

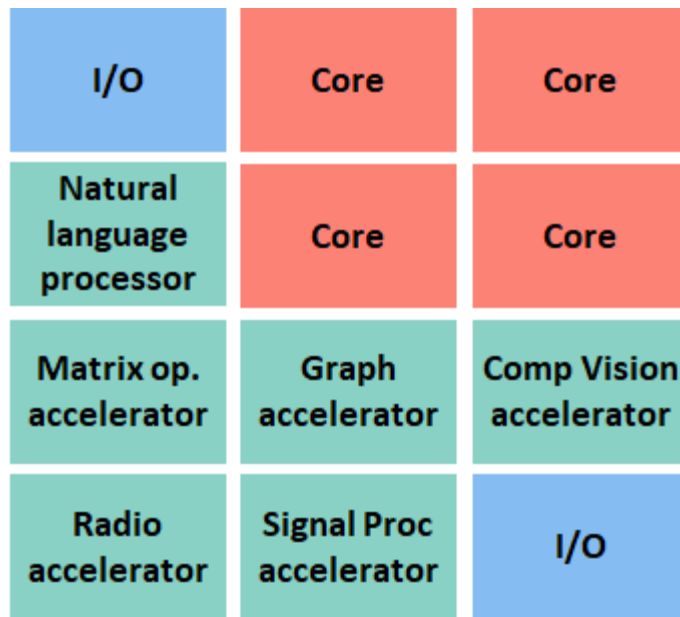
# Presenting for OSCAR 2023: Pipelining an Open-Source Last-Level Cache

Kevin Jiang, Joseph Zuckerman, and Luca P. Carloni

# Motivation

- SoCs are increasingly heterogeneous and complex
- On-chip shared memory can reduce memory access time and simplify programming in SoCs
- SoCs utilize cache hierarchies to enforce memory coherency of on-chip shared memory across the entire system

The **performance of the cache hierarchy** is crucial to **reducing memory access time** in an SoC



# ESP: An Open-Source Platform for SoC design

[www.esp.cs.columbia.edu](http://www.esp.cs.columbia.edu)

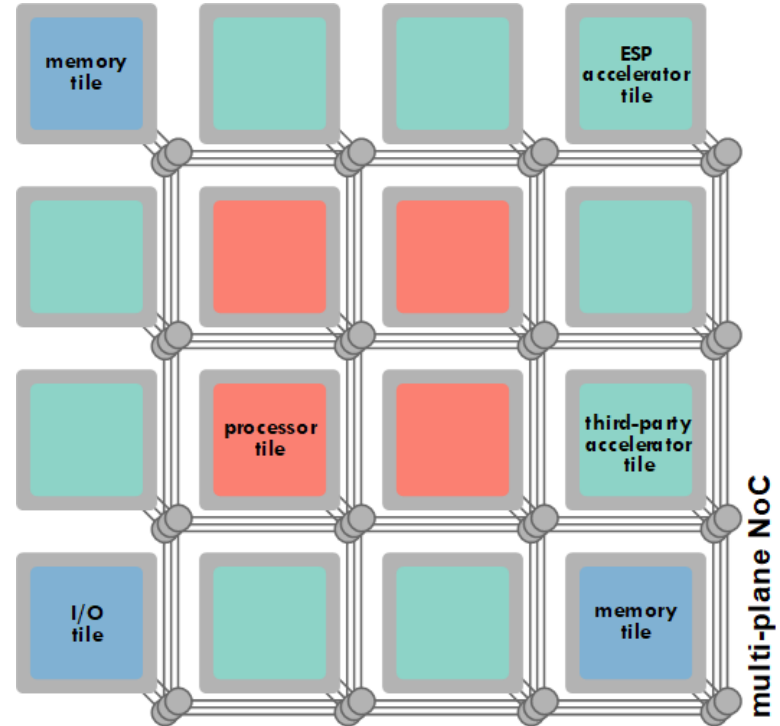
- ESP combines a flexible architecture with automated IP integration and a large variety of accelerator design flows to provide a platform for rapid SoC design and prototyping
- ESP also provides a cache hierarchy for implementing on-chip shared memory

The screenshot displays the ESP website interface. At the top, there is a navigation bar with links for Home, Release, Resources, News, Press, Team, and Contact. Below this is the main header with the ESP logo and the tagline "the open-source SoC platform". A secondary navigation bar includes icons for home, social media, and other site functions. The main content area features a "News" section with a link to an upcoming tutorial at ASPLOS 2021. Below this is a section titled "The ESP Vision" which includes a diagram illustrating the design flow from various design flows (HLS, RTL) through a library of accelerators and IP to SoC integration and rapid prototyping on an FPGA. The diagram shows three parallel design flows: HLS Design Flows (using PyTorch, Vivado HLS, and Catapult HLS), RTL Design Flows (using C-ESL and SystemC), and a SW Library (containing Linux apps, bare-metal apps, device drivers, and third-party accelerators' SW). These flows converge into a "SoC Integration" step, followed by "Rapid Prototyping" and "SoC SW Build" leading to an "FPGA" target. A text block below the diagram explains that ESP provides three accelerator flows (RTL, high-level synthesis (HLS), machine learning frameworks) that converge to an automated SoC integration flow for full-system prototyping on FPGA. To the right of the main content is a "Latest Posts" sidebar with two entries: "Upcoming tutorial at ASPLOS 2021" (published Apr 2, 2021) and "Release 2021.1.0" (published Jan 26, 2021). The ESP logo is also visible in the sidebar.

# 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



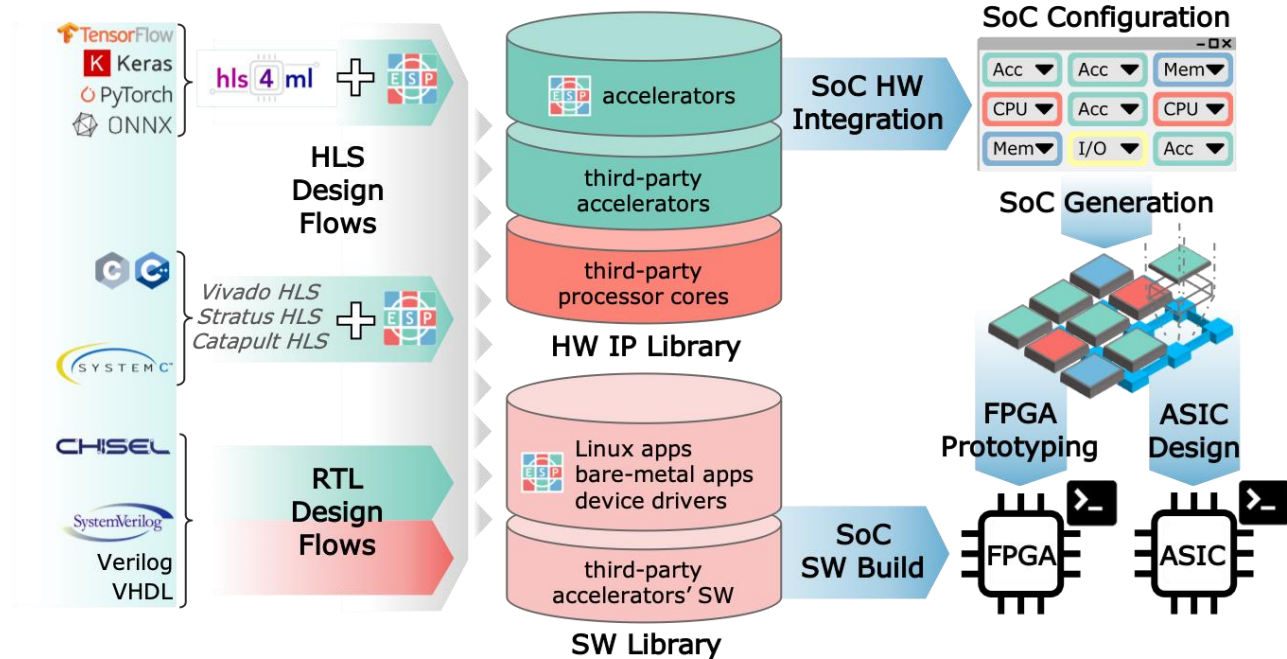
# ESP Methodology

## Accelerator Flow

- Simplified design
- Automated integration

## SoC Flow

- Mix & match floor-planning GUI
- Rapid FPGA prototyping



# ESP Methodology

## SoC Flow

- Mix & match floor-planning GUI
- Rapid FPGA prototyping

The screenshot displays the ESP SoC Generator interface, which is organized into several configuration panels:

- General SoC configuration:** Includes options like virtexp, ETH FPnew, No JTAG, Eth (192.168.1.2), Use SGMII, No SVGA, and With synchronizers.
- Data transfers:** Offers choices between Bigphysical area and Scatter/Gather.
- Cache Configuration:** Allows setting Cache En., L2 SETS (512), L2 WAYS (4), LLC SETS (1024), LLC WAYS (16), ACC L2 SETS (512), and ACC L2 WAYS (4).
- CPU Architecture:** Shows the selected Core as 'ariane'.
- NoC configuration:** Features a 'Config' button and a list of monitoring options such as Monitor DDR bandwidth, Monitor memory access, Monitor injection rate, Monitor router ports, Monitor accelerator status, Monitor L2 Hit/Miss, Monitor LLC Hit/Miss, and Monitor DVFS. It also displays summary statistics: Num CPUs: 1, Num memory controllers: 1, Num I/O tiles: 1, Num accelerators: 0, Num CLK regions: 1, Num CLKBUF: 0, and VF points: 4.
- NoC Tile Configuration:** A 2x2 grid of tiles with coordinates (0,0) to (1,1). The tiles are: (0,0) mem (blue), (0,1) cpu (red), (1,0) empty (white), and (1,1) IO (blue). Each tile has associated options for Has L2, Clk Reg, Has PLL, and CLK BUF.
- Generate SoC config:** A button located at the bottom right of the interface.

# ESP cache hierarchy

- Consists of private L2 caches and a LLC (Last Level Cache)
- Extended MESI directory-based cache coherence protocol
- LLC maintains coherence between L2 caches
- **Processor cores** have L2 caches provided by ESP
- **Accelerators** typically perform DMA to DRAM, but can also interface with LLC and optionally have L2 cache
- In ESP, accelerators can operate under **4 coherence modes**:
  - Non-coherent DMA: No L2 cache, DMA to DRAM only
  - LLC-coherent DMA: No L2 cache, DMA to LLC with coherence enforced by software flush
  - Coherent DMA: No L2 cache, DMA to LLC with coherence enforced by protocol
  - Fully Coherent: Accelerator has L2 cache just like processor cores

# ESP cache coherence protocol

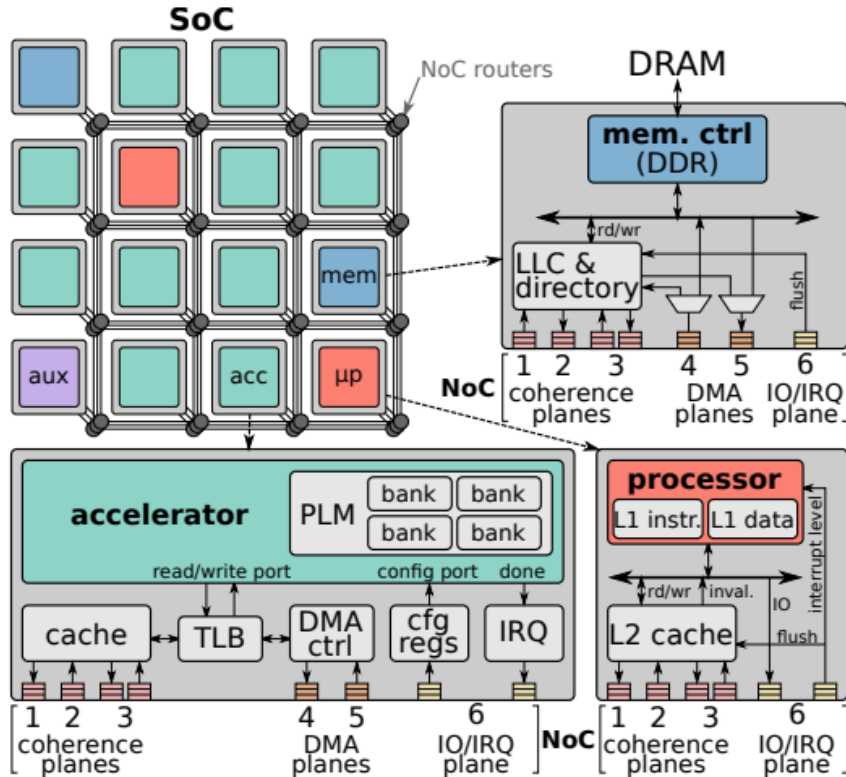
- Extended MESI directory-based cache coherence protocol
- Cache line states: Modified, Exclusive, Shared, Invalid, (Valid)

TABLE I  
DIRECTORY CONTROLLER'S EXTENDED MESI PROTOCOL.

	REQUESTS					DMA REQUESTS		RESPONSES	
	GetS	GetM	PutS	PutM	Evict	Read	Write	Inv-Ack	Data
I	read mem. Excl. Data to req, owner = req / E	read mem. Data to req, owner = req / M	Put-Ack to req	Put-Ack to req		read mem, Data to req / V	[read mem], write LLC, / V		
V	Excl. Data to req, owner = req / E	Data to req, owner = req / M	Put-Ack to req	Put-Ack to req	[write mem] / I	Data to req	write LLC		
S	Data to req, sharers += req	Data to req, Inval. to sharers, owner = req, clear sharers / M	Put-Ack to req, sharers -= req / V (if last sharer)	Put-Ack to req, sharers -= req / V (if last sharer)	[write mem], Inval. to sharers, clear sharers / I				
E	Fwd-GetS to owner, sharers+=req+owner, clear owner / S <sup>D</sup>	Fwd-GetM to owner, owner = req / M	Put-Ack to req, if req is owner: - clear owner / V	write LLC, Put-Ack to req, if req is owner: - clear owner / V	Fwd-GetM to owner, clear owner / E I <sup>D</sup>				
M	Fwd-GetS to owner, sharers+=req+owner clear owner / S <sup>D</sup>	Fwd-GetM to owner, owner = req	Put-Ack to req	write LLC, Put-Ack to req, if req is owner: - clear owner / V	Fwd-GetM to owner, clear owner / M I <sup>D</sup>				
S <sup>D</sup>	stall	stall	Put-Ack to req, sharers -= req	Put-Ack to req, sharers -= req	stall				write LLC, / V (if no sharers), / S (otherwise)
E I <sup>D</sup>	stall	stall	Put-Ack to req, sharers -= req	Put-Ack to req, sharers - = req				[write mem] / I	write mem / I
M I <sup>D</sup>	stall	stall	Put-Ack to req, sharers -= req	Put-Ack to req, sharers -= req					write mem / I



# ESP cache hierarchy example



- 4x4 tile ESP system
- Processor tiles have off-the-shelf L1 cache
- ESP provides L2 caches for Processor tiles and optionally Accelerator tiles
- Memory tiles contain LLC
- Cache hierarchy is connected via a multi-plane NoC (Network-on-Chip)

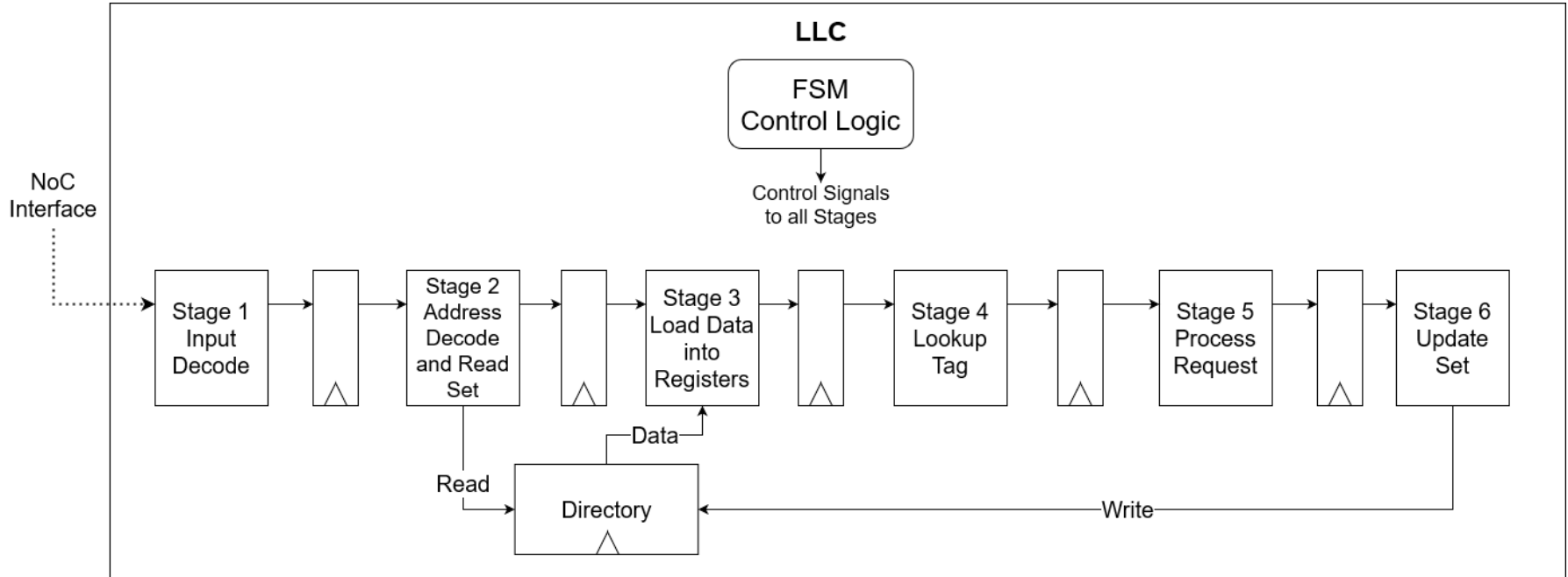
# Improving LLC throughput

- LLC is the **main synchronization point** for the SoC
  - All L2 caches must interface with LLC
- Throughput of the LLC can limit performance of SoC when **density of requests to the LLC is high**
- The current LLC implementation utilizes a multi-cycle data path, only handling one request per multi-cycle iteration

We implement a **pipelined data path** to increase the throughput of the LLC

# LLC Microarchitecture without pipelining

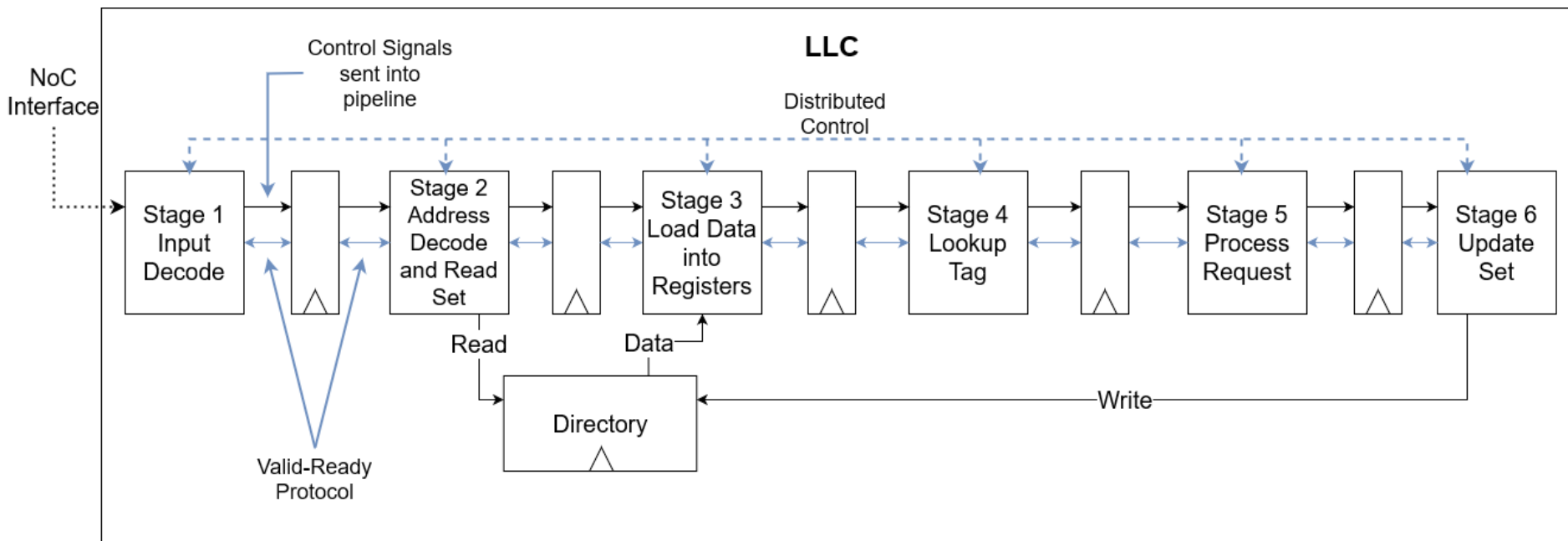
6-stage multi-cycle datapath controlled by FSM unit



# Pipelined LLC: Step 1

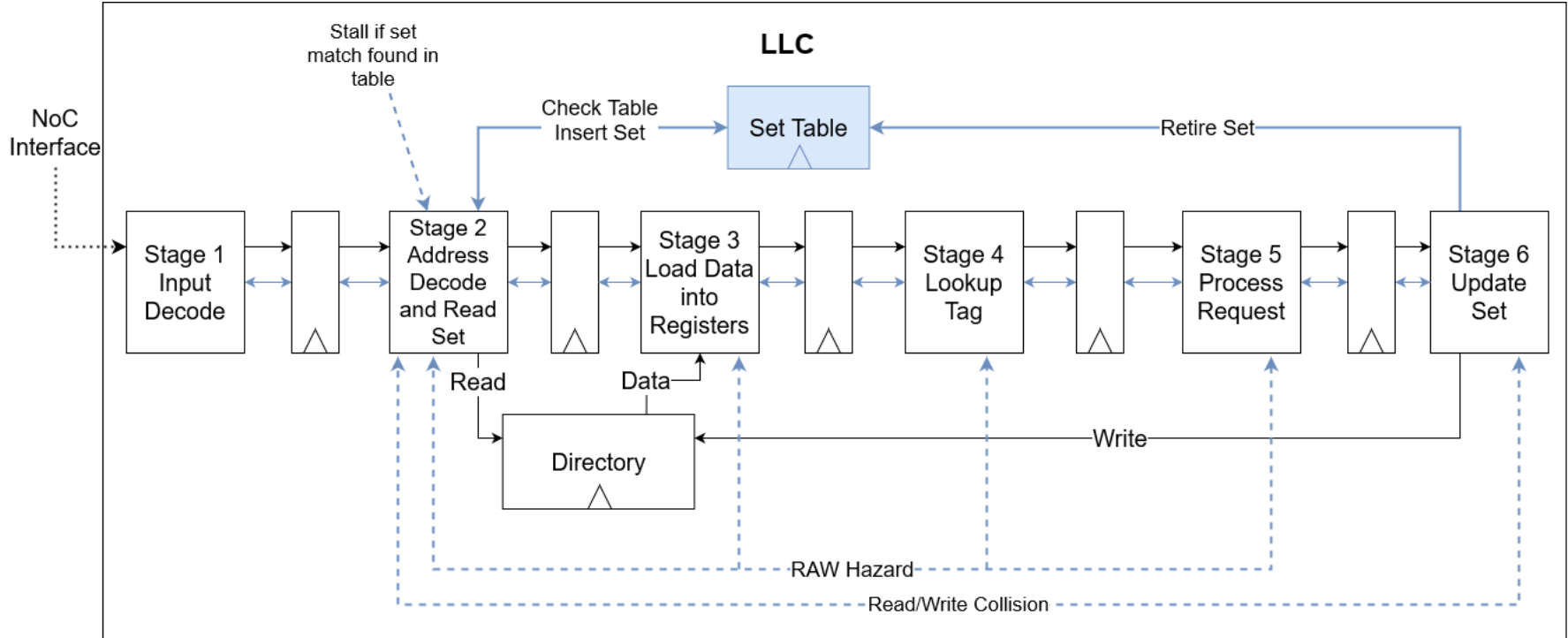
Distribute control logic across all stages

Implement valid-ready protocol pipeline registers



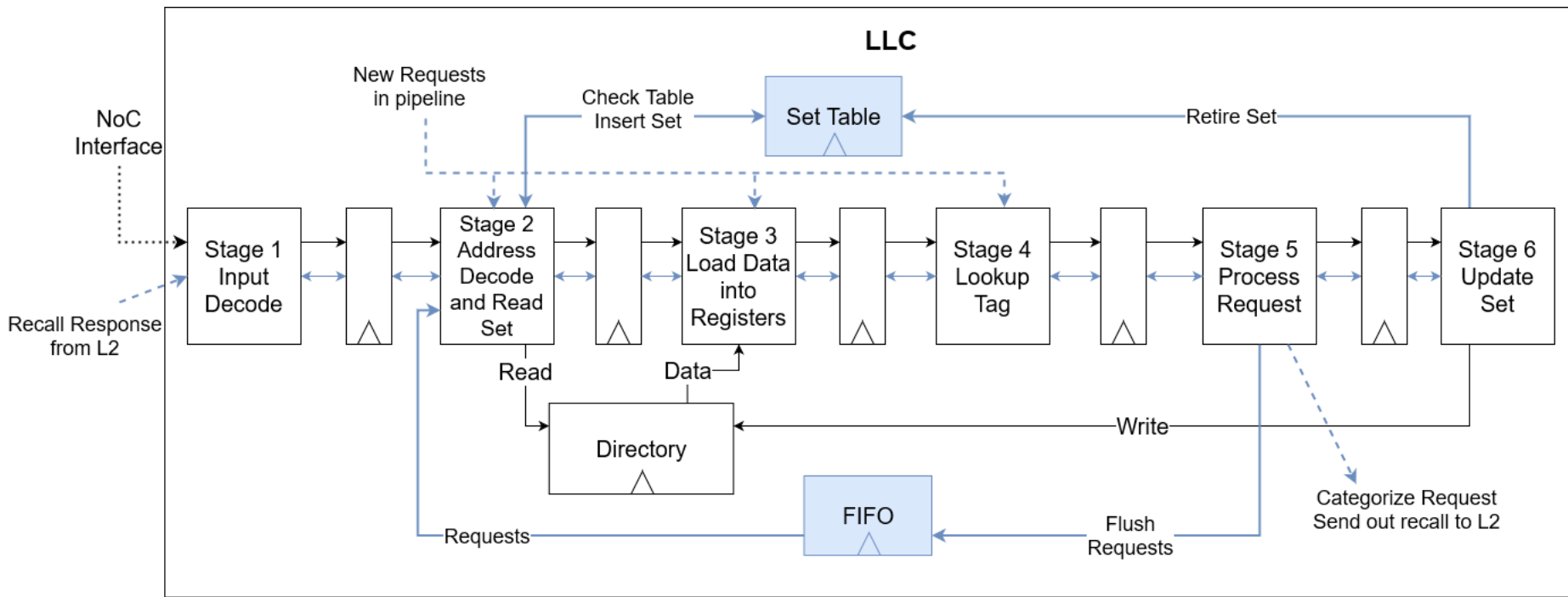
# Pipelined LLC: Step 2

Elimination of read-after-write hazards and read/write collisions



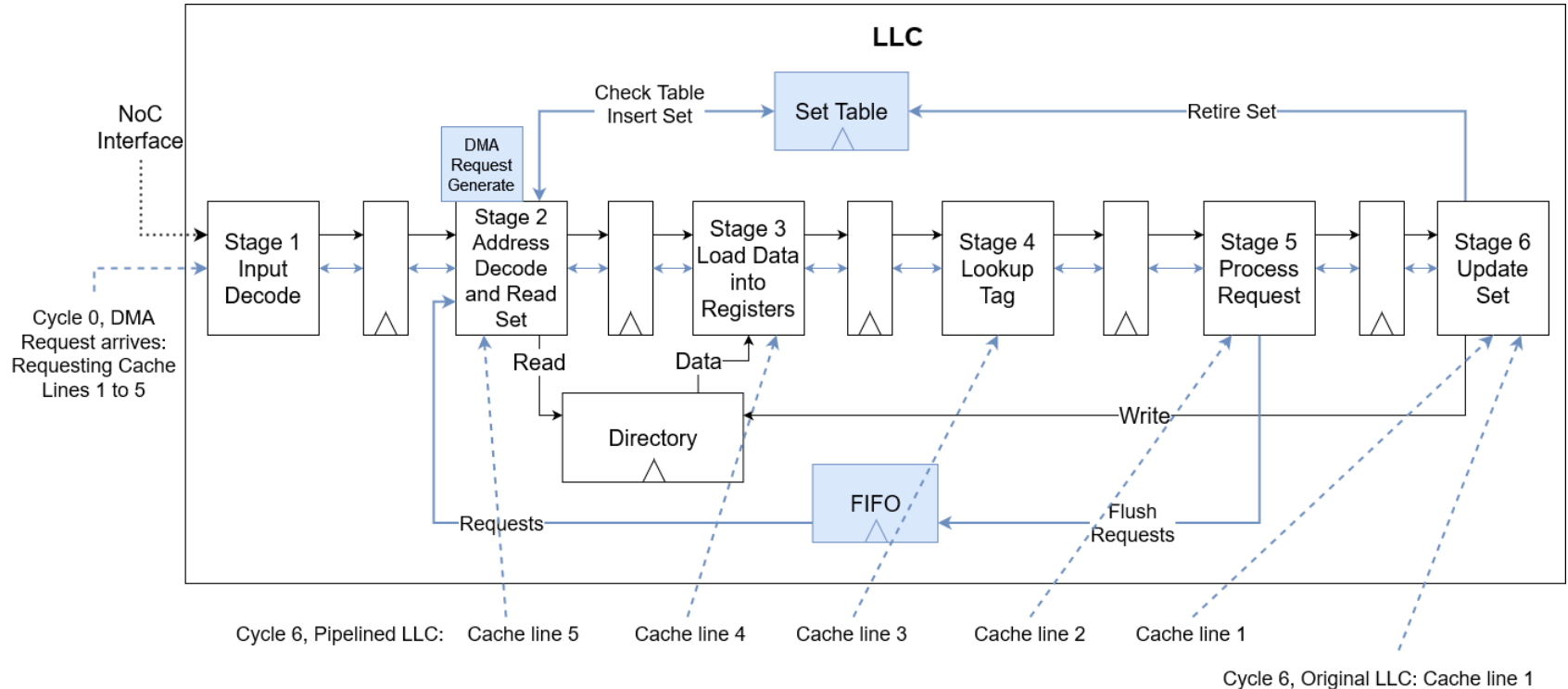
# Pipelined LLC: Step 3

Prevent out-of-order completion of requests

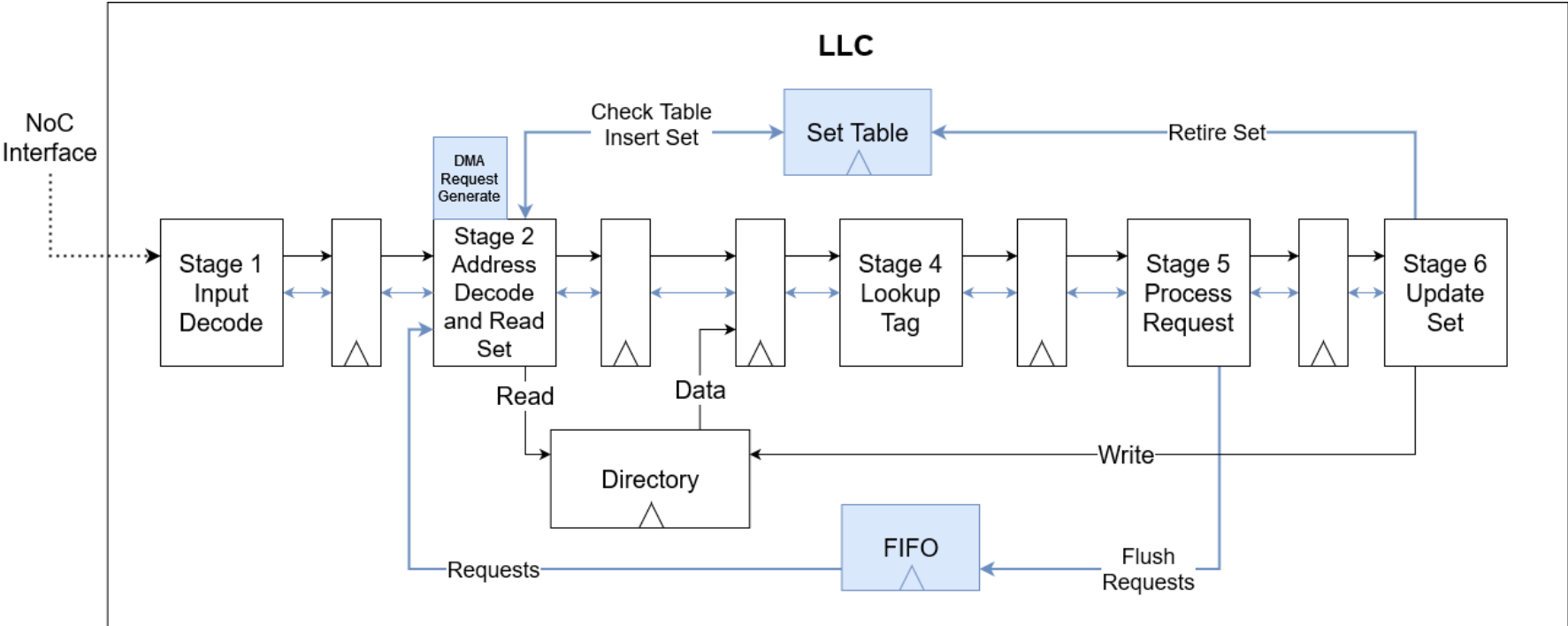


# Pipelined LLC: Step 4

Increase pipeline utilization of DMA requests



# LLC Pipelined Microarchitecture

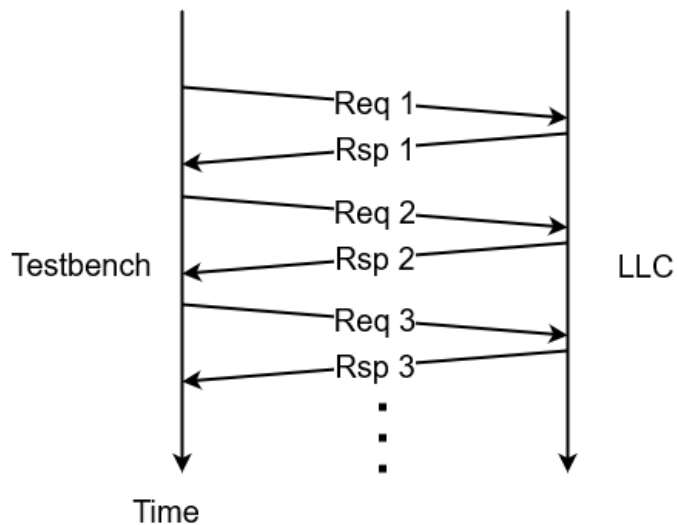




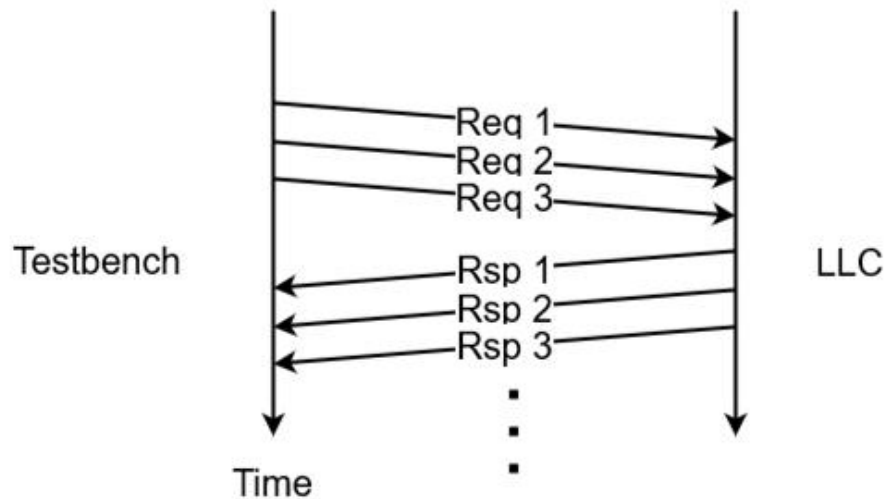
# Verification: RTL simulation for LLC

## SystemVerilog testbenches for RTL simulation

Large coverage test for basic functionality

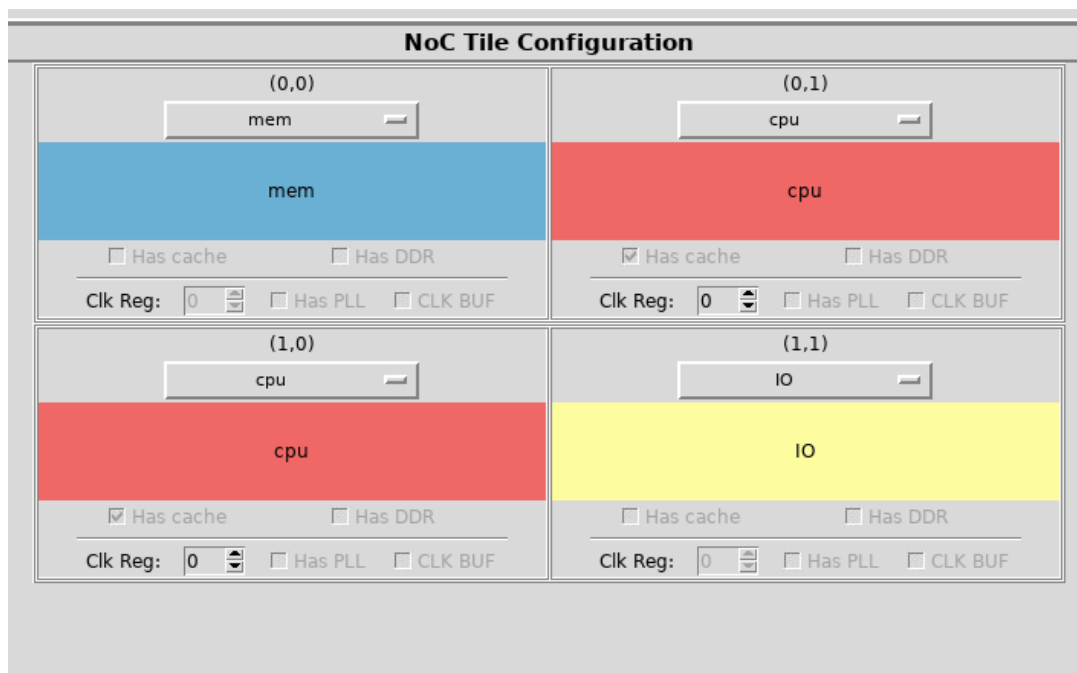


Small coverage test for pipeline functionality



# Verification: RTL simulation for full ESP system

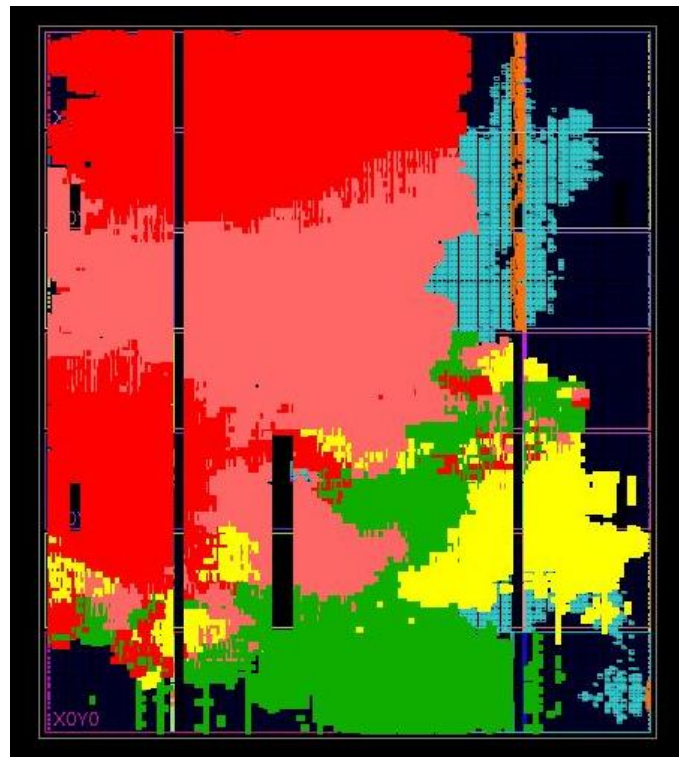
Single-core and multi-core simulation of basic “Hello World” program



# Verification: FPGA testing

Implementing SoC on FPGA and running applications

- Small applications:
  - Single-core and multi-core “Hello World”
  - Multi-core shared memory and lock program
- Booting Linux with Ethernet enabled on Single-core SoC
  - Ethernet uses coherent DMA to the LLC



Picture shows 2-core SoC. Red: CPU, Green: Memory Tile with LLC, Yellow: I/O Tile

# Performance Assessment: Method

Monitor memory access time of 3 different accelerator workloads, compare times when using LLC without pipelining and with pipelining

- FFT (Fast Fourier Transform)
- Matrix Multiplication (GEMM)
- 2D Convolution (CONV2D)

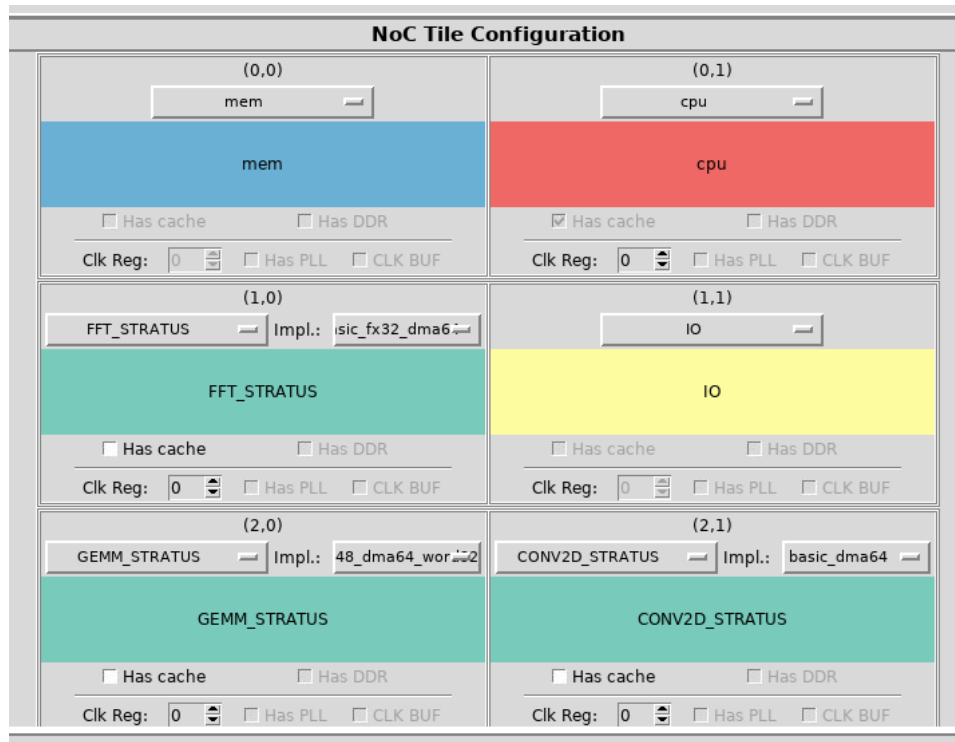
ESP Performance Monitors API allows monitoring of memory access time only

# Performance Assessment: FPGA Implementation

Implement two SoCs, one SoC with pipelined LLC, one SoC with original LLC

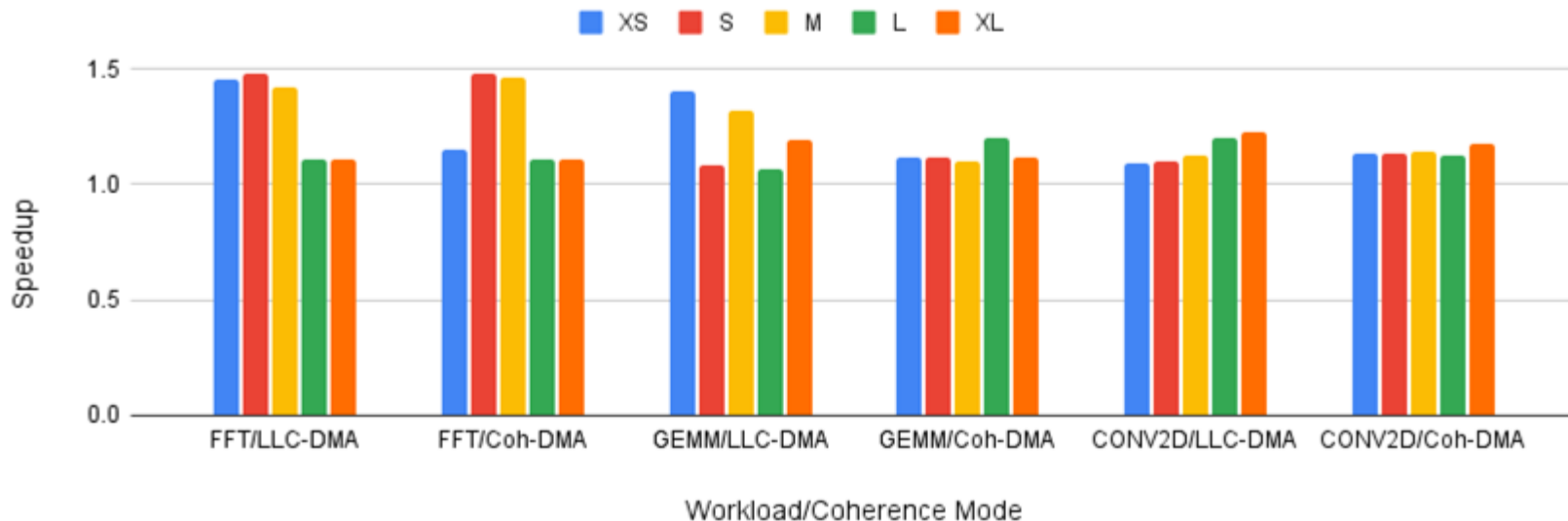
Workload variables:

- 5 different sizes from XS to XL
- 2 Coherence Modes
  - LLC-Coherent DMA: Accelerators access LLC after software flush of L2 caches for coherence
  - Coherent DMA: Accelerators access LLC with coherence enforced by hardware coherence protocol



# Performance Assessment: Speedup Results

- Speedup of memory access times on SoC with pipelined LLC compared to SoC with original LLC
- Speedup is as high as 50%, ranges from 10%-25%



# Conclusion

- We implemented pipelining in the ESP Last Level Cache, resolved pipelining hazards, optimized DMA performance, and enabled concurrent processing of LLC requests.
- We achieved significant speedup in memory access times of accelerator DMA, up to ~50% for some workloads with a consistent range of 10% to 25%, and maintained modularity and scalability of ESP

Verification is still in progress (booting Linux on multi-core configurations), but we plan to release a version of ESP with the improved cache hierarchy later this year!

# Thank you for listening!

[www.esp.cs.columbia.edu](http://www.esp.cs.columbia.edu)

