Open ESP
The Open-Source SoC Platform

Luca P. Carloni

Columbia University
in the City of New York

UW-Madison Virtual Computer Architecture Seminar
September 22nd, 2020

Open Source Release of ESP

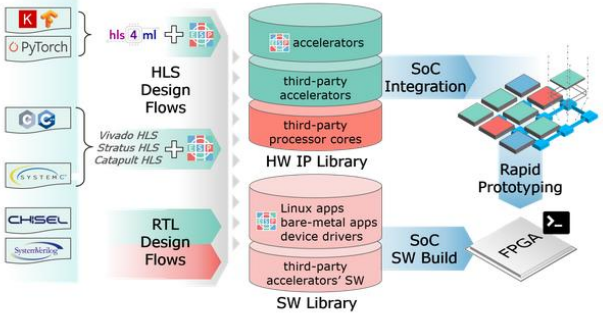
https://www.esp.cs.columbia.edu

Home Resources News Press Team Contact

ESP
the open-source SoC platform

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

Latest Posts

Upcoming talk at VLSISoC 2020

Professor Carloni will give a talk titled *"Scalable Open-Source System-on-Chip Design"* at the VLSISoC conference on October 7th, 2020.

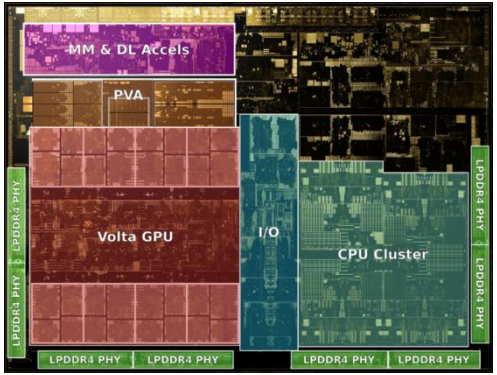
Read more

Published: Sep 11, 2020

Upcoming tutorial at MICRO 2020

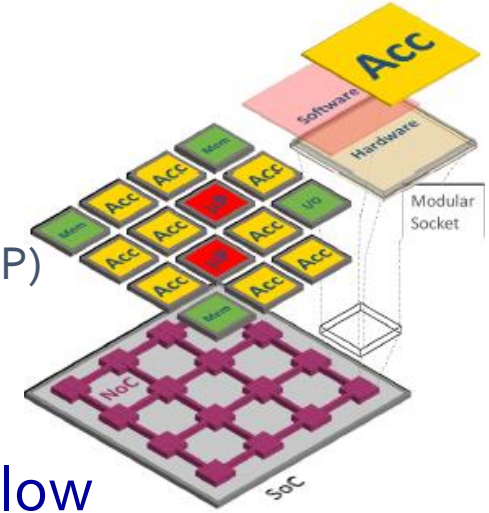
We will present a tutorial on ESP at MICRO 2020.

Outline

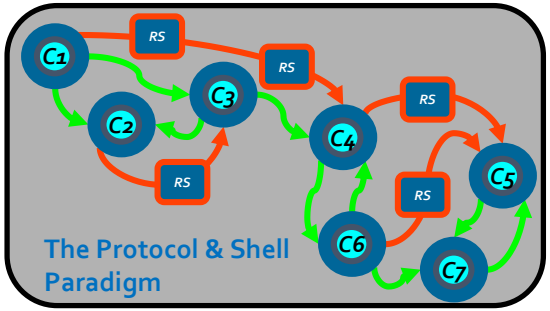


1. Motivation
   - The Rise of Heterogeneous Computing

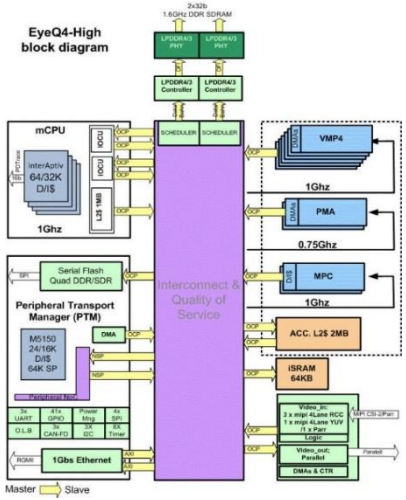
2. Proposed Architecture
   - Embedded Scalable Platforms (ESP)



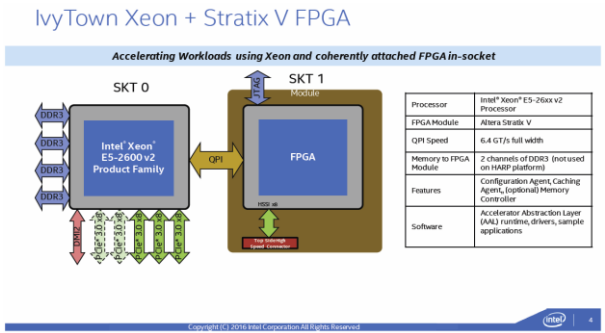
3. Methodology and Design Flow
   - with a Retrospective on Latency-Insensitive Design



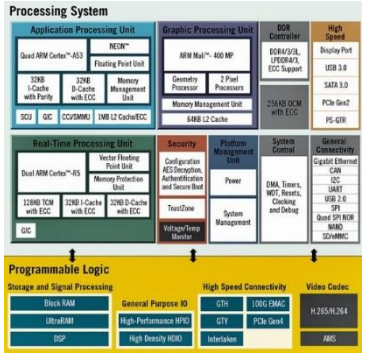
Heterogeneous Architectures Are Emerging Everywhere



[Source: www.mobileye.com/]



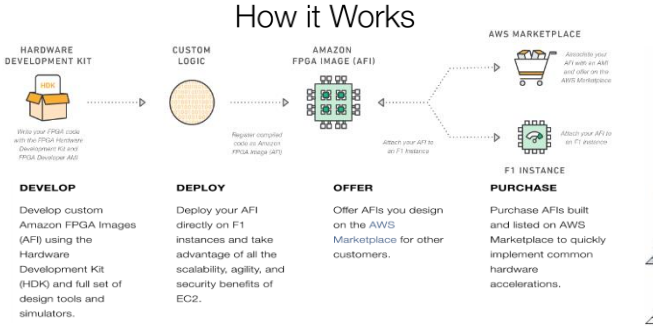
[Source: "Xeon+FPGA Tutorial @ ISCA'16"]



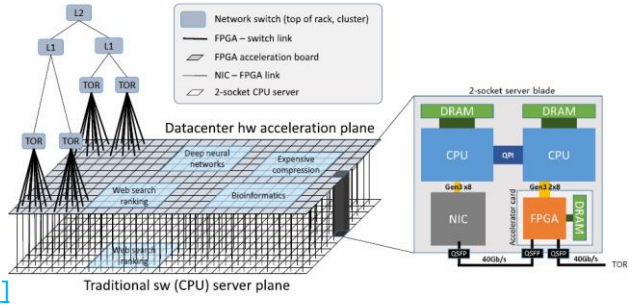
[Source: www.xilinx.com/]

[Source: https://cloudplatform.googleblog.com/]

[Source: https://blogs.nvidia.com/]



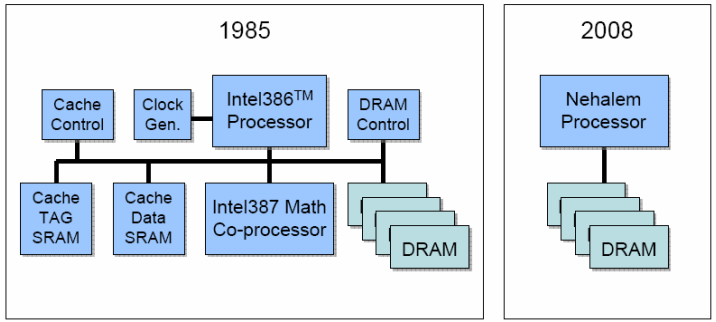
[Source: https://aws.amazon.com/ec2/instance-types/f1/]

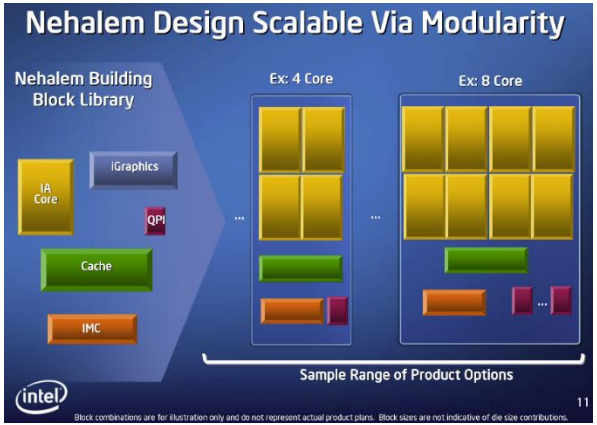


[Source: www.microsoft.com/]

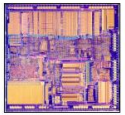
From Microprocessors to Systems-on-Chip (SoC)

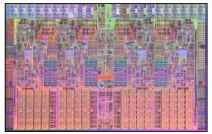
[Source: M. Bohr 2009]





Source Intel.com [P. Gelsinger Press Briefing, Mar'08]





	Intel386TM	Nehalem
Transistor Count:	280 thousand	731 million
Frequency:	16 MHz	>3.6 GHz
# Cores:	1	4
Cache Size:	None	8 MB
I/O Peak Bandwidth:	64 MB/sec	50 GB/sec
Adaptive Circuits:	None	Sleep Mode Turbo Mode Power Gating Adaptive Frequency Clocking

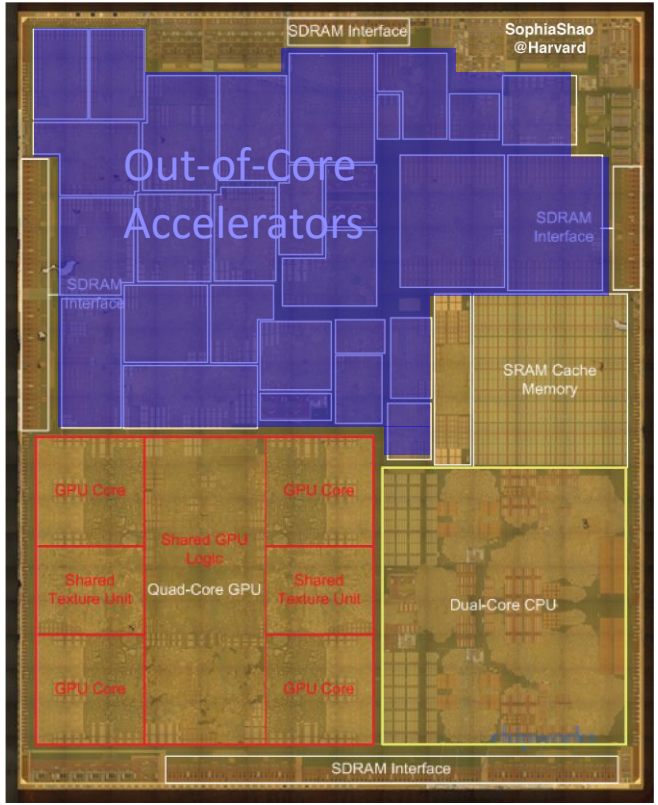
4.1 14nm 6th-Generation Core Processor SoC with Low Power Consumption and Improved Performance

Eyal Fayneh, Marcelo Yuffe, Ernest Knoll, Michael Zelikson, Muhammad Abozaed, Yair Talker, Ziv Shmuely, Saher Abu Rahme

Intel, Haifa, Israel

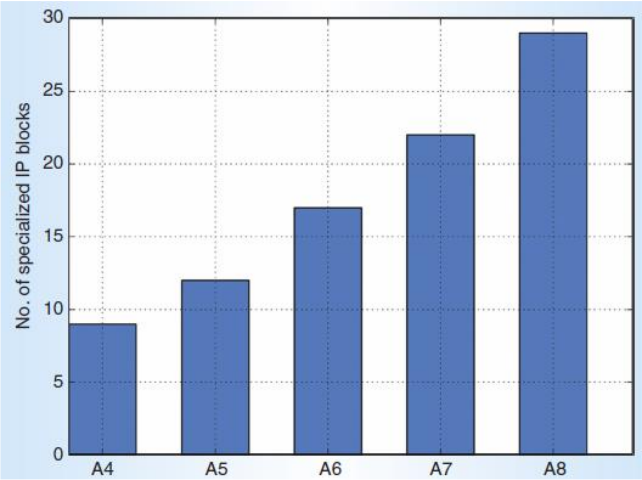
Intel's 6th generation Core processor (code named "Skylake" or SKL) was designed

The Growth of Specialized IP Blocks: The Apple A8 SoC



[Source: Shao et al. 2015]

Number of specialized IP blocks across five generations of Apple SoCs



- The analysis of die photos from Apple's A6, A7, and A8 SoCs shows that more than half of the die area is dedicated to blocks that are neither CPUs nor GPUs, but rather specialized Intellectual Property (IP) blocks
- Many IP blocks are *accelerators*, i.e. specialized hardware components that execute an important computation more efficiently than software

The Age of Heterogeneous Computing

- **The migration from homogeneous multi-core architectures to heterogeneous System-on-Chip architectures will accelerate, across almost all computing domains**
  - from IoT devices, embedded systems and mobile devices to data centers and supercomputers
- **A heterogeneous SoC will combine an increasingly diverse set of components**
  - different CPUs, GPUs, hardware accelerators, memory hierarchies, I/O peripherals, sensors, reconfigurable engines, analog blocks…
- **The set of heterogeneous SoCs in production in any given year will be itself heterogeneous!**
  - no single SoC architecture will dominate all the markets

Where the Key Challenges in SoC Design Are…

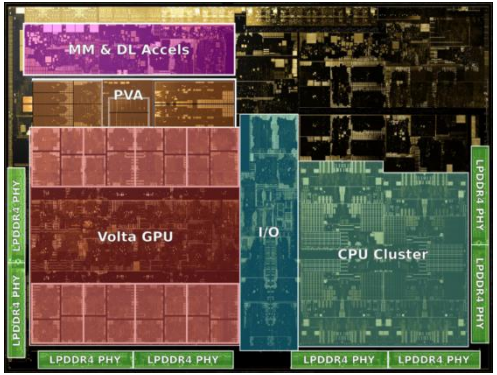
- The biggest challenges are (and will increasingly be) found in the complexity of system integration
  - How to design, program and validate scalable systems that combine a very large number of heterogeneous components to provide a solution that is specialized for a target class of applications?

- How to handle this complexity?
  - raise the level of abstraction to System-Level Design
  - adopt compositional design methods with the Protocol & Shell Paradigm
  - promote Design Reuse

What is Needed? To Think at the System Level.

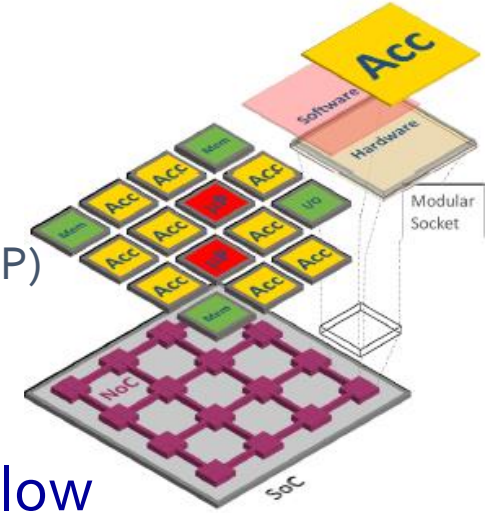
- Move from a processor-centric to an SoC-centric perspective
  - The processor core is just one component among many others
- Develop platforms, not just architectures
  - A platform combines an architecture and a companion design methodology
- Raise the level of abstraction
  - Move from RTL Design to System-Level Design
- Promote Open-Source Hardware
  - Build libraries of reusable components

Outline

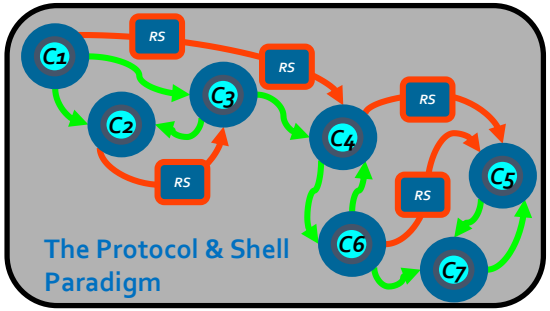


1. Motivation
   - The Rise of Heterogeneous Computing

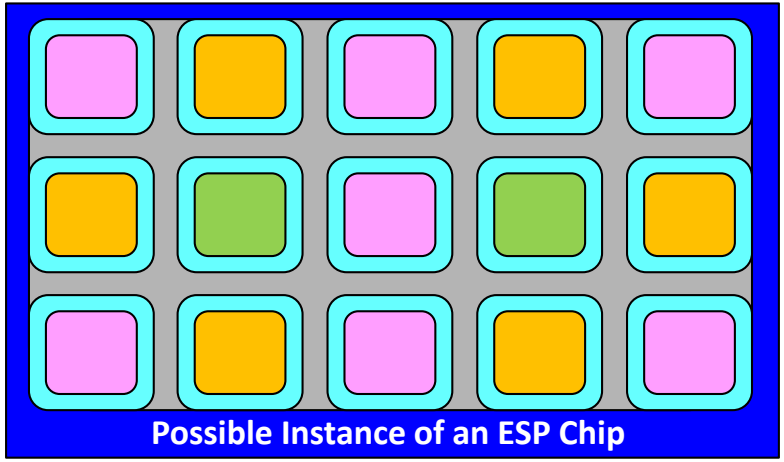
2. Proposed Architecture
   - Embedded Scalable Platforms (ESP)



3. Methodology and Design Flow
   - with a Retrospective on Latency-Insensitive Design



The ESP Scalable Architecture Template

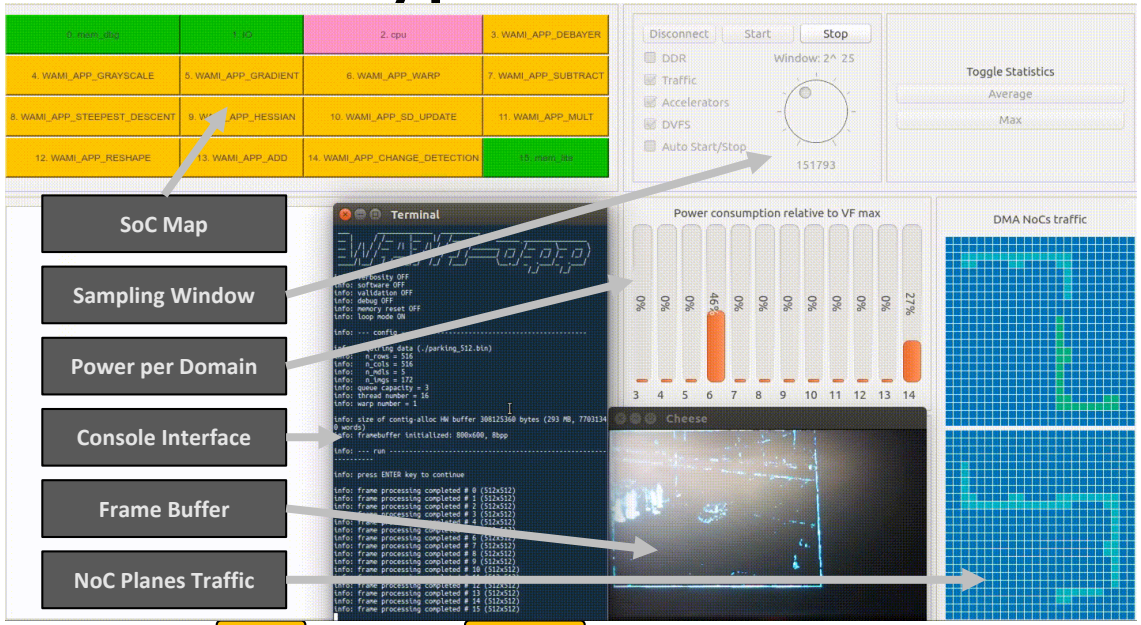


- Processor Tiles
  - each hosting at least one configurable processor core capable of running an OS
- Accelerator Tiles
  - synthesized from high-level specs
- Other Tiles
  - memory interfaces, I/O, etc.
- Network-on-Chip (NoC)
  - playing key roles at both design and run time

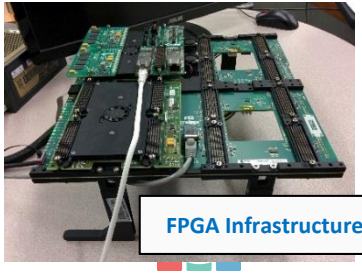
Template Properties

- Regularity
  - tile-based design
  - pre-designed on-chip infrastructure for communication and resource management
- Flexibility
  - each ESP design is the result of a configurable mix of programmable tiles and accelerator tiles
- Specialization
  - with automatic high-level synthesis of accelerators for key computational kernels

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



Motion Detection from WAMI-Application



FPGA Infrastructure

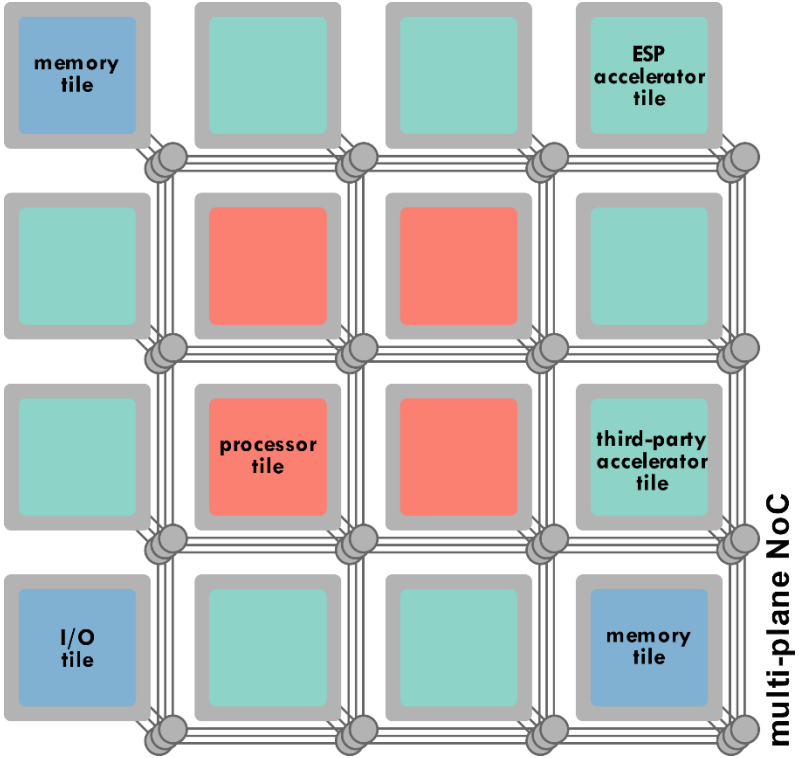
- **Design:** Complete design of WAMI-App running on an FPGA implementation of an ESP architecture
  - featuring 1 embedded processor, 12 accelerators, 1 five-plane NoC, and 2 DRAM controllers
  - SW application running on top of Linux while leveraging multi-threading library to program the accelerators 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]

ESP Architecture

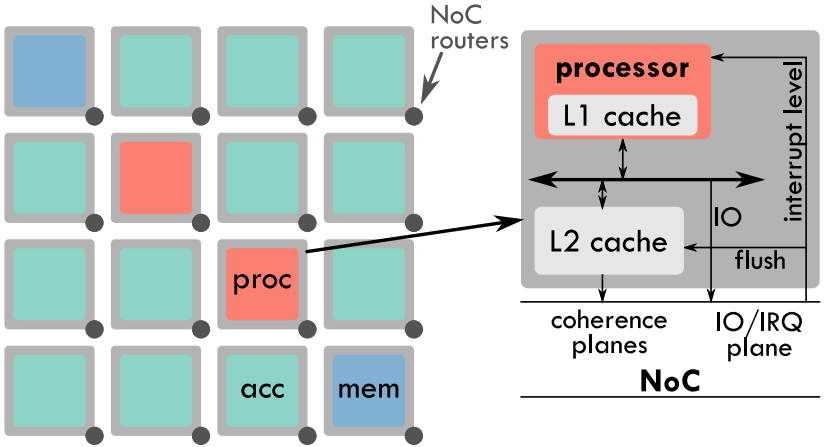
- RISC-V Processors
- Many-Accelerator
- Distributed Memory
- Multi-Plane NoC

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



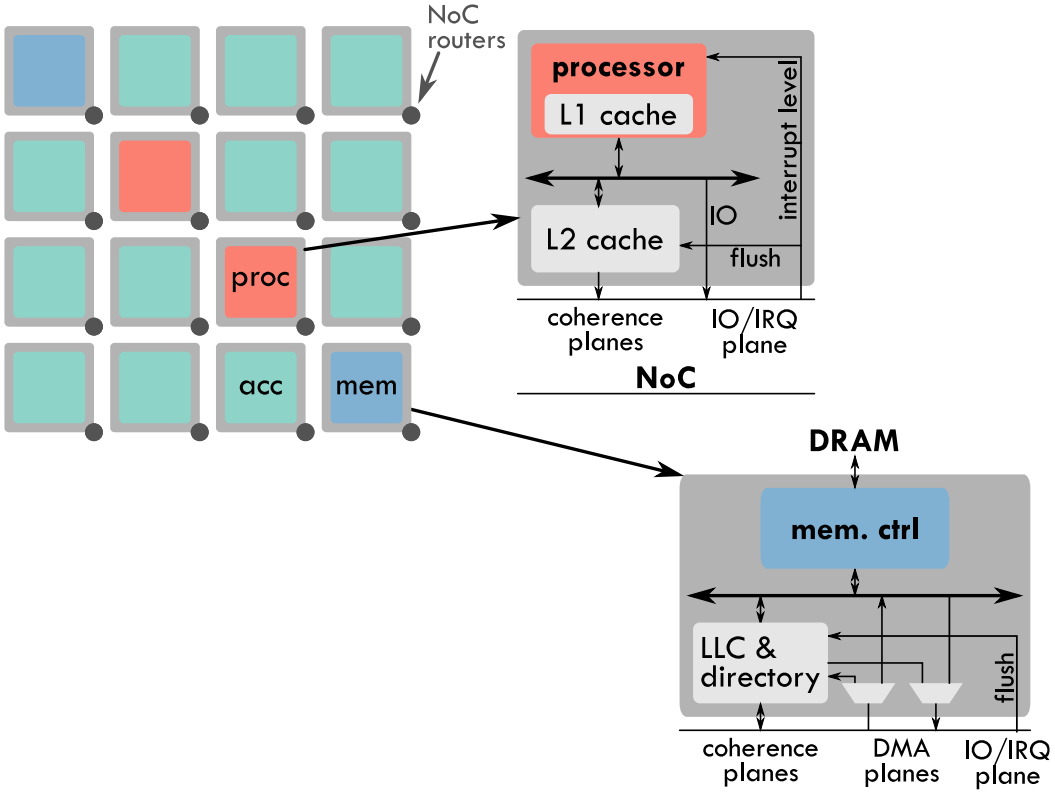
ESP Architecture: Processor Tile

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



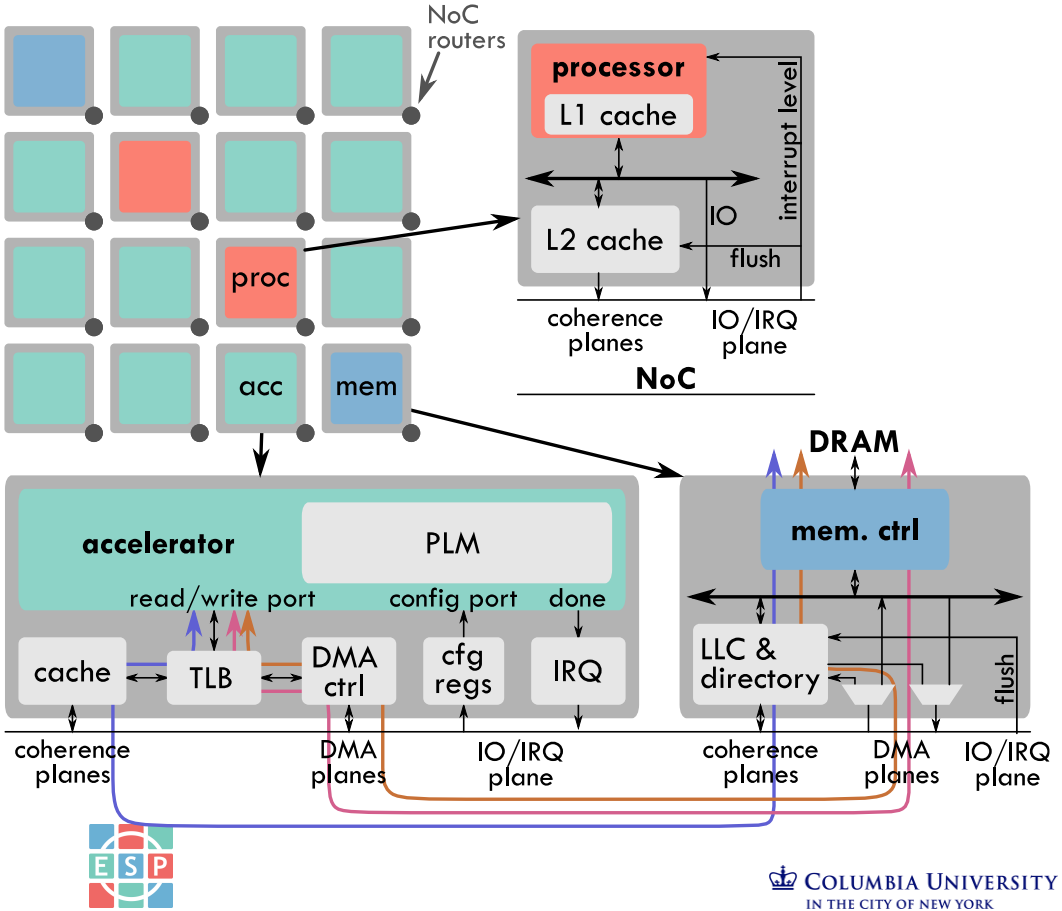
ESP Architecture: Memory Tile

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



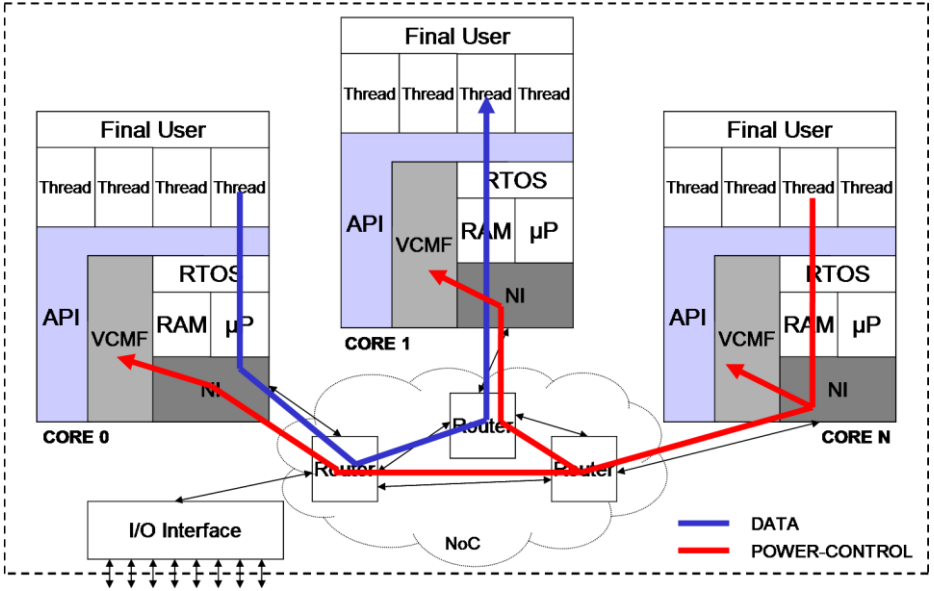
ESP Architecture: Accelerator Tile

- 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



The Twofold Role of the Network-on-Chip

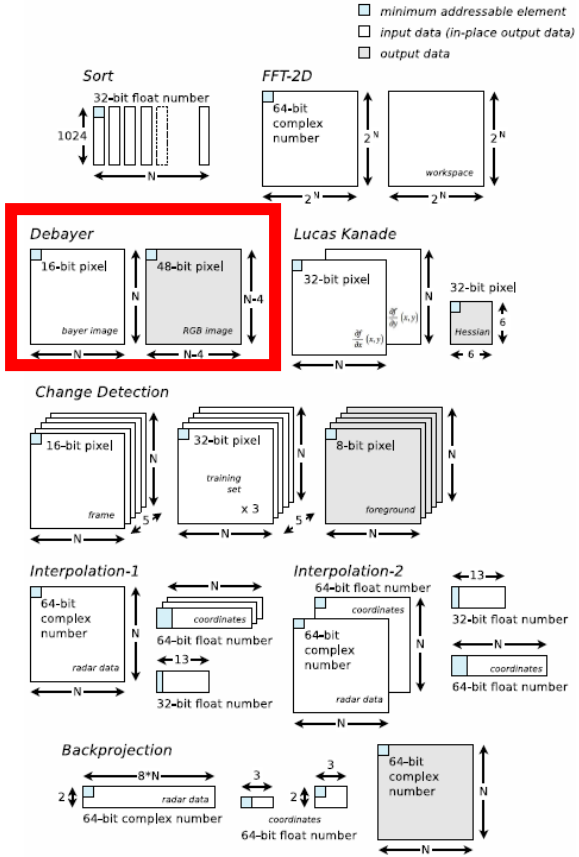
- A scalable NoC is instrumental to accommodate heterogeneous concurrency and computing locality in ESP
  - At Design Time
    - simplifies integration of heterogeneous tiles to balance regularity and specialization
  - At Run Time
    - energy efficient inter-tile data communication with integrated support for fine-grain power management and other services



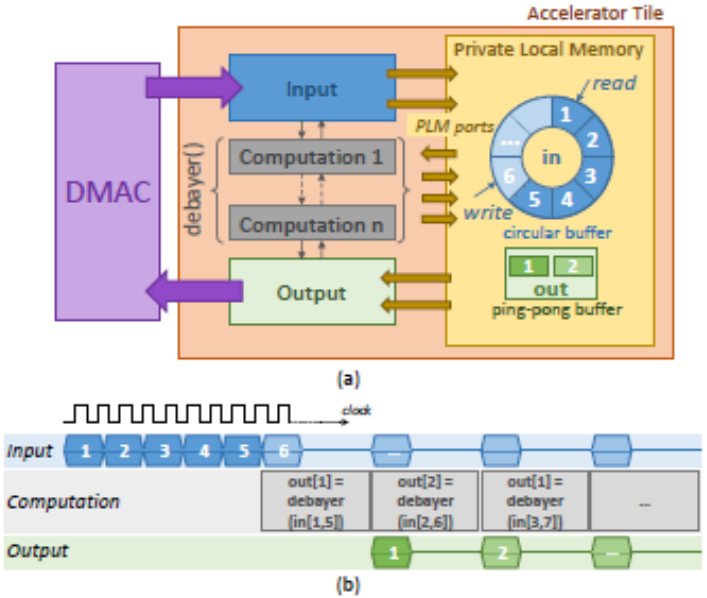
- The NoC Interface interacts directly with the Tile Socket that supports the ESP Platform Services
  - communication/synchronization channels among tiles
  - fine-grain power management with dynamic voltage-frequency scaling
  - seamless dynamic support for various accelerator coherence models

Heterogeneous Applications Bring Heterogeneous Requirements

Data Structures of the PERFECT TAV Benchmarks



Structure and Behavior of the Debayer Accelerator



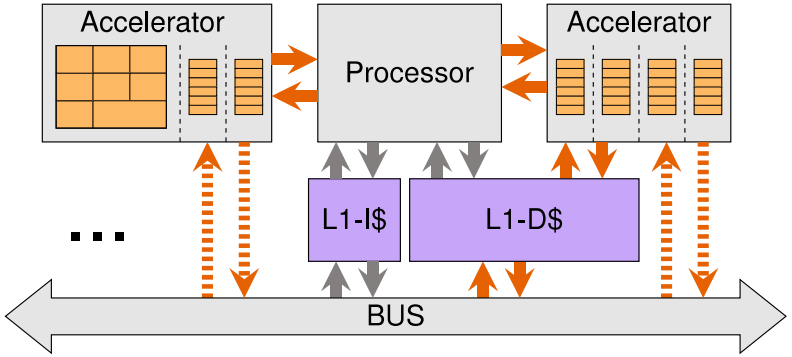
- While the Debayer structure and behavior is representative of the other benchmarks, the specifics of the actual computations, I/O patterns, and scratchpad memories vary greatly among them

How to Couple Accelerators, Processors and Memories?

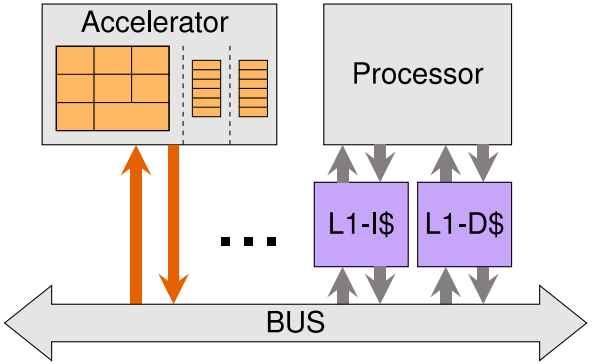
- **Private local memories** (aka **scratchpads**) are key to performance and energy efficiency of accelerators
- There are two main models of coupling accelerators with processors, memories
  - Tightly-Coupled Accelerators
  - Loosely-Coupled Accelerators

[E. G. Cota, P. Mantovani, G. Di Guglielmo, and L. P. Carloni, An Analysis of Accelerator Coupling in Heterogeneous Architectures, DAC'15]

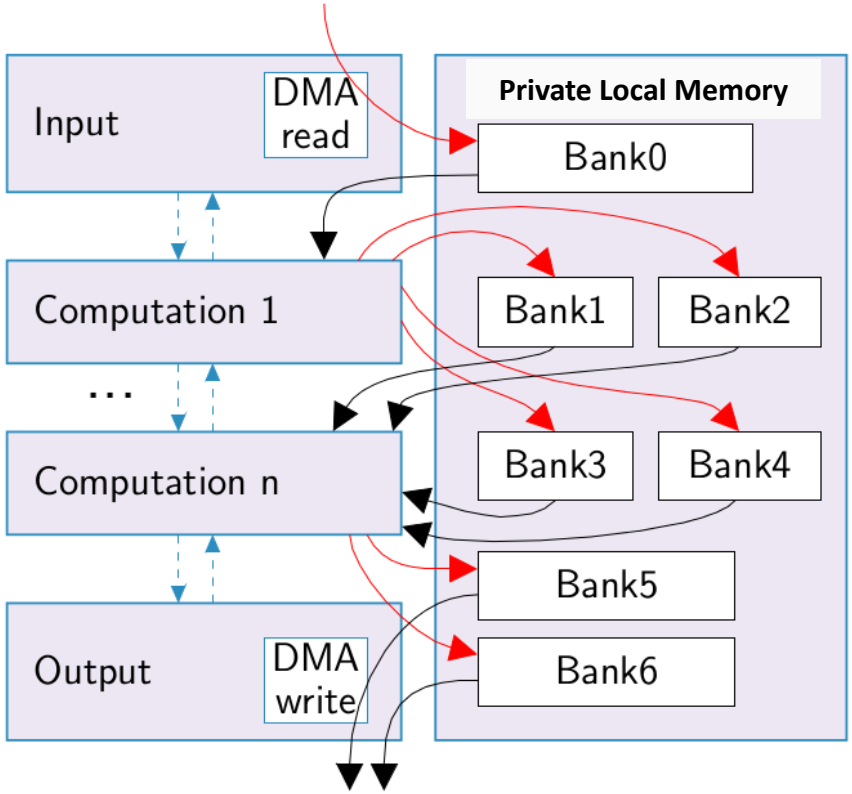
Tightly-Coupled Accelerators (TCA)



Loosely-Coupled Accelerators (LCA)



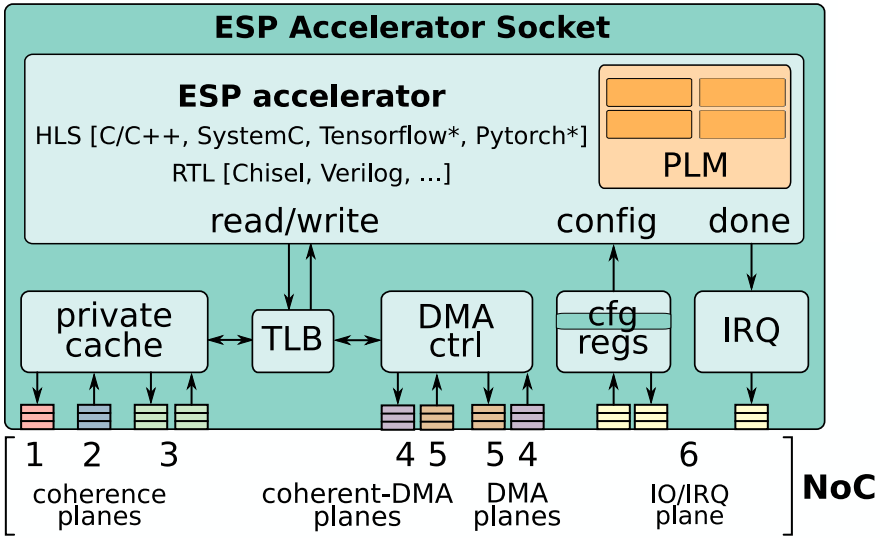
The Key Role of the Private Local Memories (PLM)

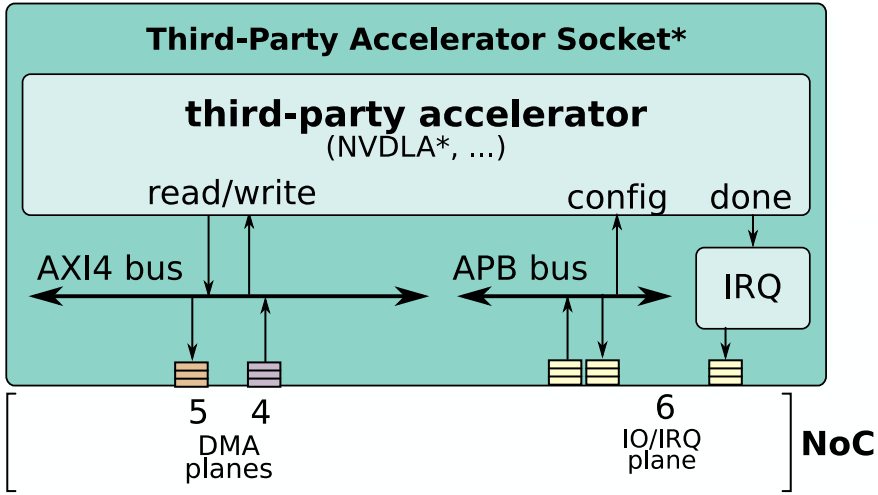


- Tailored, many-ported PLMs are **key to accelerator performance**
- A scratchpad features **aggressive SRAM banking** that provides multi-port memory accesses to match the multiple parallel blocks of the computation datapath
  - Level-1 caches cannot match this parallelism

[C. Pilato, P. Mantovani, G. Di Guglielmo, and L. P. Carloni, System-Level Optimization of Accelerator Local Memory for Heterogeneous Systems-on-Chip. IEEE Trans. on CAD of Integrated Circuits and Systems, 2017.]

ESP Accelerator Socket





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

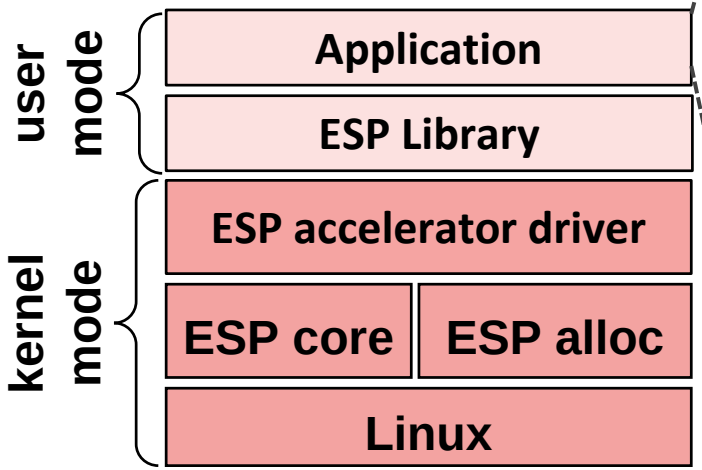
- Debug interface
- Performance counters access
- Coherent DMA
- Shared peripherals (UART, ETH, …)

Memory Tile

- Independent DDR Channel
- LLC Slice
- DMA Handler

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        */
}
```

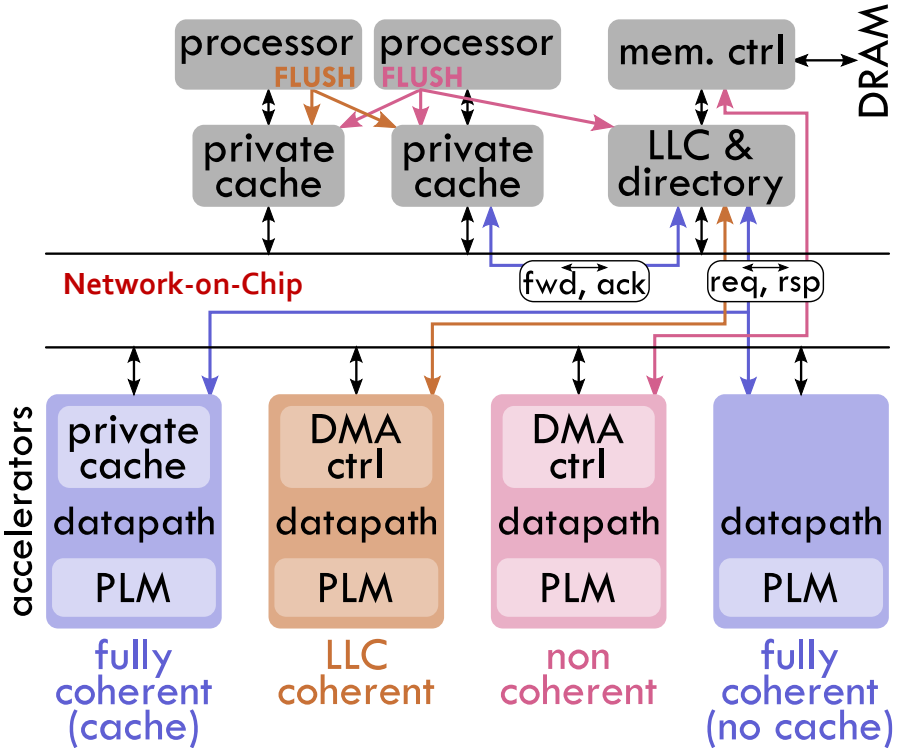
Cache Coherence and Loosely-Coupled Accelerators

- **An analysis of the literature indicates that there are three main cache-coherence models for loosely-coupled accelerators:**

1. **Non-Coherent Accelerator**
   - the accelerator operates through DMA bypassing the processor caches
2. **Fully-Coherent Accelerator**
   - the accelerator issues main-memory requests that are coherent with the entire cache hierarchy
     - this approach can endow accelerators with a private cache, thus requiring no updates to the coherence protocol
3. **Last Level Cache (LLC)-Coherent Accelerator**
   - the accelerator issues main-memory requests that are coherent with the LLC, but not with the private caches of the processors
     - in this case, DMA transactions address the shared LLC, rather than off-chip main memory

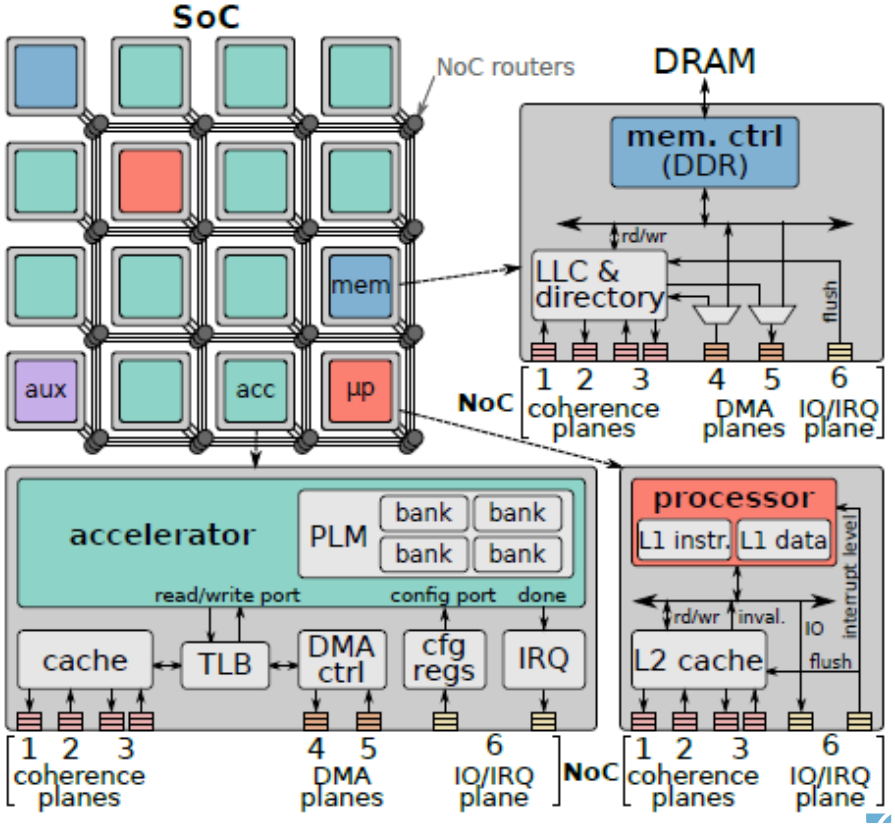
Example: NoC Services to Support Heterogeneous Cache-Coherence Models for Accelerators

- Seamless dynamic support for 3 coherence models:
  - Fully coherent accelerators
  - Non-coherent accelerators
  - Last-Level-Cache (LCC) coherent accelerators



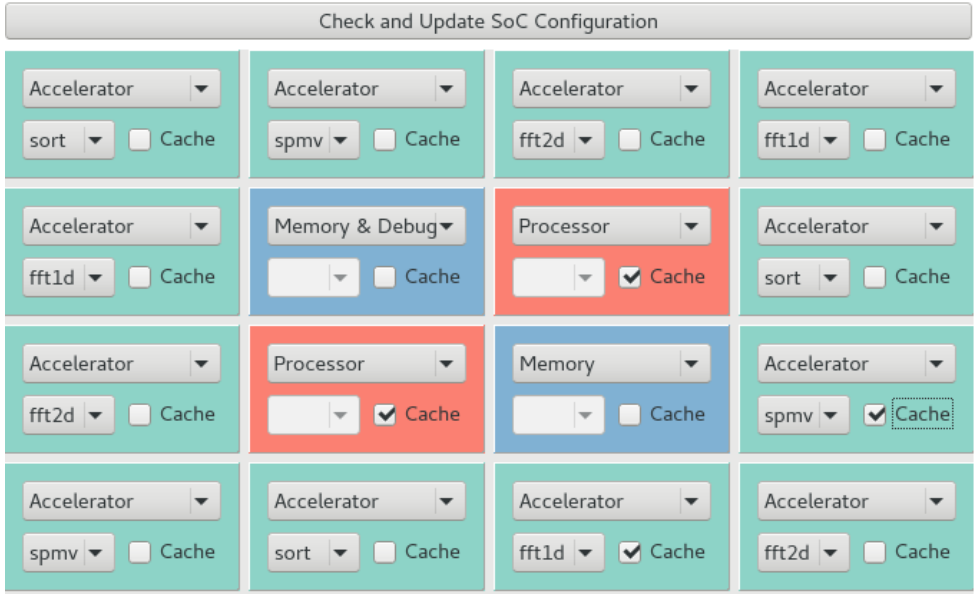
[D. Giri, P. Mantovani, and L. P. Carloni, *Accelerators & Coherence: An SoC Perspective*. IEEE MICRO, 2018.]

Extending ESP to Support Heterogeneous Cache-Coherence Models for Accelerators



- First NoC-based system enabling the three models of coherence for accelerators to coexist and operate simultaneously through run-time selection in the same SoC
  - design based on ESP Platform Services
- Extension of the MESI directory-based protocol to integrate LLC-coherent accelerators into an SoC
  - the design leverages the tile-based architecture of ESP to guarantee scalability and modularity

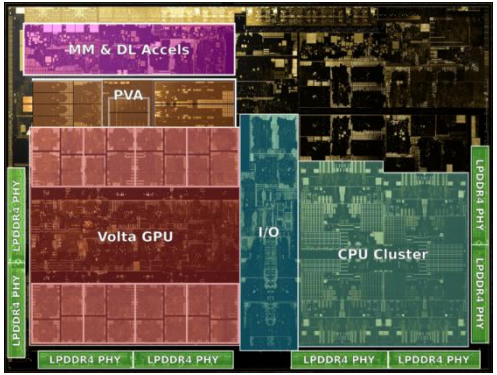
Heterogeneous Coherence Implementation



[D. Giri, P. Mantovani, L. P. Carloni, *Accelerators & Coherence: An SoC Perspective*, IEEE Micro, Nov/Dec 2018]

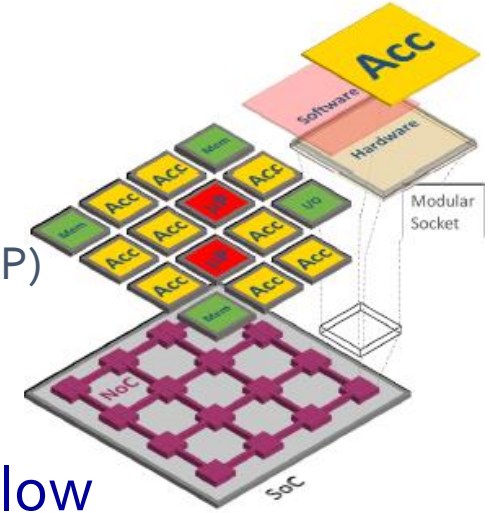
- **The CAD Infrastructure of ESP allows**
  - direct instantiation of heterogeneous configurable components from predesigned libraries
  - fully automated flow from the GUI to the bitstream for FPGAs
- **Extension of ESP to support atomic test-and-set and compare-and-swap operations over the NoC allows**
  - running multi-processor and multi-accelerator applications on top of Linux SMP

Outline

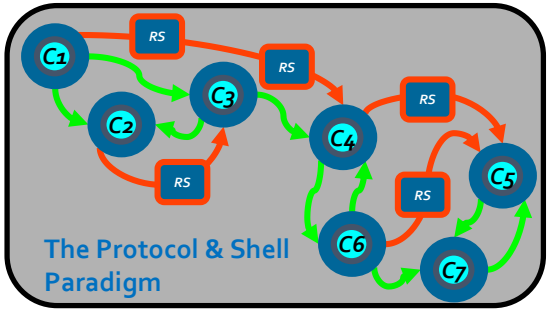


1. Motivation
   - The Rise of Heterogeneous Computing

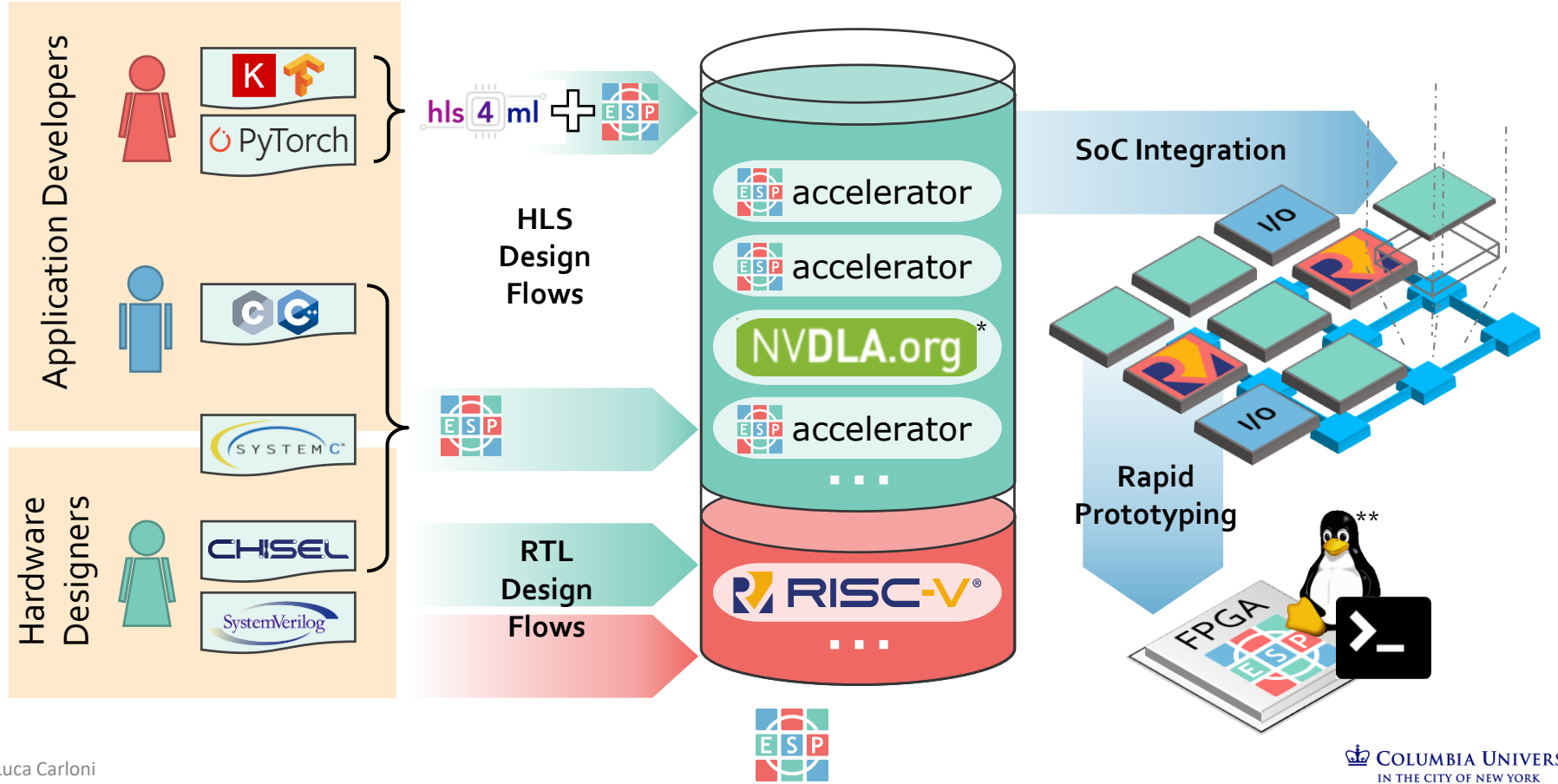
2. Proposed Architecture
   - Embedded Scalable Platforms (ESP)



3. Methodology and Design Flow
   - with a Retrospective on Latency-Insensitive Design



ESP Vision: Domain Experts Can Design SoCs



* By Nvidia Corporation
** By lewing@isc.tamu.edu Larry Ewing and The GIMP

Our System-Level Design Approach to Heterogeneous Computing: Key Ingredients

- **Develop Platforms, not just Architectures**
  - A platform combines an architecture and a companion design methodology
- **Raise the level of abstraction**
  - Move from RTL Design to System-Level Design
  - Move from Verilog/VHDL to high-level programming languages like SystemC
  - Move from ISA and RTL simulators to Virtual Platforms
  - Move from Logic Synthesis to High-Level Synthesis (both commercial and in-house tools), which is the key to enabling rich design-space exploration
- **Adopt compositional design methods**
  - Rely on customizable libraries of HW/SW interfaces to simplify the integration of heterogeneous components
- **Use formal metrics for design reuse**
  - Synthesize Pareto frontiers of optimal implementations from high-level specs
- **Build real prototypes (both chips and FPGA-based full-system designs)**
  - Prototypes drive research in systems, architectures, software and CAD tools

ESP Methodology In Practice

automated
interactive
manual (opt.)
manual

Accelerator Flow

Generate accelerator

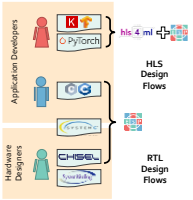
Specialize accelerator
* this step is automated
* for ML applications

Test behavior

Generate RTL

Test RTL

Optimize RTL



SoC Flow

Generate sockets

Configure RISC-V SoC

Compile bare-metal

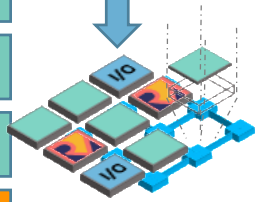
Simulate system

Implement for FGPA

Configure runtime

Compile Linux

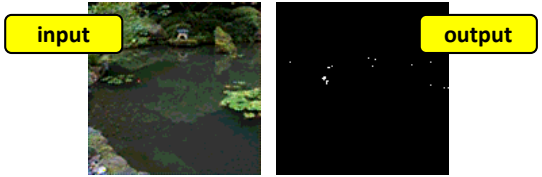
Deploy prototype

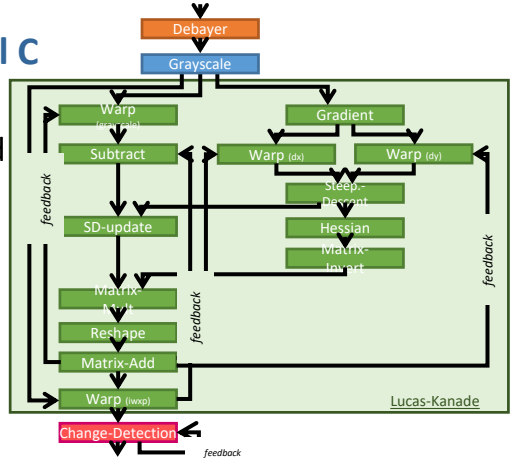


COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

ESP Design Example: An Accelerator for WAMI

- **The PERFECT WAMI-app is an image processing pipeline in behavioral C code**
  - From a sequence of frames it extracts masks of "meaningfully" changed pixels





  - Complex data-dependency among kernels
  - Computational intensive matrix operations
    - Global-memory access to compute ratio **45%**
    - Floating-point operation to compute ratio **15%**

- **We designed 12 accelerators starting from a C "programmer-view" reference implementation**
  - Methodology to port C into synthesizable SystemC
  - Automatic generation of customized RTL memory subsystems for each accelerator

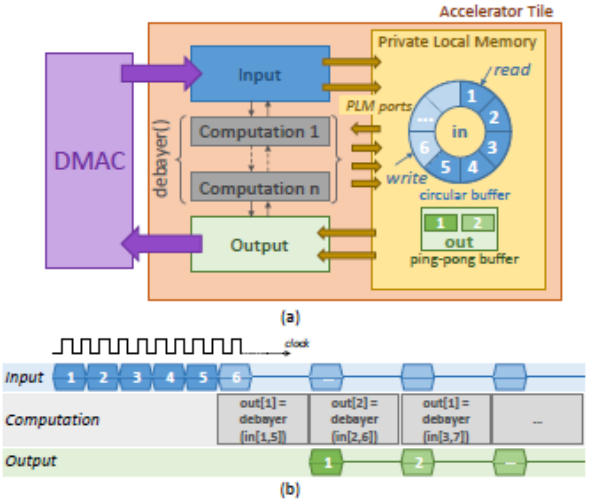
Lines of Code

Kernels	*C*	*SystemC*	*RTL*
Debayer	195	664	8440
Grayscale	21	368	4079
Warp	88	571	6601
Gradient	65	540	12163
Subtract	36	379	4684
Steep.-Descent	34	410	8744
SD-Update	55	383	7864
Hessian	43	358	7042
Matrix-Invert	166	388	7392
Matrix-Mult	55	307	2708
Reshape	42	269	2160
Matrix-Add	36	287	2310
Change-Detect.	128	939	18416
Total	**964**	**5863**	**92603**

[P. Mantovani, G. Di Guglielmo, and L. P. Carloni, High-Level Synthesis of Accelerators in Embedded Scalable Platforms, ASPDAC 2016]

Example of Accelerator Design with HLS: Debayer - 1

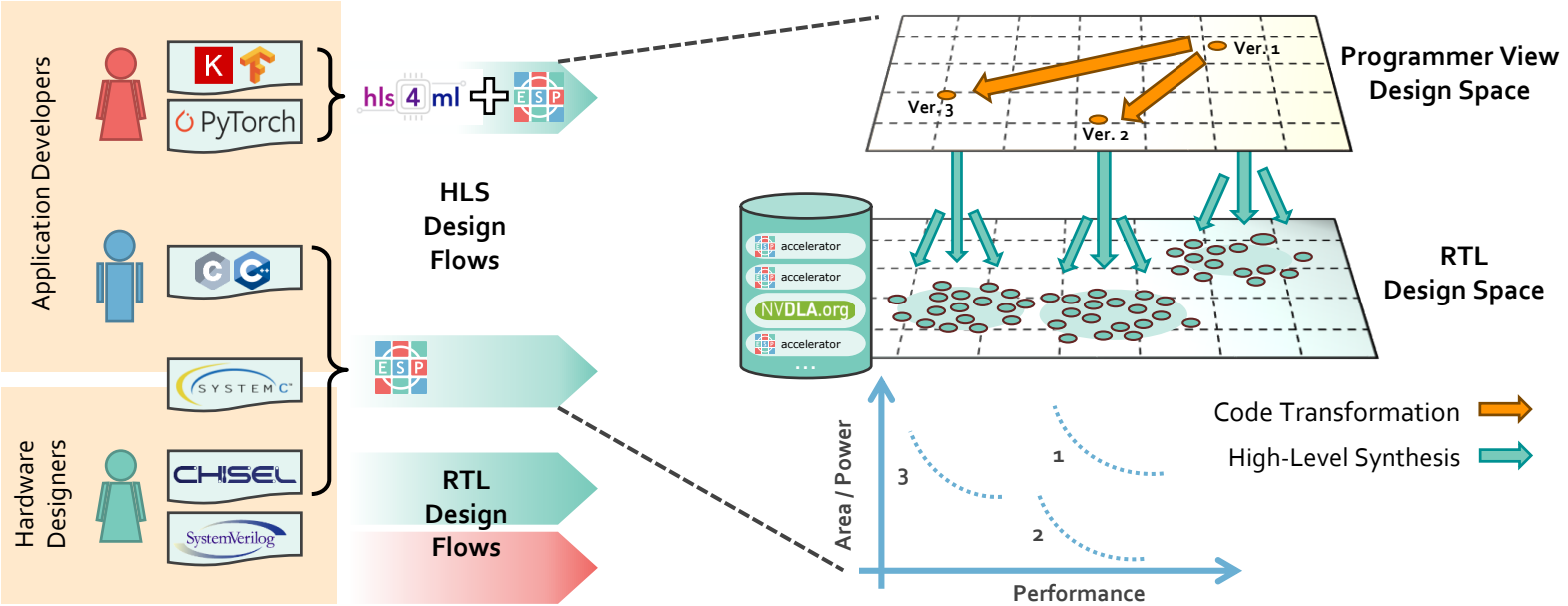
```
#include <systemc.h>
SC_MODULE(Debayer) {
 sc_in<bool> clk, rst;
private:
 sc_signal<bool> i_valid, i_ready, o_valid, o_ready;
 int A0[6][2048];  // circular buffer
 int B0[2048];
 int B1[2048];
public:
 //...
 SC_CTOR(debayer) {
  SC_CTHREAD(input, clk.pos());
  reset_signal_is(rst, false);
  SC_CTHREAD(compute, clk.pos());
  reset_signal_is(rst, false);
  SC_CTHREAD(output, clk.pos());
  reset_signal_is(rst, false);
  //...
 }
 void input(void) {
  // reset ...
  unsigned circ = 0; // circular buffer write pointer
  wait();
  while(true) {
   L0: for (int r=0; r<2048; r++) {
    // DMA request
    // read input ...
    L1:  for (int c=0; c<2048; c++)
     { A0[circ][c] = f(...); } //write to A0
    // output ...
    if (r >=5){
      // wait for ready from compute then notify as valid
    }
    circ++;
    if (circ == 6)
     circ = 0;
   }
  }
 }
 void compute(void) {
  int PAD = 2; bool flag = true;
  int r_r = 0; // central row of the mask
  // reset ...
  wait();
  while(true) {
   L2: for (int r=0; r<2048-PAD; r++) {
    // (wait for valid from input then notify as ready)
    r_r = circ_buffer_row(r + 2);
    L3:  for (int j=PAD; j<2048-PAD; j++) {
     if (flag) B0[j] = g(A0[r_r][j-2], A0[r_r][j-1],
        A0[r_r][j], A0[r_r][j+1], A0[r_r][j+2], ...);
     else B1[j] = g(A0[r_r][j-2], A0[r_r][j-1],
        A0[r_r][j], A0[r_r][j+1], A0[r_r][j+2], ...);
     }
    // (valid to output, ready to compute)
    flag = !flag;
   }
  }
 }
 void output(void) {
  int PAD = 2; bool flag = true;
  // reset ...
  wait();
  while(true) {
   L4: for (int r=PAD; r<2048-PAD; r++) {
    // (wait for valid from compute then notify as ready)
    // prepare DMA request
    // send data
    L5:  for (int c=PAD; c<2048-PAD; c++) {
     if (flag) h(B0[c], ...);  //read from array B0
     else h(B1[c], ...);       //read from array B1
    }
    // (ready to compute)
    flag = !flag;
   }
  }
 }
};
```



- The 3 processes execute in pipeline
  - on a 2048×2048-pixel image, which is stored in DRAM, to produce the corresponding debayered version
- The circular buffer allows the reuse of local data, thus minimizing the data transfers with DRAM
- The ping-pong buffer allows the overlapping of computation and communication

ESP Accelerator Flow

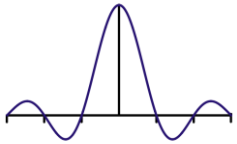
Developers focus on the **high-level specification**, **decoupled** from memory access, system communication, hardware/software interface



Example of Design-Space Exploration with HLS: Accelerator for the SAR Interp-1 Kernel

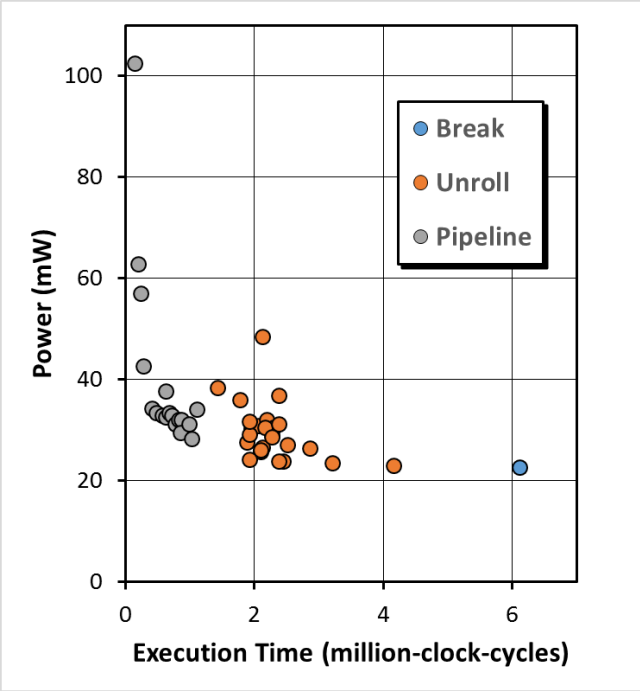
Main loop in *Interpolation-1* kernel

```
function interp1()
{
    for(...)
    {
        accum = 0;
        for(...)
        {
            accum += sinc(input);
        }
        store(accum);
    }
}
```



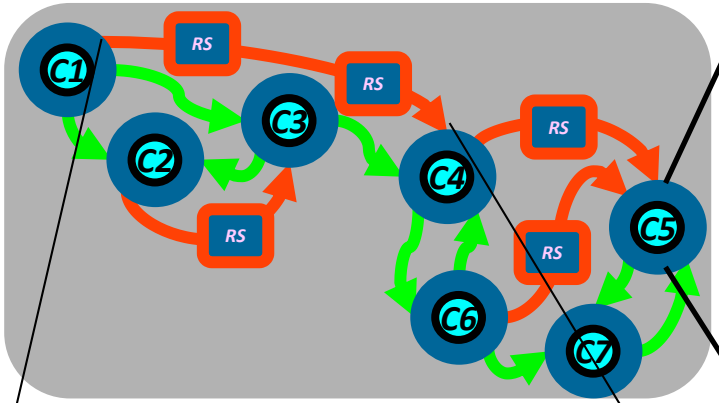
- Presence of expensive combinational function (sinc()) in the inner most loop
- Use of "loop knobs" provided by HLS tools to optimize for power and performance
- Derivation of Pareto set highlighting Power-Performance trade-offs

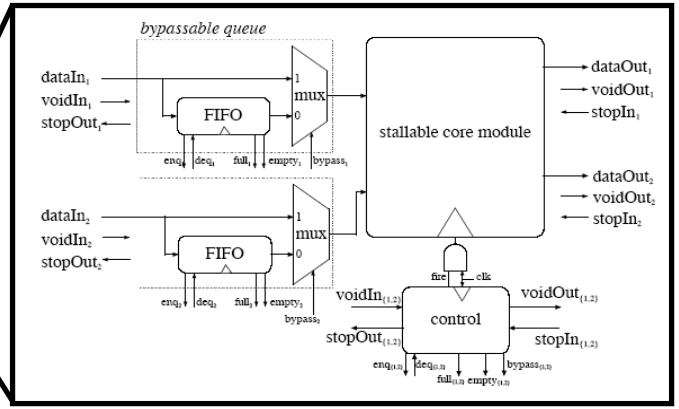
Pareto Set Obtained with High-Level Synthesis (1GHz@1V, CMOS 32nm)

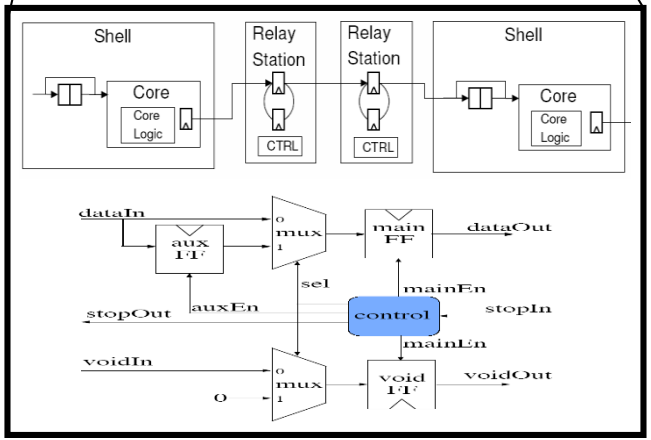


Retrospective: Latency-Insensitive Design

[Carloni et al. ’99]



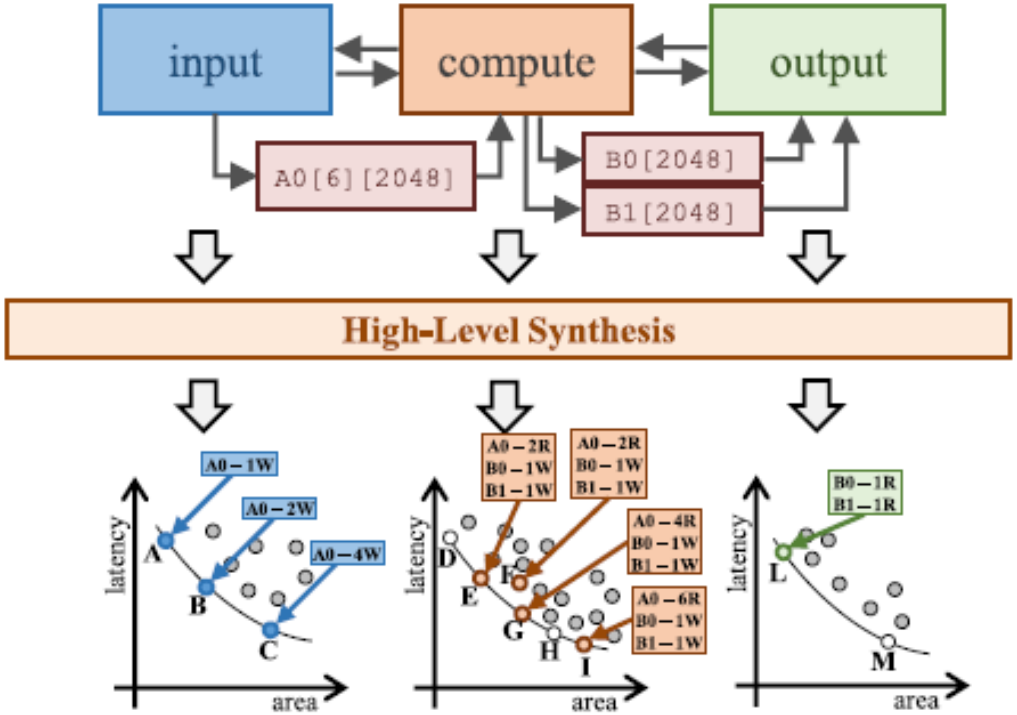




Latency-Insensitive Design

- 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

Example: Combining LID and HLS in the Design of the Debayer Accelerator



- **The combination of the ESP interface and the latency-insensitive protocol enable a broad HLS-supported design-space exploration**
- **For example, for the compute process**
  - Implementation E is obtained by unrolling loop L3 for 2 iterations, which requires 2 concurrent memory-read operations
  - Implementation F is obtained by unrolling L3 for 4 iterations to maximize performance at the cost of more area, but with only 2 memory-read interfaces; this creates a bottleneck because the 4 memory operations cannot be all scheduled in the same clock cycle
  - Implementation G, which Pareto-dominates implementation F, is obtained by unrolling L3 for 4 iterations and having 4 memory-read interfaces to allow the 4 memory-read operations to execute concurrently

[C. Pilato, P. Mantovani, G. Di Guglielmo, and L. P. Carloni, System-Level Optimization of Accelerator Local Memory for Heterogeneous Systems-on-Chip, TCAD '17]

ESP4ML

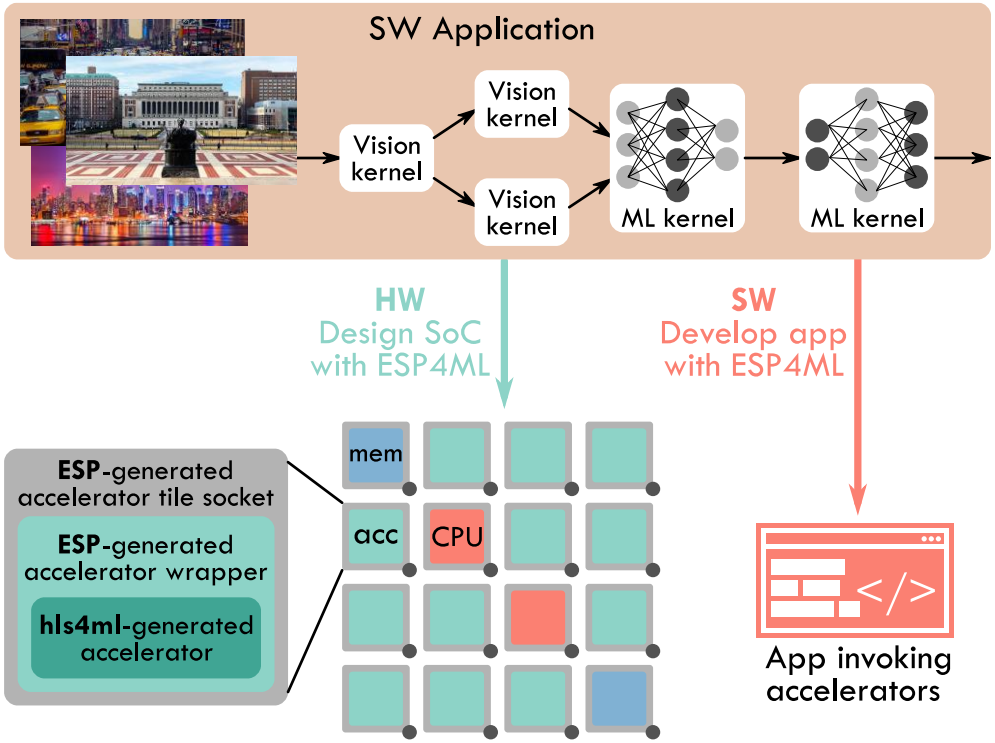
Open-source design flow to build and program SoCs for ML applications.

Combines  and hls 4 ml

- **ESP** is a platform for heterogeneous SoC design
- **hls4ml** automatically generates accelerators from ML models

Main **contributions to ESP**:

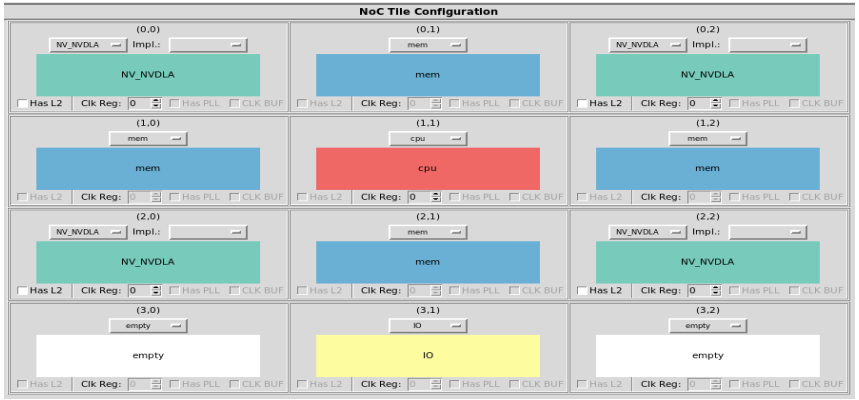
- Automated integration of hls4ml accelerators
- Accelerator-accelerator communication
- Accelerator invocation API

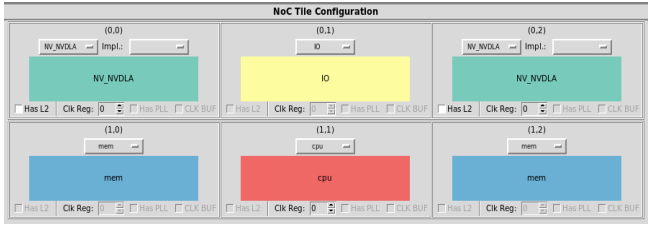


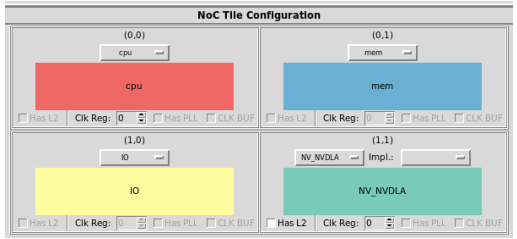
[D. Giri, K.-L. Chiu, G. Di Guglielmo, P. Mantovani, and L. P. Carloni. "ESP4ML: Platform-Based Design of Systems-on-Chip for Embedded Machine Learning", DATE '20]

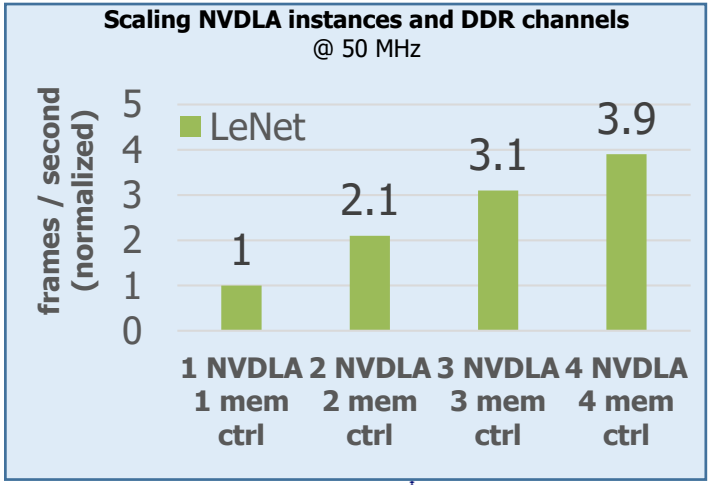
Seamless Integration of Third-Party Accelerators

- New design flow of general applicability
  - demonstrated w/ NVIDIA NVDLA
- Transparent accelerator integration
  - original software apps can run "as is"
- Linear performance scalability
  - when scaling up NVDLA instances with DDR channels



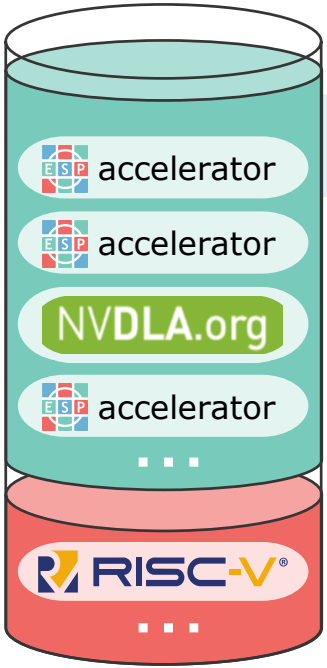




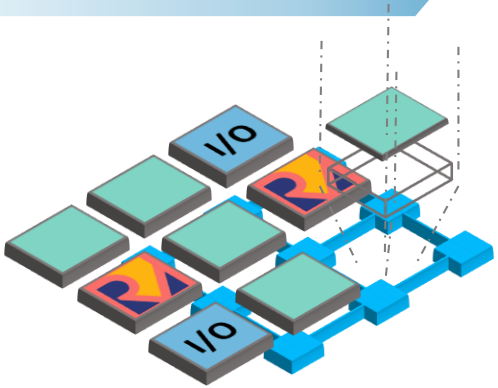


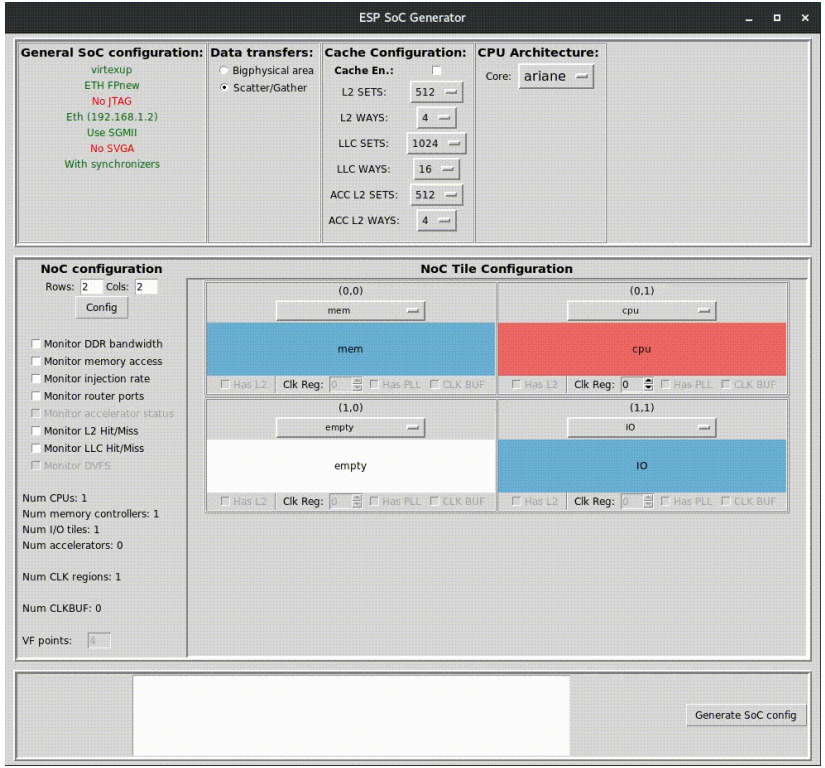
[D. Giri et al. "Ariane + NVDLA: Seamless Third-Party IP Integration with ESP", CARRV'20]

ESP Interactive SoC Flow



SoC Integration

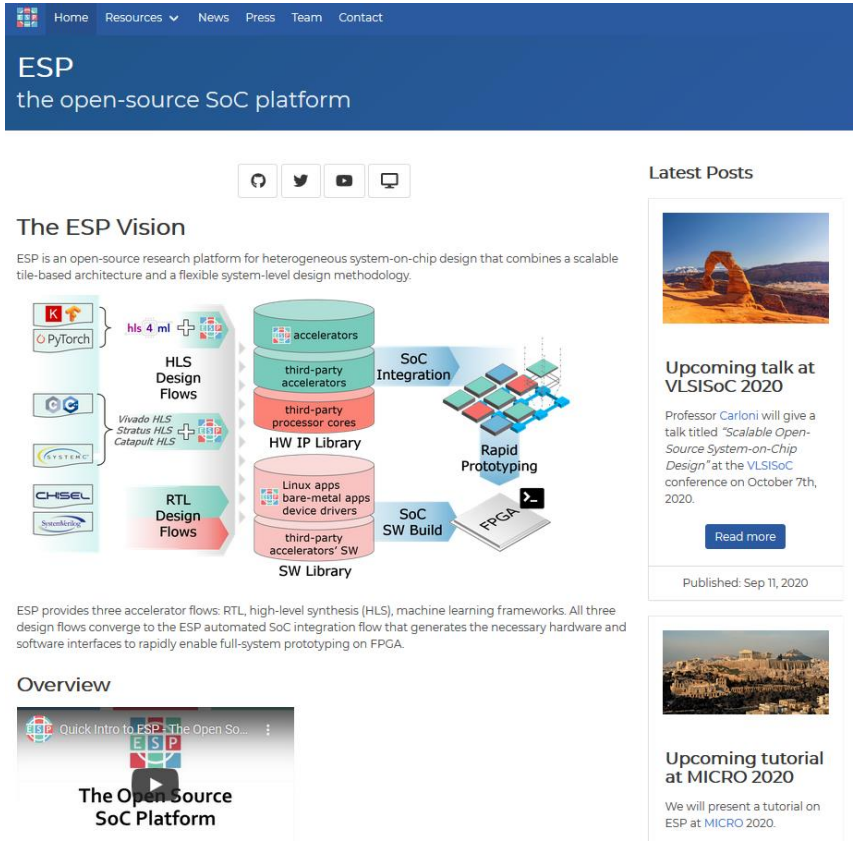




In Summary: ESP for Open-Source Hardware

- 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

COMPUTER SCIENCE

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK