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

Luca P. Carloni
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!
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
Open-Source Hardware (OSH)

• An opportunity to reenergize the innovation in the semiconductor and electronic design automation industries

• The OSH community is gaining momentum
  o many diverse contributions from both academia and industry
  o multi-institution organizations
  o 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/
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?
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
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

Quick Intro to ESP: The Open Source SoC Platform
Outline

The ESP Architecture

- SW Library
- HW IP Library
- Third-party accelerators
- Third-party processor cores
- SoC Integration
- Rapid Prototyping
- SoC SW Build

The ESP Methodology

- SW Build

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.
• Processor off-the-shelf
  o RISC-V CVA6-Ariane (64 bit)
  SPARC V8 Leon3 (32 bit)
  o RISC-V IBEX (32 bit)
  o L1 private cache

• L2 private cache
  o Configurable size
  o MESI protocol

• IO/IRQ channel
  o Un-cached
  o Accelerator config. registers, interrupts, flush, UART, ...
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**
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
ESP Accelerator Socket

**ESP Accelerator Socket**

**ESP accelerator**

- HLS [C/C++, SystemC, Tensorflow*, Pytorch*]
- RTL [Chisel, Verilog, ...]

- read/write
- config
- done

- private cache
- TLB
- DMA ctrl
- cfg regs
- IRQ

**Third-Party Accelerator Socket**

**third-party accelerator**

- (NVDLA*, ...)

- read/write
- config
- done

- AXI4 bus
- APB bus
- IRQ

[Diagram of ESP Accelerator Socket and Third-Party Accelerator Socket with labels and connections]

©Luca Carloni
ESP Software Socket

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

```c
/* 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 */
```
### 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
Outline

The ESP Architecture

The ESP Methodology

Research & Teaching with ESP
The Pillars of the ESP Approach

• Develop platforms, not just architectures
  o A platform combines an architecture and a companion design methodology

• Move from a processor-centric to an SoC-centric perspective
  o The processor core is just one component among many others

• Raise the level of abstraction
  o Move from RTL design to domain-specific system-level design with high-level synthesis...
  o …but keep supporting different abstraction levels and design flows

• Promote Open-Source Hardware
  o Build libraries of reusable components
  o Support the integration of third-party IP components
The ESP Vision: Domain Experts Can Design SoCs

- HLS Design Flows
  - Keras, Pytorch, ONNX
  - C/C++
  - SystemC
  - Chisel
  - SV, Verilog, VHDL

- RTL Design Flows
  - Vivado HLS
  - Stratus HLS
  - Catapult HLS

- HW IP Library
  - third-party cores (CPU, Mem)
  - third-party accelerators (CPU, Acc, Mem)

- SW Library
  - Linux apps
  - bare-metal apps
  - device drivers

- HLS design flows: Keras, Pytorch, ONNX
- C/C++
- SystemC
- Chisel
- SV, Verilog, VHDL

- RTL design flows: Vivado HLS, Stratus HLS, Catapult HLS

- HW IP Library
  - CPU, Mem
  - Acc

- SW Library
  - Linux apps
  - bare-metal apps
  - device drivers

- SoC Generation
  - FPGA Prototyping
  - ASIC Design

- SoC Configuration
  - Acc, Mem
  - CPU, Acc, Mem
  - I/O, Acc

- SW Build
  - FPGA
  - ASIC
ESP Methodology In Practice

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

ESP4ML

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

• Combines ESP and hls4ml
  
  o ESP is a platform for heterogeneous SoC design
  o hls4ml automatically generates accelerators from ML models

• Main contributions to ESP:
  o Automated integration of hls4ml accelerators
  o Accelerator-accelerator communication
  o Accelerator invocation API

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
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

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.
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]
Upcoming Events: The First OSCAR Workshop!

Open-Source Computer Architecture Research (OSCAR)

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

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 effort to foster the efforts of the open-source hardware community in this direction.
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!

https://www.esp.cs.columbia.edu
Thank you from the ESP team!

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

System Level Design Group