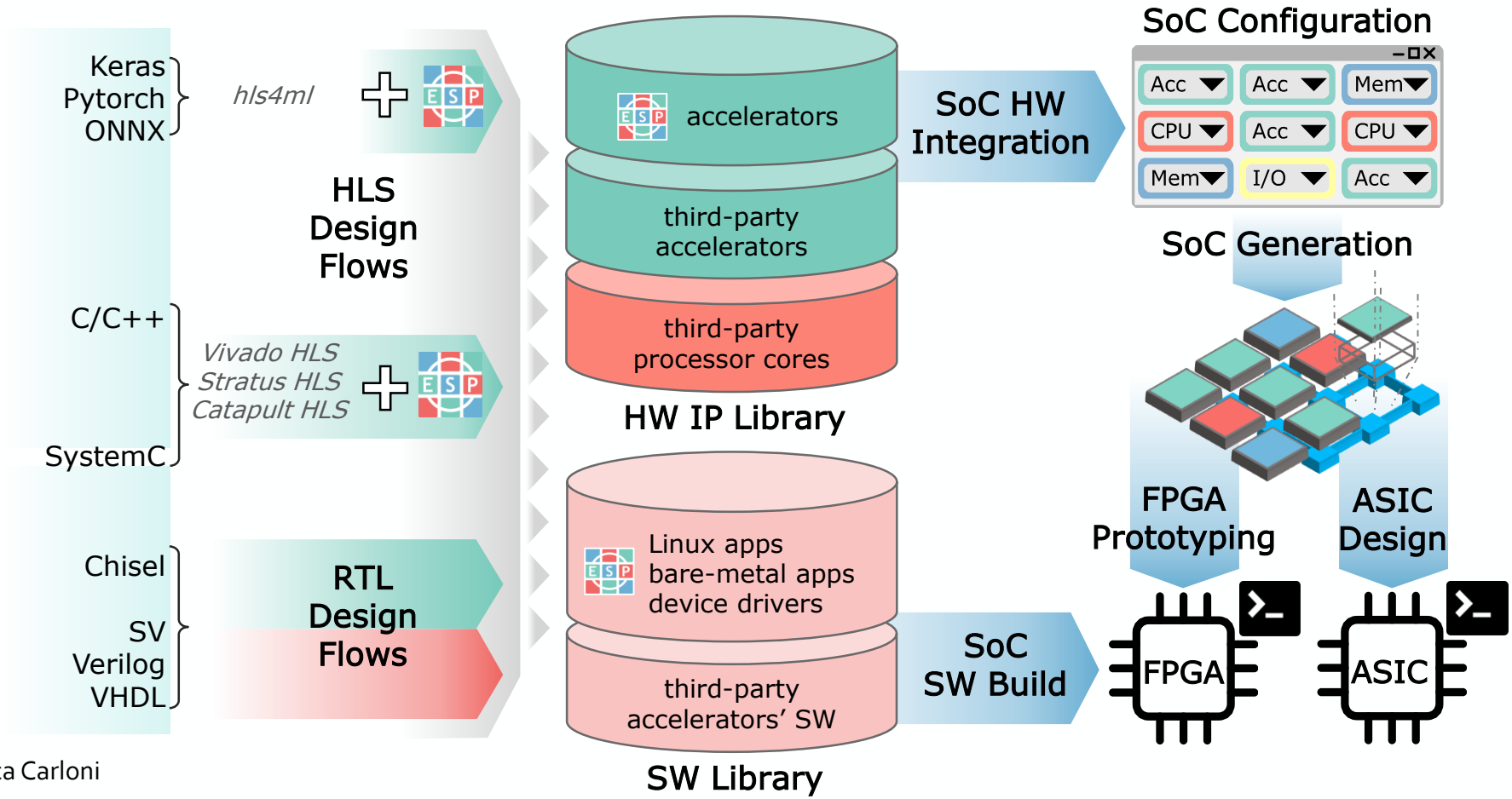
©Luca Carloni

The Pillars of the ESP Approach

- Develop platforms, not just architectures
  - A platform combines an architecture and a companion design methodology
- Move from a processor-centric to an SoC-centric perspective
  - The processor core is just one component among many others
- Raise the level of abstraction
  - Move from RTL design to domain-specific *system-level design* with high-level synthesis…
  - …but keep supporting different abstraction levels and design flows
- Promote Open-Source Hardware
  - Build libraries of reusable components
  - Support the integration of third-party IP components

©Luca Carloni

The ESP Vision: Domain Experts Can Design SoCs



©Luca Carloni

ESP Methodology In Practice

automated
interactive
manual (opt.)
manual

Accelerator Flow

Generate accelerator

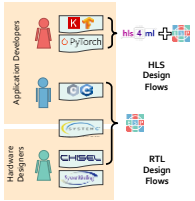
Specialize accelerator
* this step is automated
* for ML applications

Test behavior

Generate RTL

Test RTL

Optimize RTL



SoC Flow

Generate sockets

Configure RISC-V SoC

Compile bare-metal

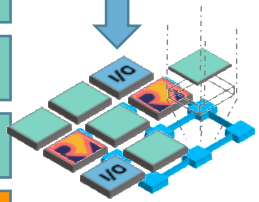
Simulate system

Implement for FGPA

Configure runtime

Compile Linux

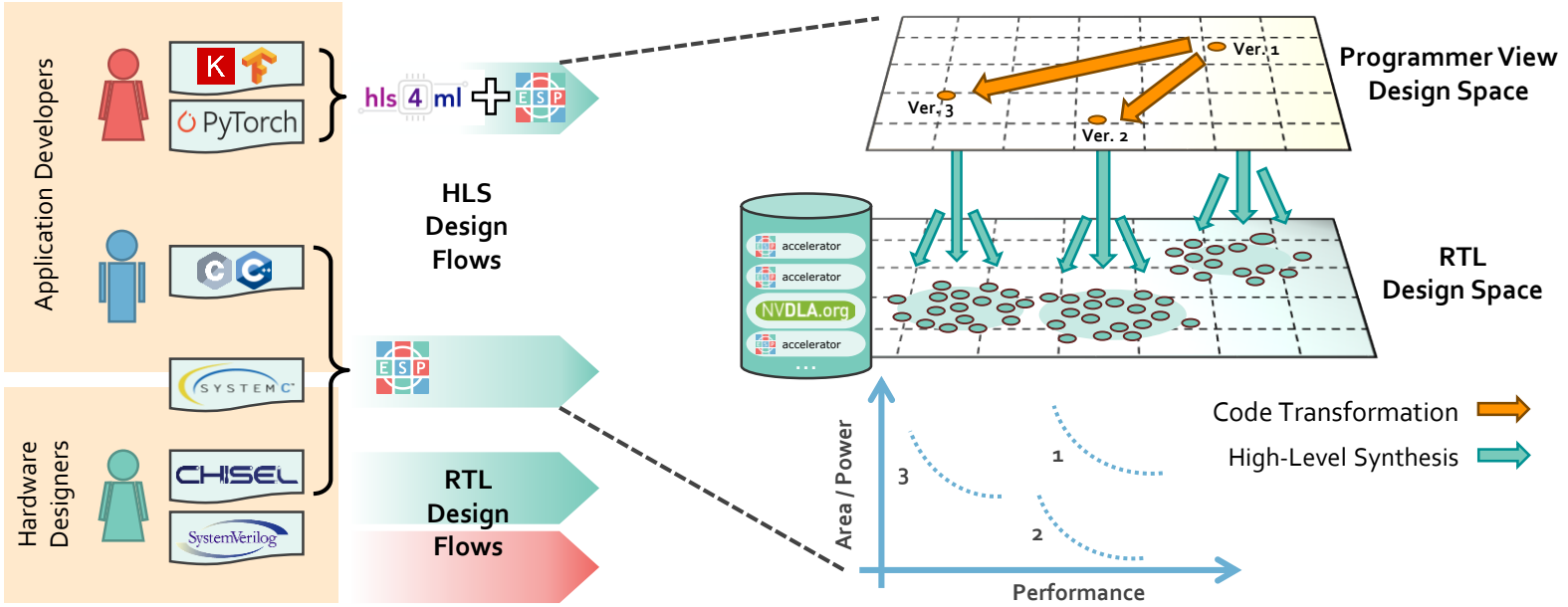
Deploy prototype



©Luca Carloni

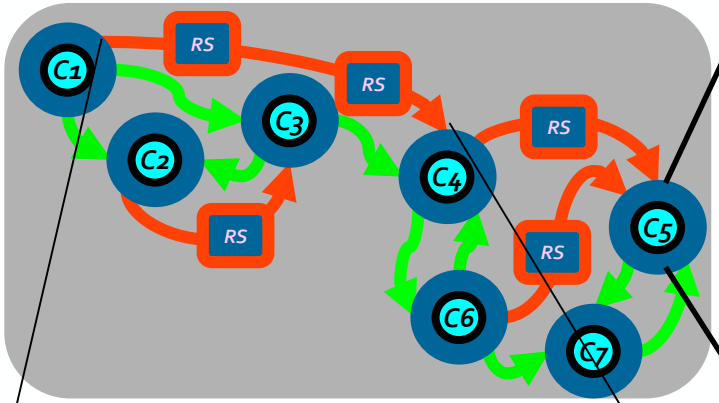
ESP Accelerator Flow

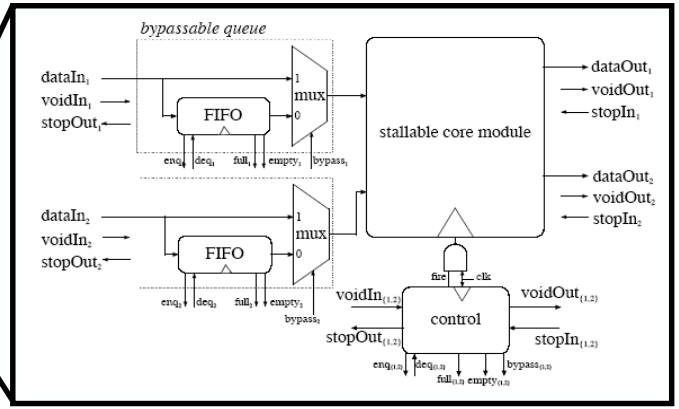
Developers focus on the high-level specification, decoupled from memory access, system communication, hardware/software interface

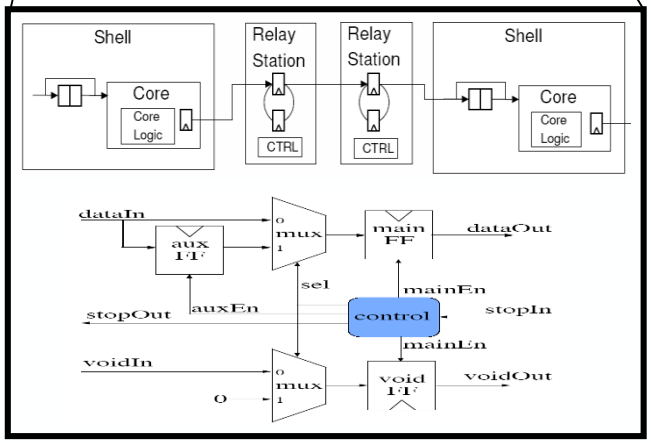


©Luca Carloni

Retrospective: Latency-Insensitive Design [Carloni et al. '99]





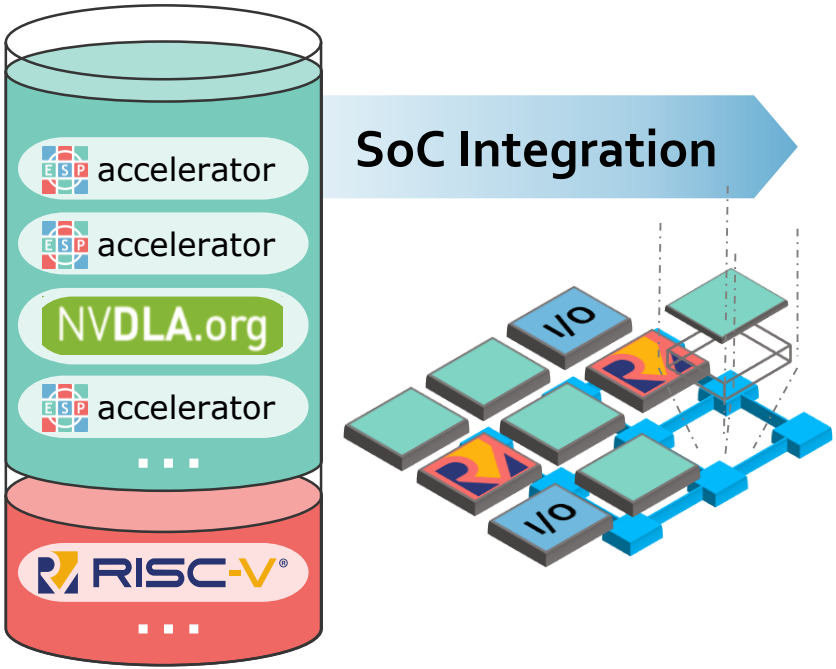


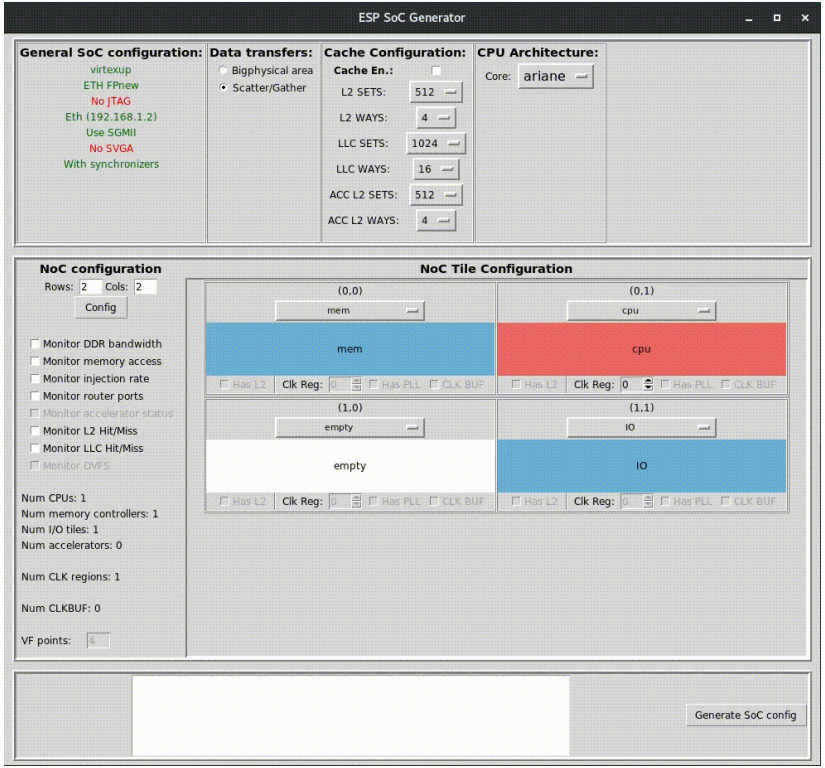
Latency-Insensitive Design

- **is the foundation for the *flexible synthesizable RTL representation***
- **anticipates the separation of computation from communication that is proper of TLM with SystemC**
  - through the introduction of the Protocols & Shell paradigm

©Luca Carloni

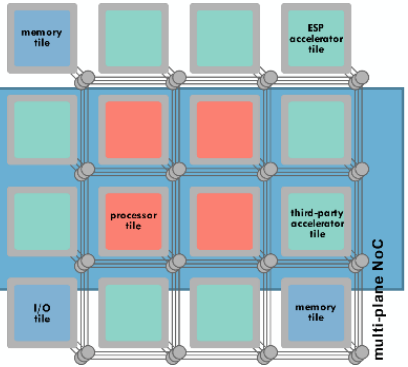
ESP Interactive Flow for SoC Integration



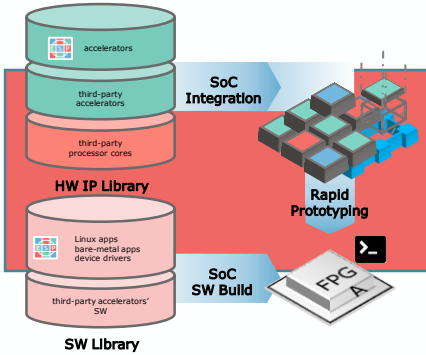


Outline

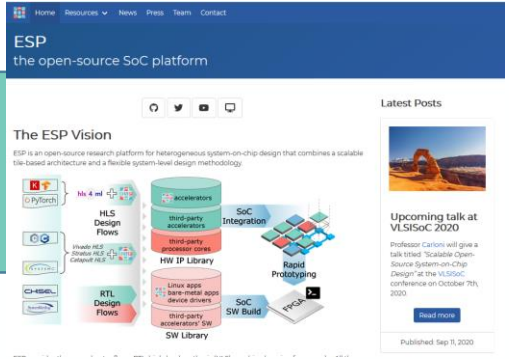
The ESP Architecture



The ESP Methodology

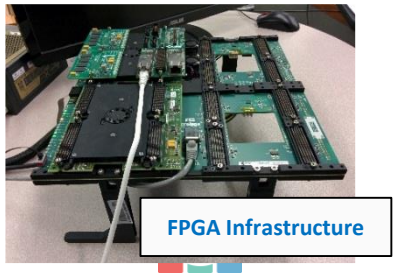


Research & Teaching with ESP



©Luca Carloni

Example of a System We Built: FPGA Prototype to Accelerate Wide-Area Motion Imagery



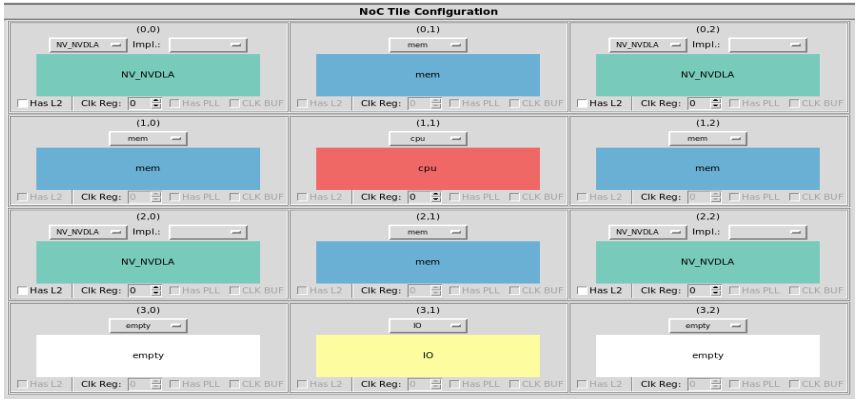
- **Design:** Complete design of WAMI-App running on an FPGA implementation of an ESP architecture
  - featuring 1 embedded processor, 12 accelerators, 1 five-plane NoC, and 2 DRAM controllers
  - SW application running on top of Linux while leveraging multi-threading library to program the accelerators and control their concurrent, pipelined execution
  - Five-plane, 2D-mesh NoC efficiently supports multiple independent frequency domains and a variety of platform services

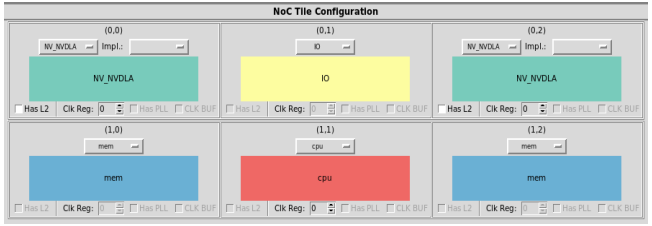
[P. Mantovani , L. P. Carloni et al., *An FPGA-Based Infrastructure for Fine-Grained DVFS Analysis in High-Performance Embedded Systems*, DAC 2016]

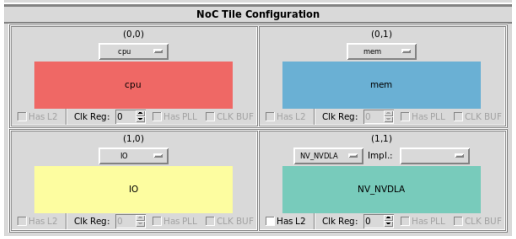
©Luca Carloni

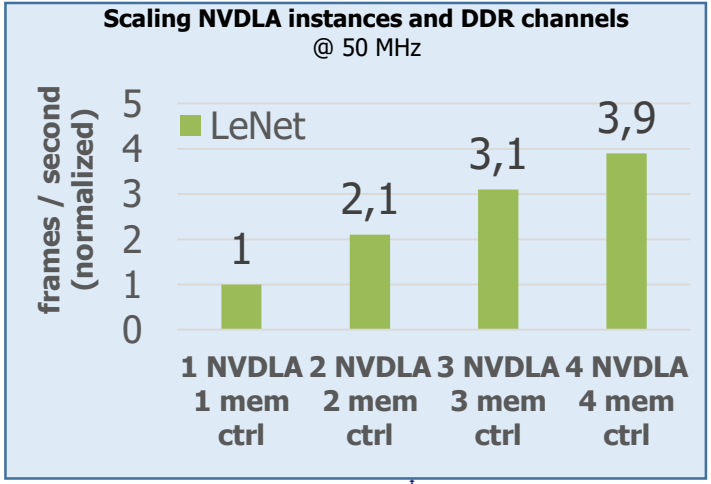
Seamless Integration of NVDLA Accelerators

- New design flow of general applicability to integrate third-party accelerators
  - demonstrated w/ NVIDIA Deep Learning Accelerator (NVDLA)
- Transparent accelerator integration
  - original software apps can run "as is"
- Linear performance scalability
  - when scaling up NVDLA instances with DDR channels









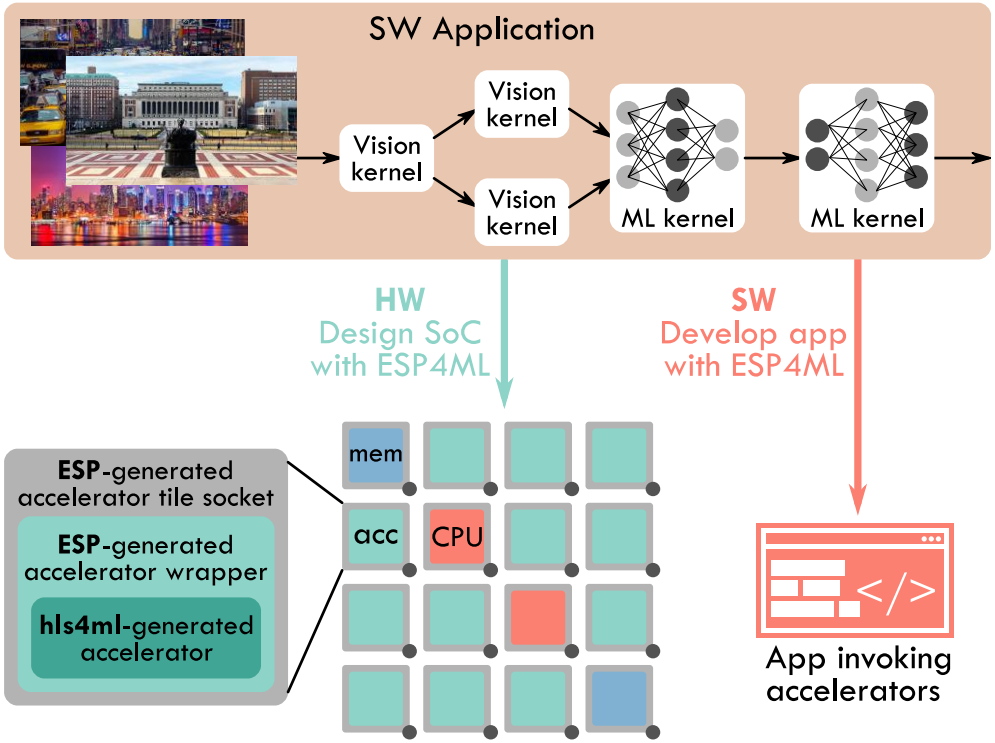
[D. Giri et al. "Ariane + NVDLA: Seamless Third-Party IP Integration with ESP", CARRV'20]

©Luca Carloni

ESP4ML

Open-source design flow to build and program SoCs for ML applications

- Combines ESP and hls4ml
  - **ESP** is a platform for heterogeneous SoC design
  - **hls4ml** automatically generates accelerators from ML models
- Main **contributions to ESP**:
  - Automated integration of hls4ml accelerators
  - Accelerator-accelerator communication
  - Accelerator invocation API



[D. Giri, K.-L. Chiu, G. Di Guglielmo, P. Mantovani, and L. P. Carloni. "ESP4ML: Platform-Based Design of Systems-on-Chip for Embedded Machine Learning", DATE '20]

©Luca Carloni

hls4ml

- Open-source tool developed by **Fast ML Lab**
- Translates ML algorithms into accelerator specifications that are synthesizable with high-level synthesis tools for both FPGA and ASIC implementations
- Born for high-energy physics (small and ultra-low latency networks), it is gaining broad applicability and a growing community of contributors and users

hls4ml: An Open-Source Codesign Workflow to Empower Scientific Low-Power Machine Learning Devices

Farah Fahim*, Benjamin Hawks, Christian Herwig, James Hirschauer, Sergo Jindariani, Nhan Tran*
Fermilab, Batavia, IL, USA

Luca P. Carloni, Giuseppe Di Guglielmo
Columbia University, New York, NY, USA

Philip Harris, Jeffrey Krupa, Dylan Rankin
MIT, Cambridge, MA, USA

Manuel Blanco Valentin, Josiah Hester, Yingyi Luo, John Mamish, Seda Orgrenci-Memik
Northwestern University, Evanston, IL, USA

Thea Aarrestad, Hamza Javed, Vladimir Loncar, Maurizio Pierini, Adrian Alan Pol, Sioni Summers
European Organization for Nuclear Research (CERN), Geneva, Switzerland

Javier Duarte
UC San Diego, La Jolla, CA, USA
jduarte@ucsd.edu

Scott Hauck, Shih-Chieh Hsu
University of Washington, Seattle, WA, USA

Jennifer Ngadiuba
Caltech, Pasadena, CA, USA

Mia Liu
Purdue University, West Lafayette, IN, USA

Duc Hoang
Rhodes College, Memphis, TN, USA

Edward Kreinar
HawkEye360, Herndon, VA, USA

Zhenbin Wu
University of Illinois at Chicago, Chicago, IL, USA

arXiv:2103.05579v3 [cs.LG] 23 Mar 2021

ABSTRACT

Accessible machine learning algorithms, software, and diagnostic tools for energy-efficient devices and systems are extremely valuable across a broad range of application domains. In scientific domains, real-time near-sensor processing can drastically improve experimental design and accelerate scientific discoveries. To support domain scientists, we have developed hls4ml, an open-source software-hardware codesign workflow to interpret and translate machine learning algorithms for implementation with both FPGA and ASIC technologies. In this paper, we describe the essential features of the hls4ml workflow including network optimization techniques—such as pruning and quantization-aware training—which can be incorporated naturally into the device implementations. We expand on previous hls4ml work by extending capabilities and techniques towards low-power implementations and increased usability: new Python APIs, quantization-aware pruning, end-to-end FPGA workflows, long pipeline kernels for low power, and new device backends include an ASIC workflow. Taken together, these and continued efforts in hls4ml will arm a new generation of domain scientists with accessible, efficient, and powerful tools for machine-learning-accelerated discovery.

KEYWORDS

hls4ml, machine learning, neural networks, tinyML, FPGA, ASIC, low-power, low-latency

ACM Reference Format:
Farah Fahim, Benjamin Hawks, Christian Herwig, James Hirschauer, Sergo Jindariani, Nhan Tran, Luca P. Carloni, Giuseppe Di Guglielmo, Philip Harris, Jeffrey Krupa, Dylan Rankin, Manuel Blanco Valentin, Josiah Hester, Yingyi Luo, John Mamish, Seda Orgrenci-Memik, Thea Aarrestad, Hamza Javed, Vladimir Loncar, Maurizio Pierini, Adrian Alan Pol, Sioni Summers, Javier Duarte, Scott Hauck, Shih-Chieh Hsu, Jennifer Ngadiuba, Mia Liu, Duc

*Also affiliated with Northwestern University

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

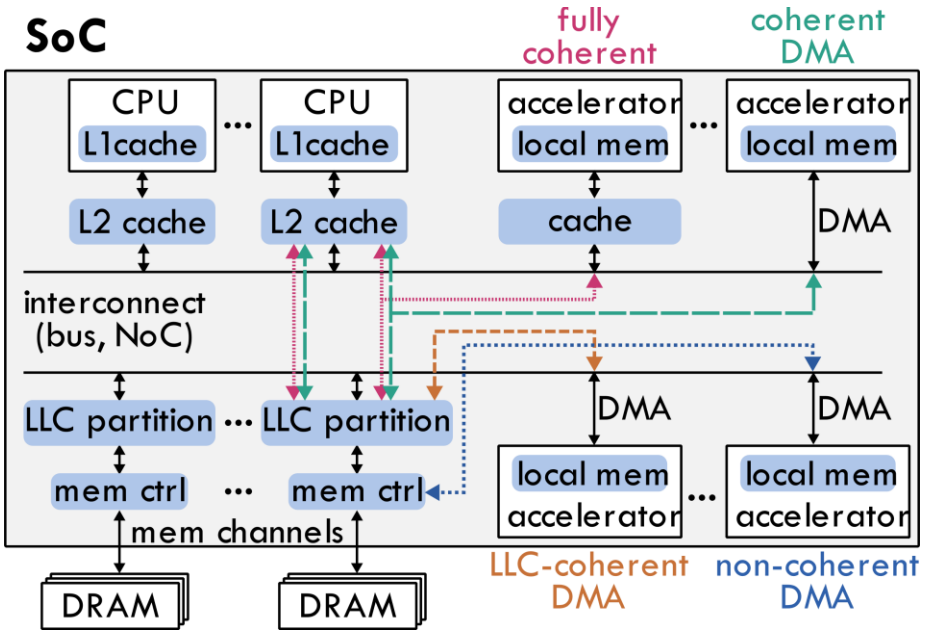
TinyML Research Symposium'21, March 2021, San Jose, CA

© 2021 Copyright held by the owner/author(s).

©Luca Carloni

Cohmeleon: Learning-Based Orchestration of Accelerator Coherence in Heterogeneous SoCs

- Accelerator performance can vary greatly based on coherence modes
  - SoCs should support multiple coherence modes for optimal performance
- Reinforcement learning can be used to automatically manage coherence mode decisions
- With little overhead, Cohmeleon provides significant performance benefits for multiple objectives

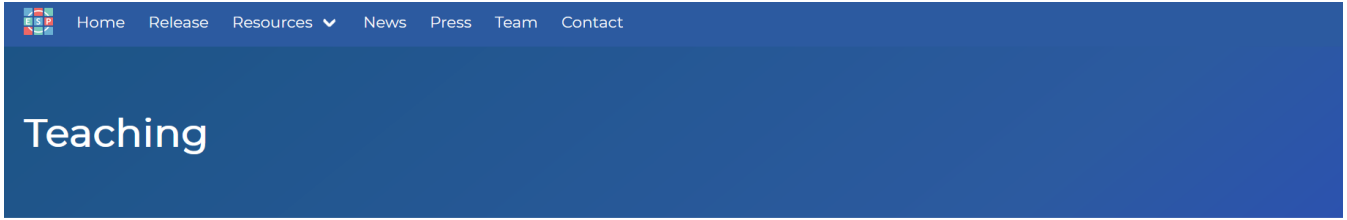


[J. Zuckerman, D. Giri, J. Kwon, P. Mantovani, and L. P. Carloni Cohmeleon: Learning-Based Orchestration of Accelerator Coherence in Heterogeneous SoCs IEEE/ACM International Symposium on Microarchitecture (MICRO-54), 2021.]

©Luca Carloni

Teaching with ESP

https://www.esp.cs.columbia.edu



Class projects

1) Design and integration of an accelerator with ESP

For this project each student will use ESP to design one or more accelerators and to integrate them in a system-on-chip (SoC), capable of booting Linux. Then the student will evaluate the SoC both with RTL simulation and on FPGA.

To get a more practical sense of the project, you should familiarize yourself with ESP by using the resources on this website. Specifically:

- Check out the ESP website Homepage including the short introductory video.
- Watch the 16 minutes overview video in the Documentation section.
- Watch the videos and read the guides of the relevant hands-on tutorials available in the Documentation section. Especially relevant are the *"How to: setup"*, *"How to: design a single-core SoC"* guides and the *"How to: design an accelerator in ..."* guide that applies to your specific project.
- Explore the rest of the website to get the full picture of the ESP project.

Accelerator flows

For your project proposal, you are asked to choose which design flow you want to use to build your accelerator.

ESP offers multiple accelerator design flows: Stratus HLS flow (accelerator designed in SystemC), the Vivado HLS flow (accelerator designed in C/C++), the Catapult HLS flow (accelerator designed in C/C++) and the hls4ml flow (accelerators designed in Keras/Pytorch/ONNX).

Other options include designing the accelerator in RTL (Verilog, VHDL, SystemVerilog, Chisel). These other options do not have full support and documentation yet. It is possible to use them, but they will require a bigger integration effort.

©Luca Carloni

CSEE-4868: System-on-Chip Platforms

- ***Foundation course on the programming, design, and validation of SoCs with emphasis on high-performance embedded applications***
- Offered at Columbia since 2011, moved to upper-level curriculum in Fall 2016
  - required course for CE BS program, elective for MS programs in CS and EE
- Course Goals
  - mastering the HW and SW aspects of integrating heterogeneous components into a complete system
  - designing new components that are reusable across different systems, product generations, and implementation platforms
  - evaluating designs in a multi-objective optimization space

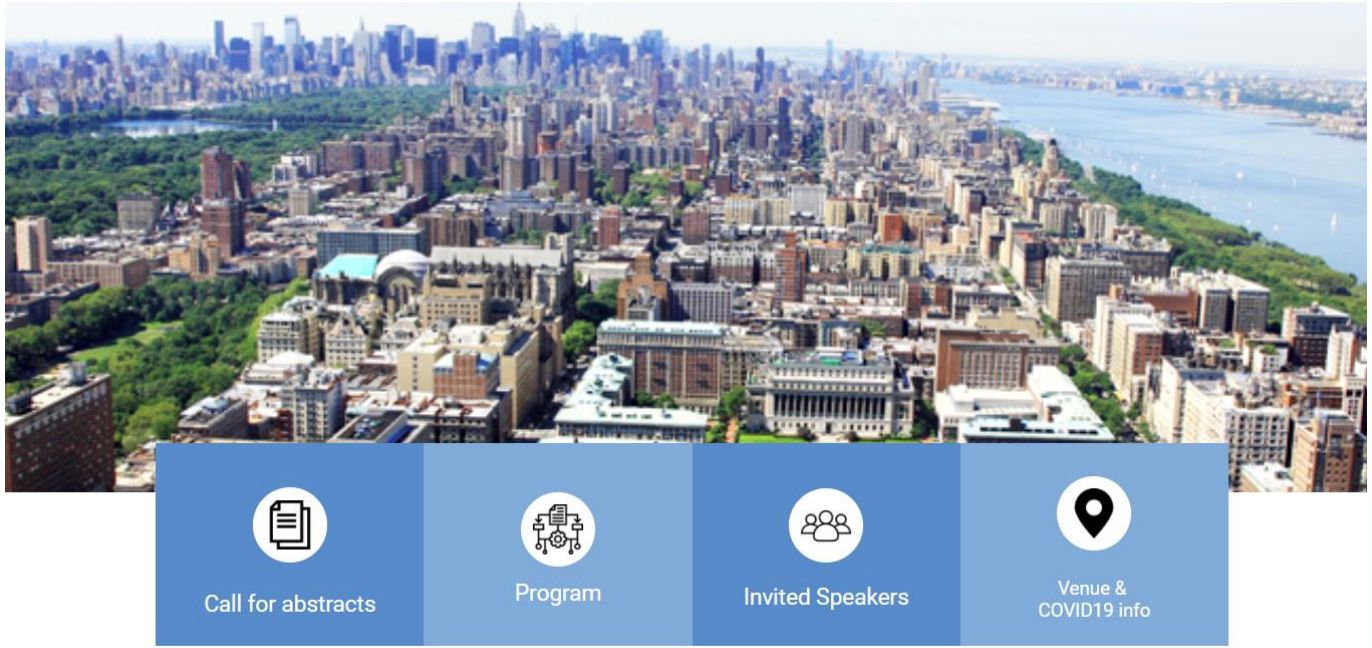
[L. P. Carloni et al. Teaching Heterogeneous Computing with System-Level Design Methods, WCAE 2019]

©Luca Carloni

Upcoming Events: The First OSCAR Workshop!

Open-Source Computer Architecture Research (OSCAR)

Saturday, June 18, 2022 - New York City (co-located with ISCA 2022)



https://oscar-workshop.github.io/

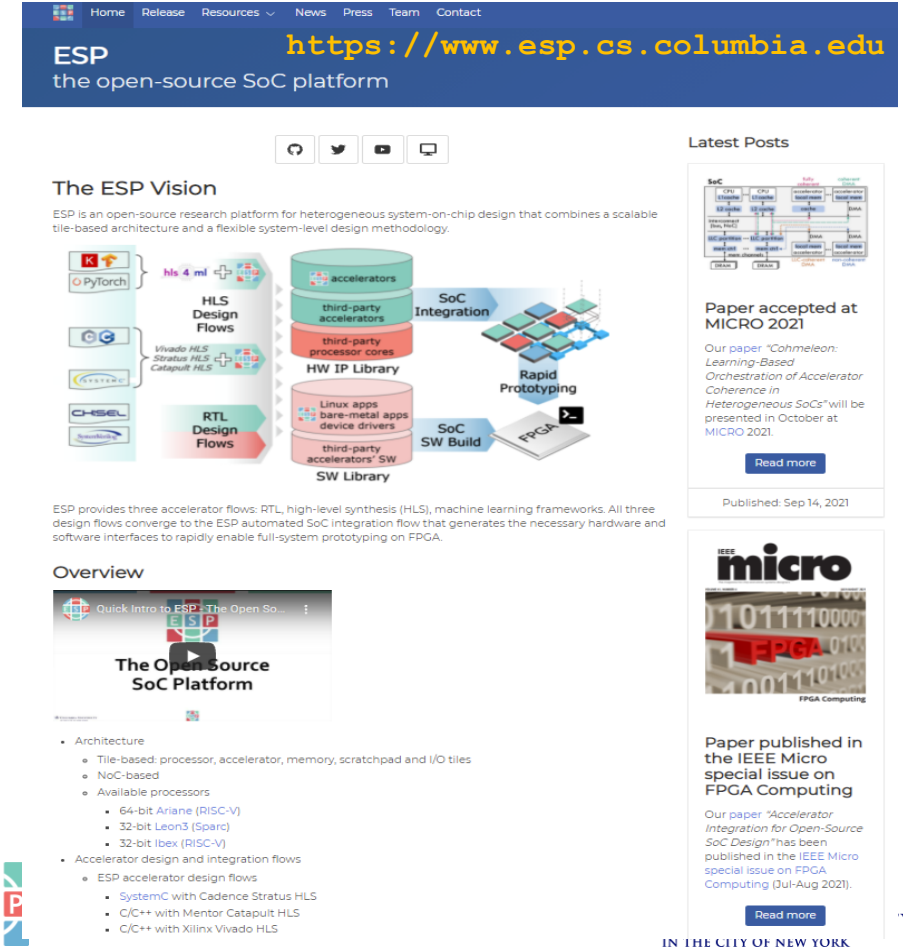
Welcome to OSCAR 2022!

OSCAR 2022 is the first edition of a new workshop on open-source hardware which addresses the wide variety of challenges encountered by both hardware and software engineers in dealing with the increasing heterogeneity of next-generation computer architectures. By providing a venue which brings together researchers from academia, industry and government labs, OSCAR promotes a collaborative approach to foster the efforts of the open-source hardware community in this direction.

©Luca Carloni

In Summary: ESP for Open-Source Hardware

- We contribute **ESP** to the OSH community in order to support the realization of
  - **more scalable** architectures for SoCs that integrate
  - **more heterogeneous** components, thanks to a
  - **more flexible** design methodology, which accommodates different specification languages and design flows
- ESP was conceived as a heterogeneous integration platform from the start and tested through years of teaching at Columbia University
- We invite you to **use ESP** for your projects and to **contribute to ESP**!



©Luca Carloni

Thank you from the ESP team!

https://esp.cs.columbia.edu

https://github.com/sld-columbia/esp

System Level Design Group

COMPUTER SCIENCE

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK