Best Practices in Design for FPGA or ASIC

Adam Sherer, Account Technical Executive, Cadence Design Systems

November 1, 2021

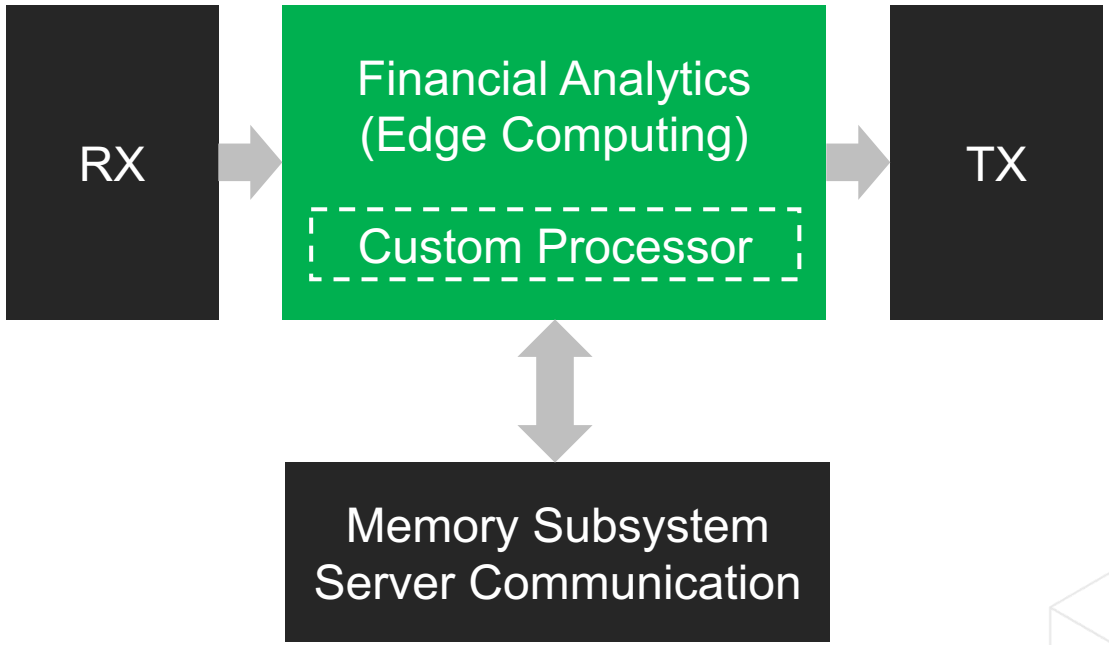
cadence®

Agenda

Be Fast and Nimble with Low Latency Flexible Architectures

The goal is to be out in front with both!

- Remove system latency
  - Speed RX/TX ethernet and memory subsystem but may require ASIC approach
- Add custom processor for flexibility
  - SW retargetable to better analytics or lower latency at the edge maintaining same HW
  - May need new IP and engineering resources
- Yet persistent bottlenecks can remain
  - Functionally correct but inefficiently coded
  - Efficiently coded but scalability/correctness occur

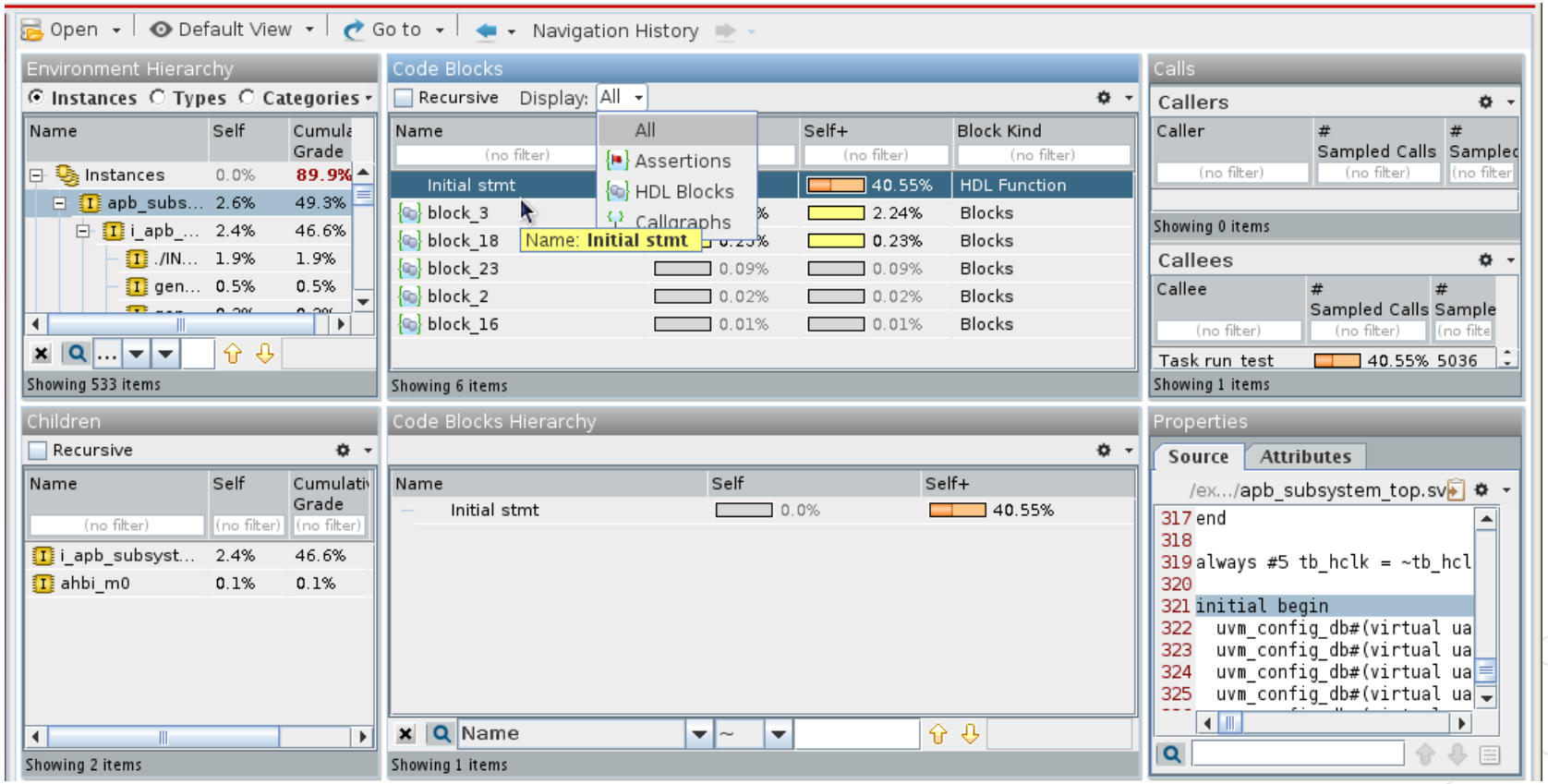


Coding Approaches for Low Latency

Point of Investment	Approach	Benefit
Architectural	• High-level, synthesizable design • Early analysis for latency bottlenecks • Effective digital twin for debug	• Detect systemic bottlenecks early • Design retargetable to FPGA or ASIC
Incremental	• Define coding styles for low-latency • Refactor elements of design to eliminate bottlenecks • Continuously analyze performance	• Continuously reduce latency within existing architectures
Debug	• Profile design to identify bottlenecks • Increase simulation and formal analysis use to resolve bottlenecks	• Achieve project confidence faster to get accelerator into market sooner

Identifying Potential Bottlenecks

- Lab: real-world data but limited visibility to internal bottlenecks
- Single simulation: high visibility into single tests
- Regression simulation: medium visibility across test suite
- Recommendation: Measure performance weekly



There are Too Many Delays!

- Localize data
  - Place register files and/or memories physically close to analytics to reduce data access time
- Channelize compute
  - Replicate analytics to reduce buffering and localize interrupts
- Multiple independent clock domains
  - Reduce synchronization points to enable parallel computing

- Note that increasing asynchronous compute creates risk of non-deterministic algorithmic execution
  - We'll come back to this point when discussing formal verification methods

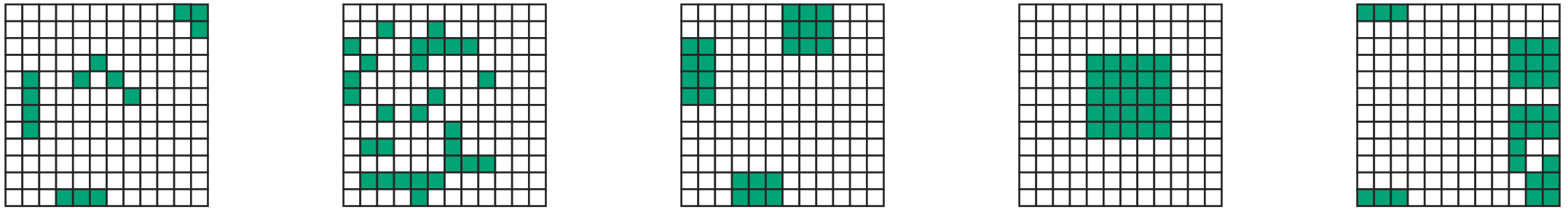
Apply Performance Best Practices

- Train engineers in common coding-for-performance approaches
- Eliminate loop invariants
  - Move constant computation ahead of loop
- Unravel loops
  - Ex: Implement as parallel computation with single summation
- Utilize incremental vs absolute calculations
  - Apply compute on just the data component used in the analytics

- Generally trade-off area for speed but keep an eye on physically long data paths

Multiple Verification Approaches Should be Applied

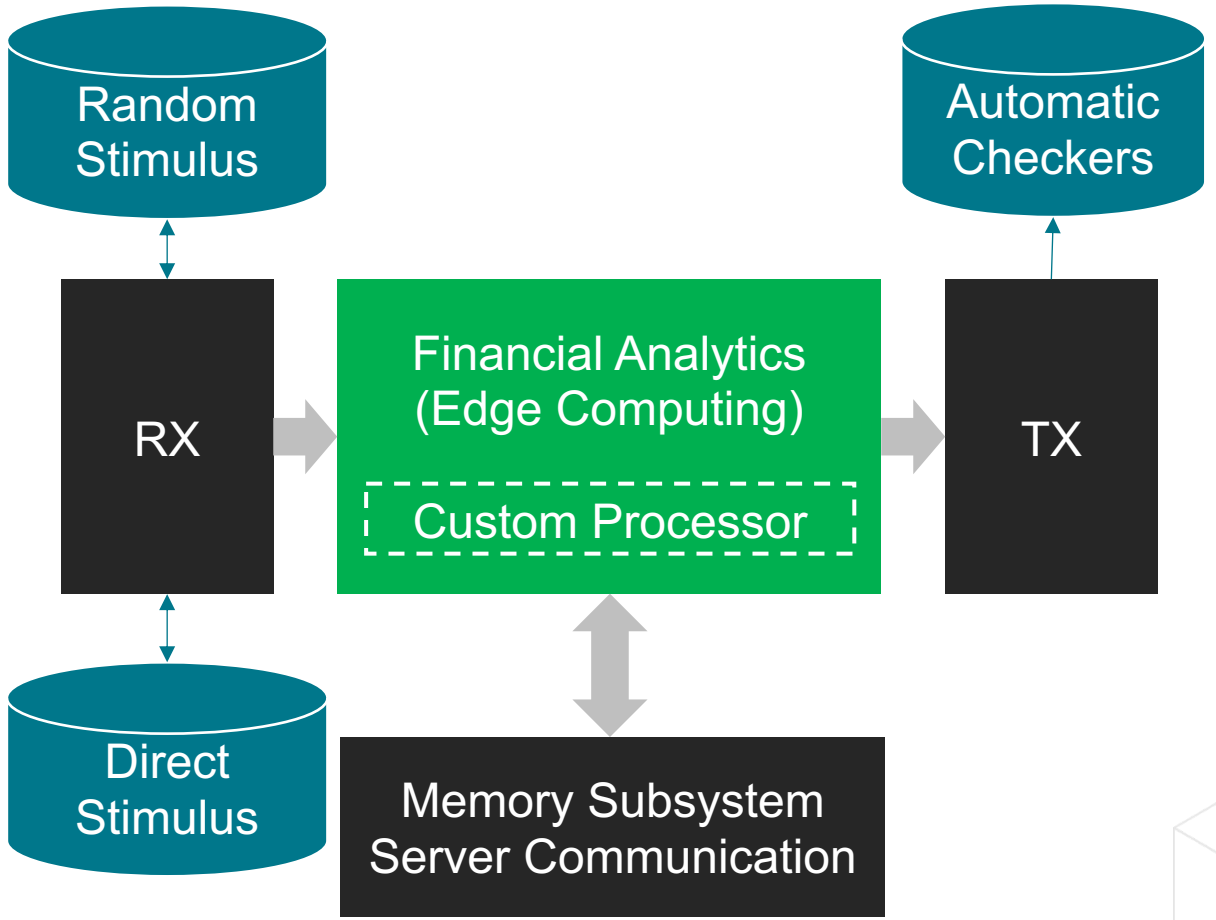
Design States



	Direct Test	UVM	Formal Apps	Formal Static	Emulation
Benefit	* Simple to write * Best for strongly sequential test (boot)	* Order of magnitude faster test generation * Cover more states with more confidence (functional coverage)	* Complete verification in specific tasks (code coverage, CDC, etc.) * Much lower compute and engineering cost vs simulation	* Complete verification for design features (bus contention) * Much lower compute and engineering cost vs simulation	* Verify bare-metal and embedded SW * Verify deep-cycle HW states * >1000x faster than simulation
Limitation	* Every state explicitly written into each test * Unknown level of completeness	* Computationally intense with some states hit on every test	* Design size (100k to 1M registers) * Pre-defined tasks built onto engine	* Formal modeling knowledge * Comprehensive assertion writing	* Two state logic * Synthesizable code required (behavioral connects via sim)
Engine	Logic Simulation	Logic Simulation	Formal Model Analysis	Formal Model Analysis	Hardware Emulation

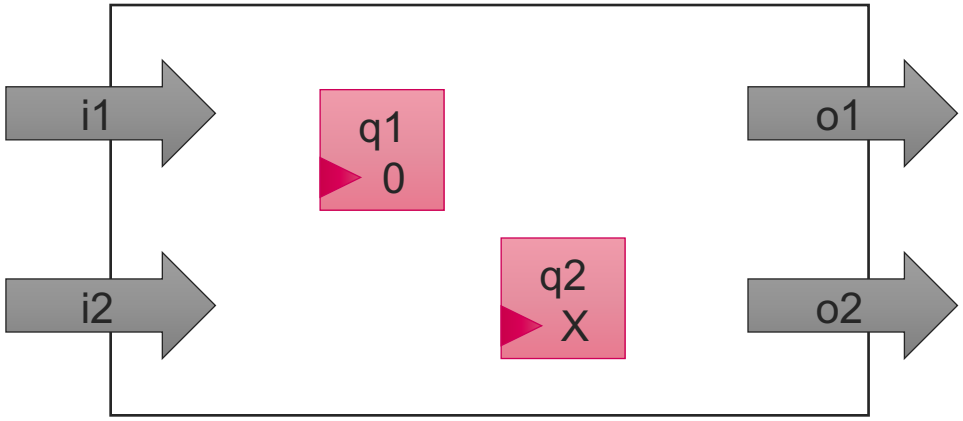
Utilize Direct and Randomized Simulation

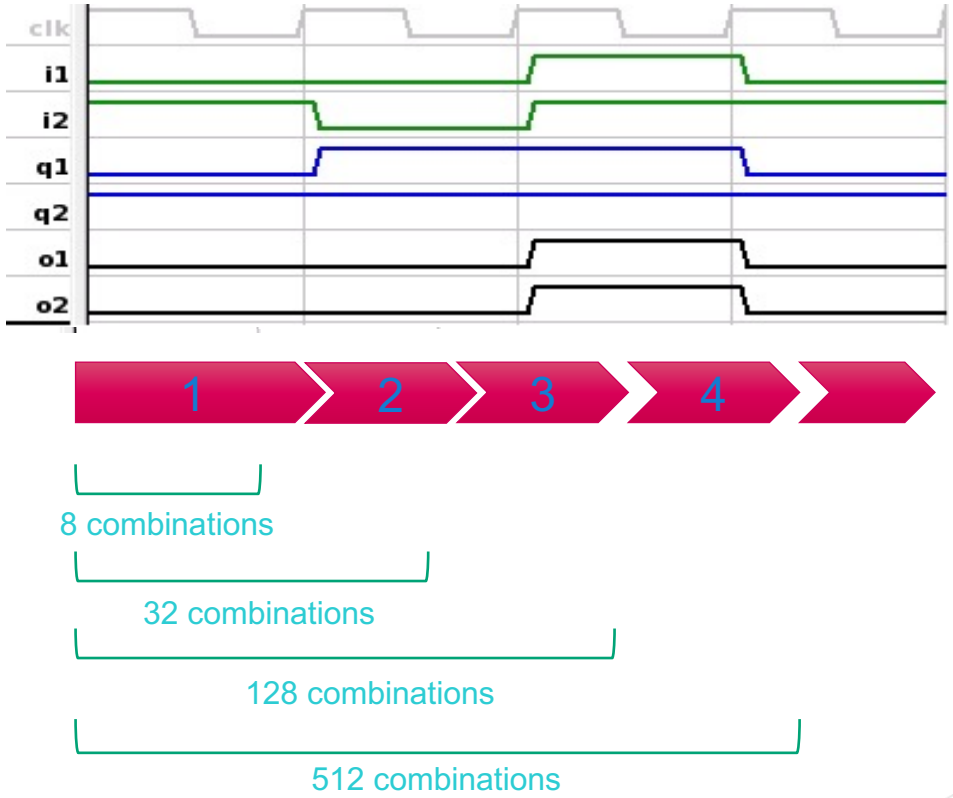
- Randomize packets and data loads
  - Models exchange traffic
- Direct stimulus
  - Tests specific corner cases
- Value: debug visibility and performance analysis of exchange-similar traffic
- Challenge: incomplete analysis of overall design state-space



Formal Analysis

- Formal tests all possible stimulus, one cycle at a time
  - All value combinations on inputs and undriven wires at every cycle
  - All value combinations on unitialized registers at first cycle

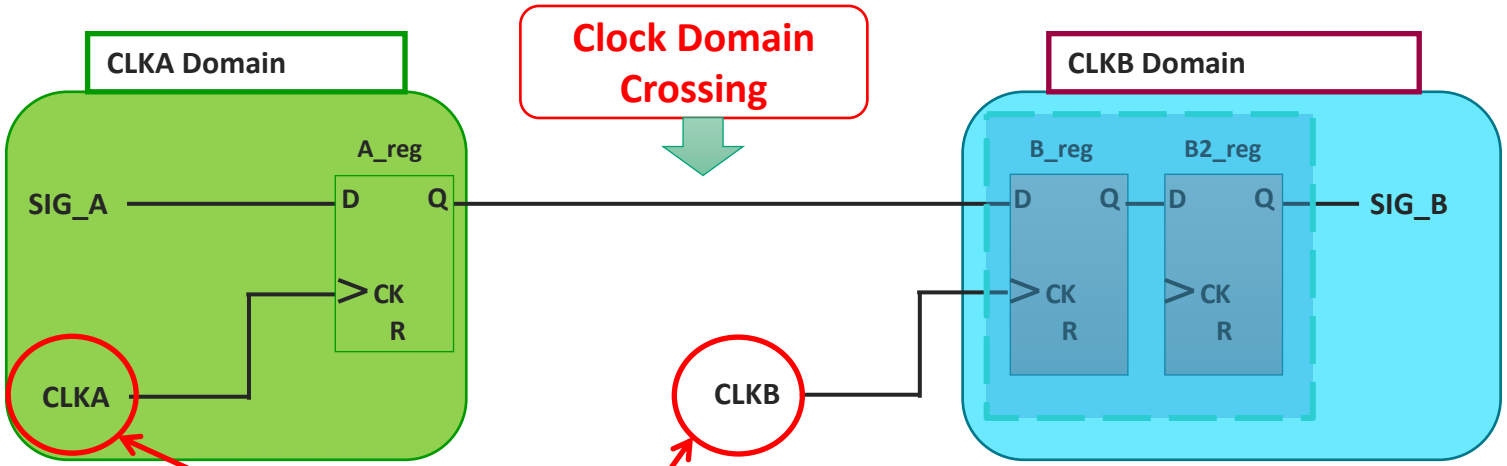


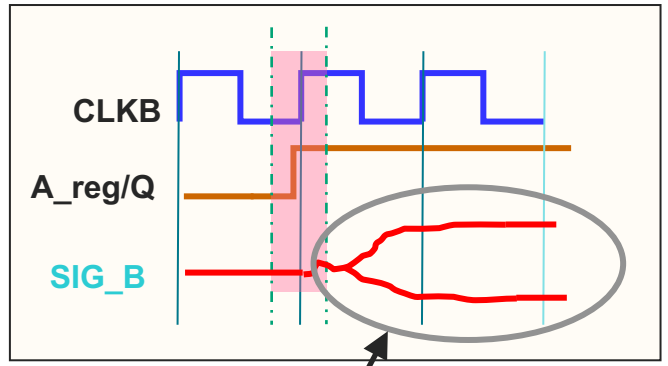


- Similar to simulating **`$random`** at every input/register for all possible random values!

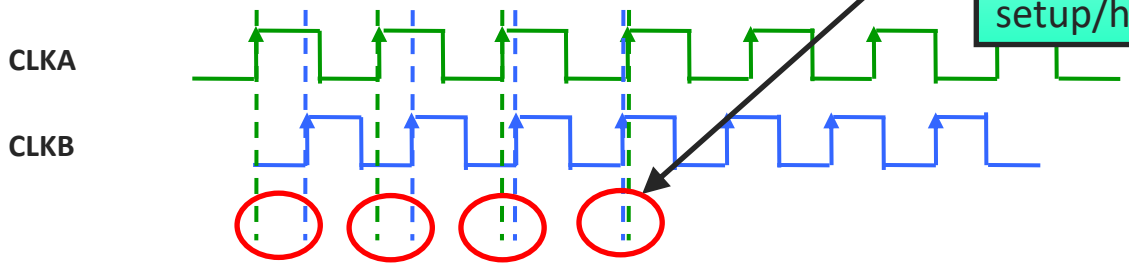
Clock Domain Crossing Problem

- Clock Domain Crossing (CDC) occurs when a signal crosses from one asynchronous clock domain to another





if CLKA & CLKB are asynchronous



Non-deterministic relationship between the clocks causes them to skew continuously resulting in setup/hold violations

Flop output goes to metastable state which eventually settles to a 1 or 0 after an unpredictable delay

- Similar problems affect Reset Domain Crossings (RDC) too

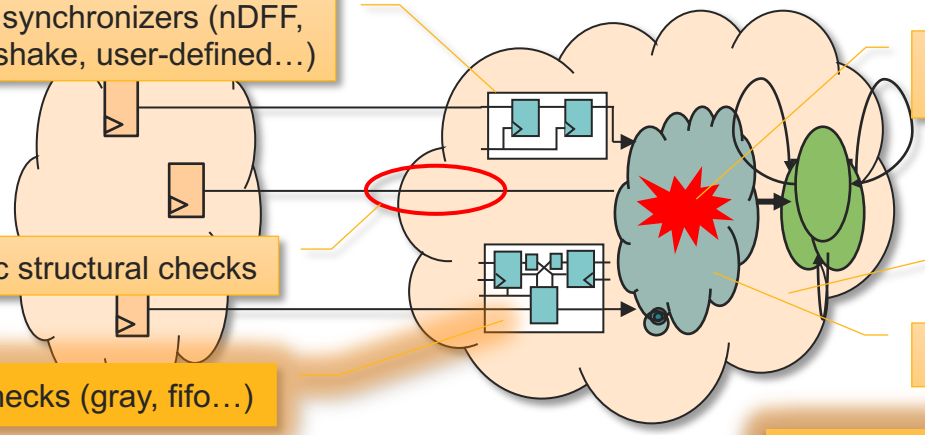
Formal Analysis of Potential CDC Issues

Comprehensive Checks

Wide range of synchronizers (nDFF, mux, fifo, handshake, user-defined…)

Automatic structural checks

Functional checks (gray, fifo…)



Convergence/ reconvergence

Hierarchical analysis

RDC checks

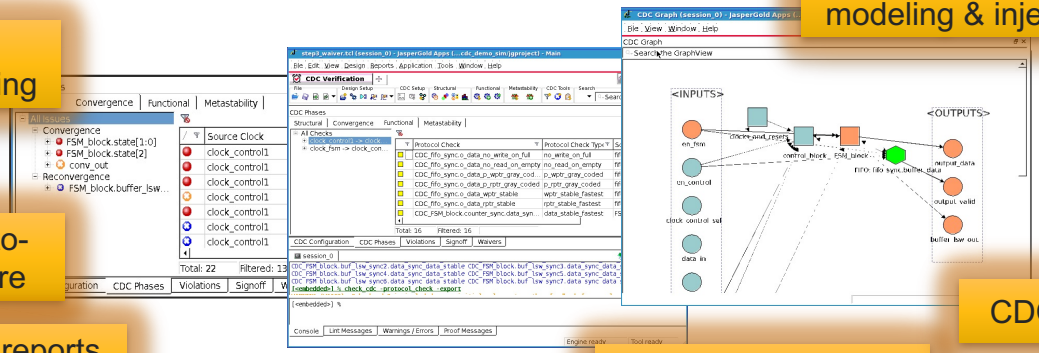
Enabled by true formal technology

Innovative Debug & Waiver Handling

Low-noise violation handling

Powerful auto-waiver feature

Flexible reports with filters



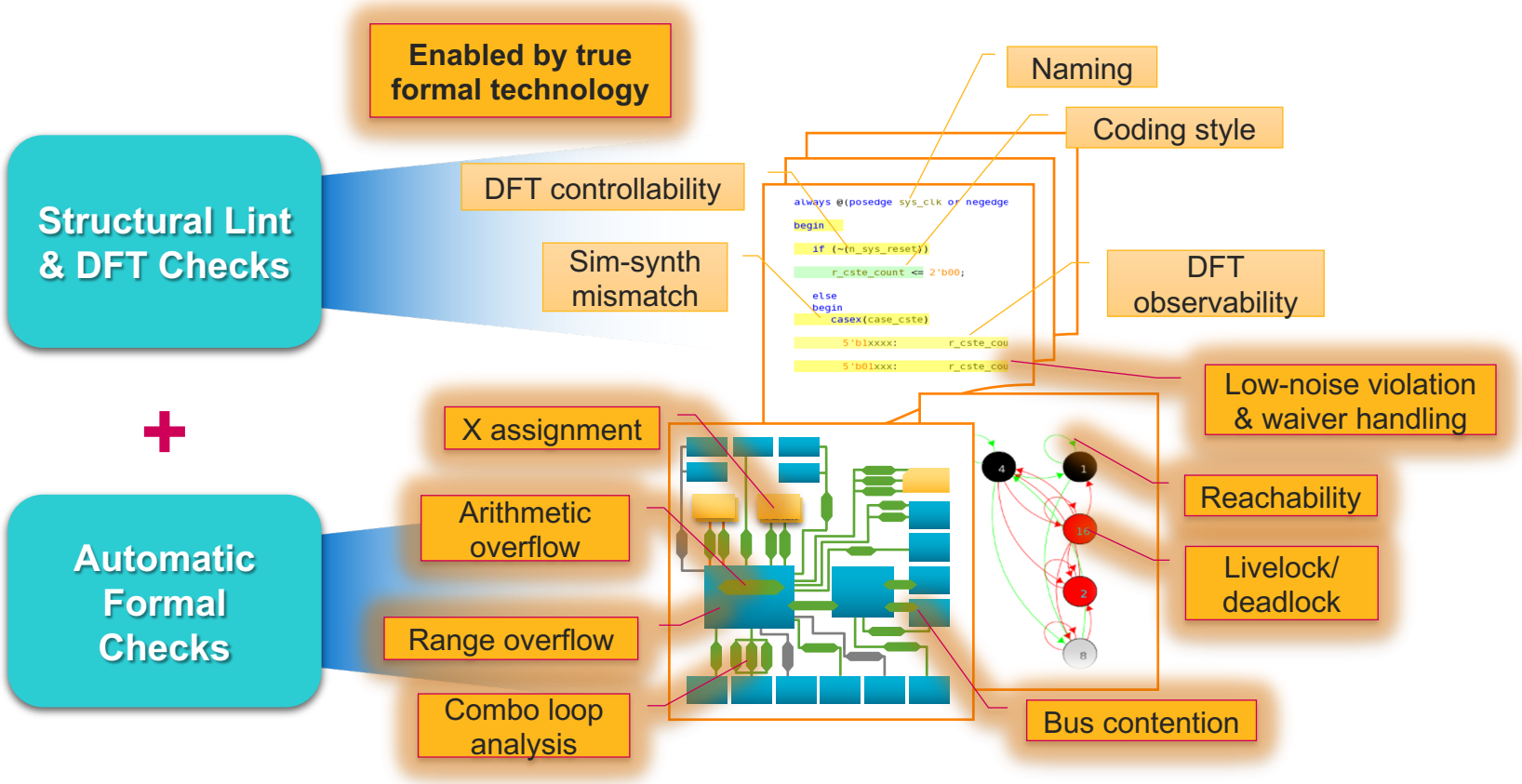
Metastability modeling & injection

Innovative debug with graph view

CDC coverage

Assertion generation & export to sim

Enforcing Design Practices with Linting



- Recent capabilities
  - Deadcode debug root-cause analysis
  - Improved waiver handling
  - Scalability: performance and memory optimization
  - FSM deadlock/livelock check performance improvements

Comprehensive functional checks, violation debug, and waiver handling based on best-in-class formal analysis

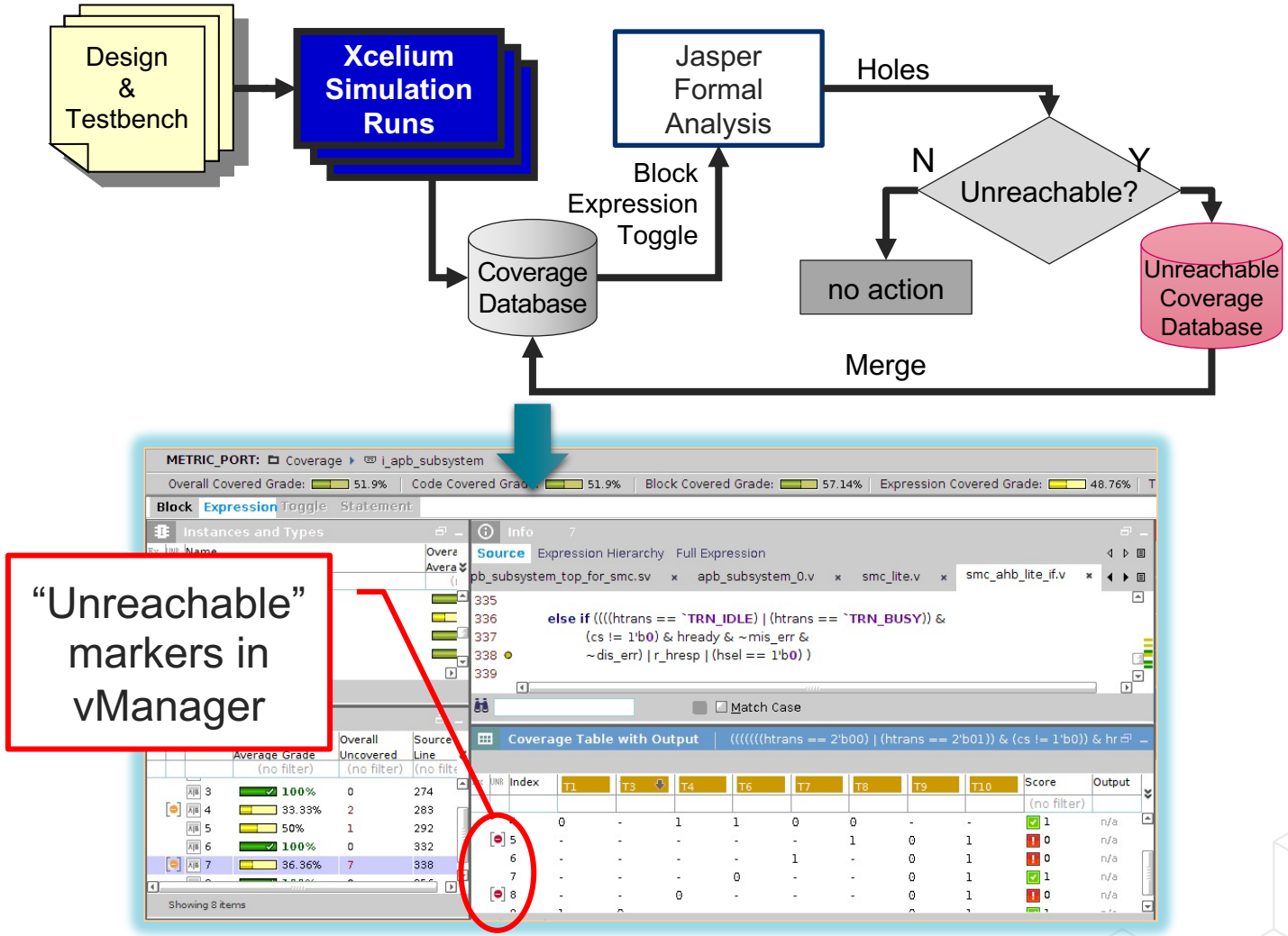
Formal Analysis for Unreachable Coverage: Dead Code Removal

Input: simulation coverage database and RTL

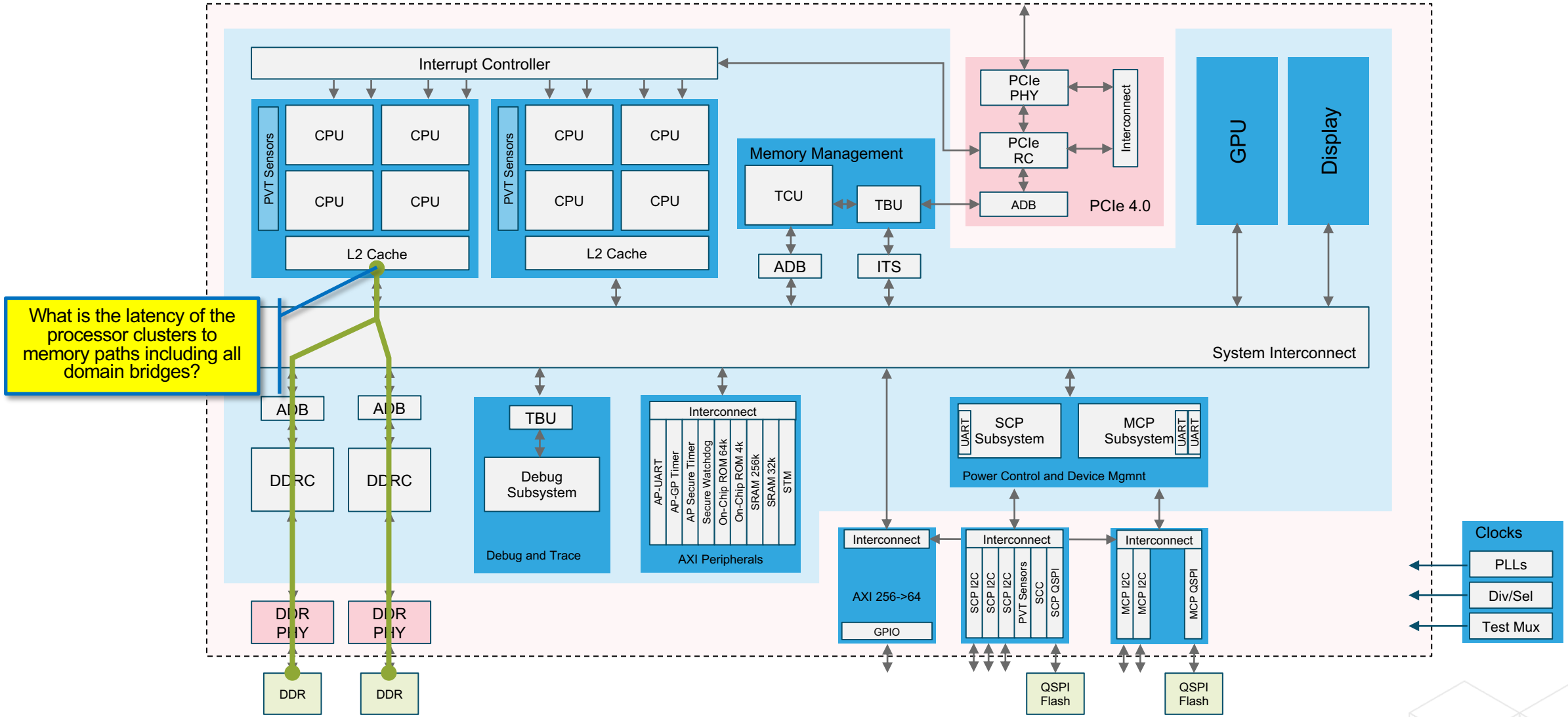
Output: Unreachable cover points database

Can run by sim users without formal expertise

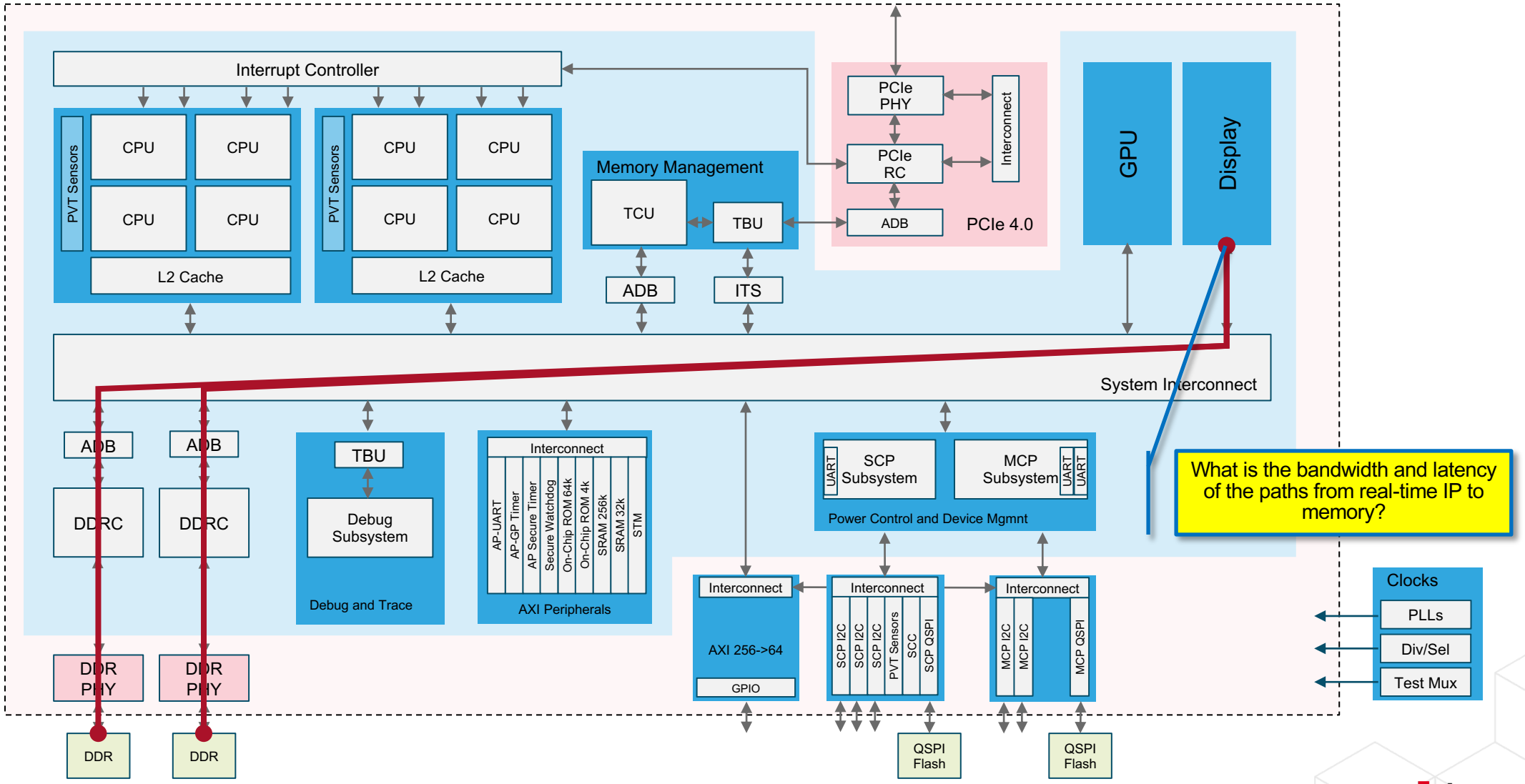
Integrated with vManager to clearly show unreachable coverage points



System Performance Challenges
Latency Critical



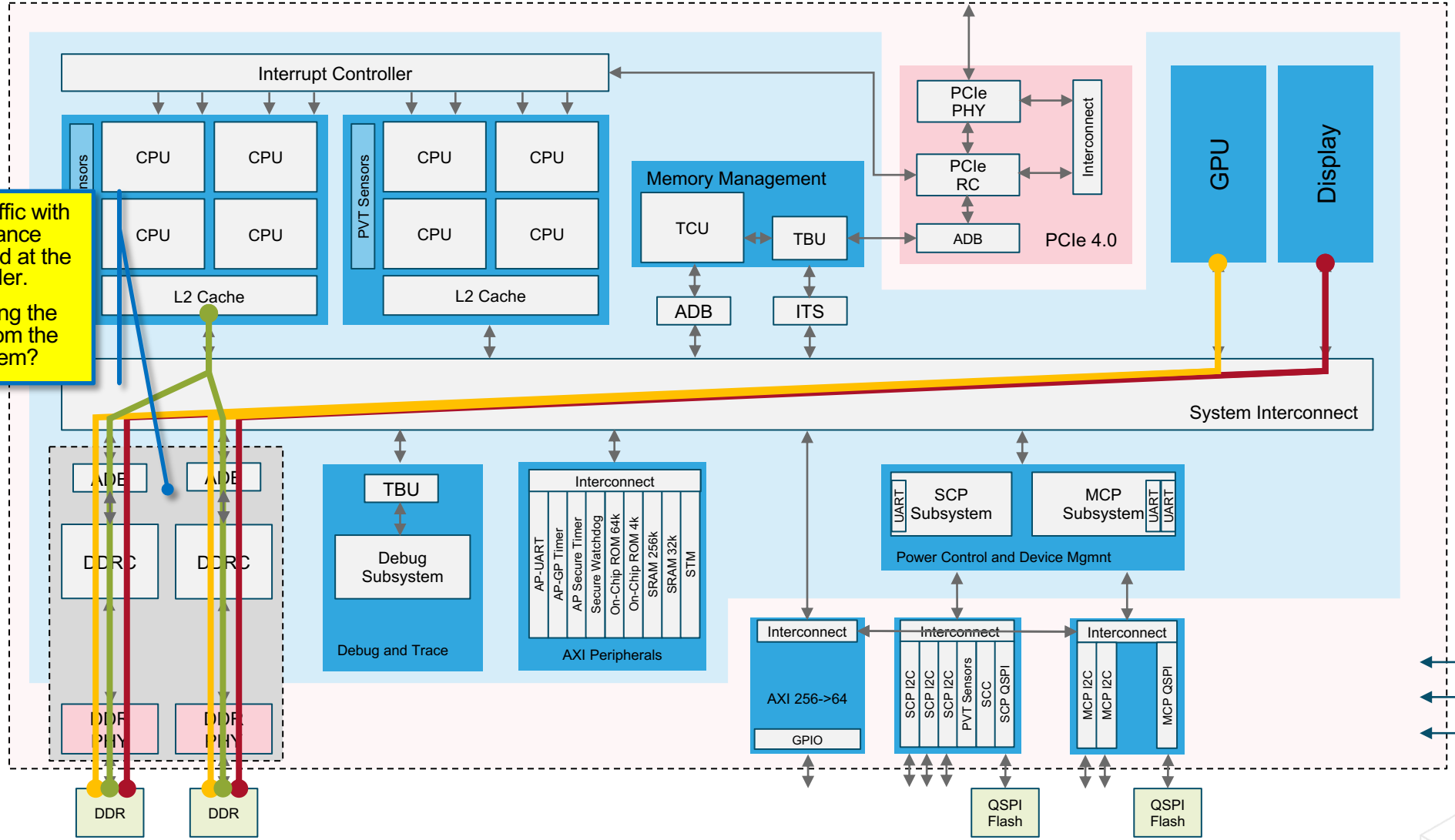
System Performance Challenges
Real Time Critical



Memory Subsystem Performance Challenges

Many streams of traffic with different performance demands are merged at the memory controller.

Are they all receiving the required service from the memory subsystem?



Metric Driven Verification Created to Quantify Verification

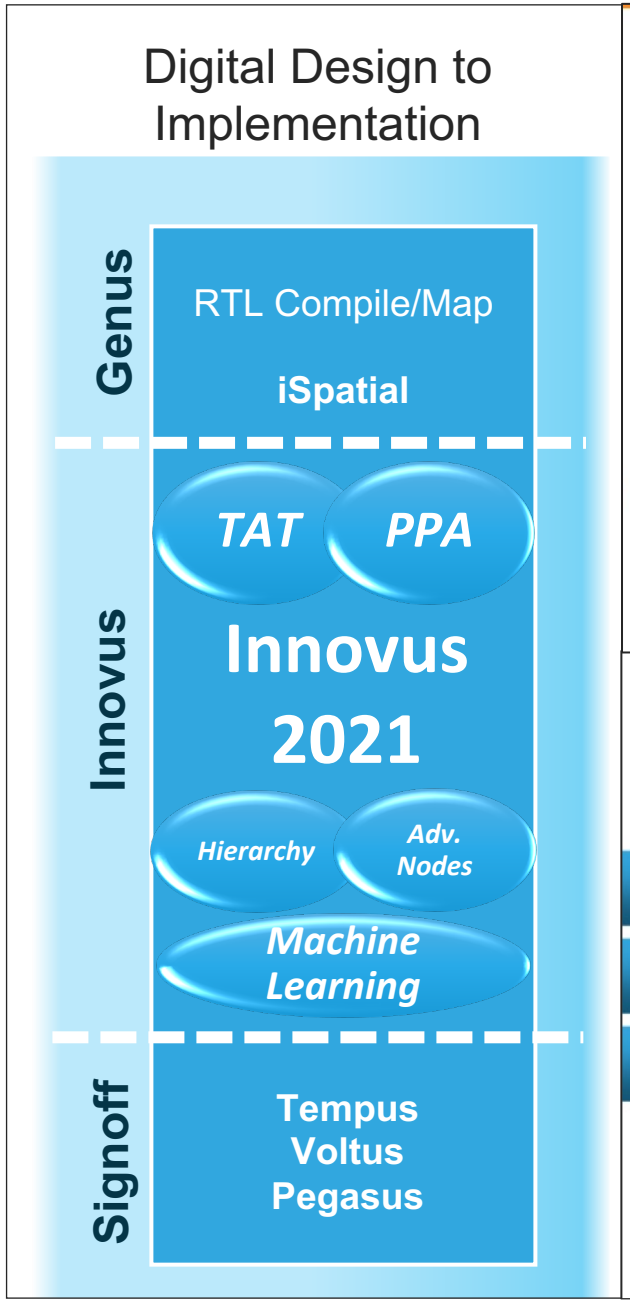
- Early 2000s: Synthesis and packet-based design rapidly increased design size
  - Verification scaled with randomization (UVM precursor), coverage, self-checking testbench
- Metric Driven Verification (MDV) assesses how comprehensive verification is
  - All tests pass
  - All code exercised with passing tests
  - All critical state combinations are exercised
  - Illegal/undefined/unspecified states are not entered
- MDV scales for complex systems
  - Data collection from multiple verification engines

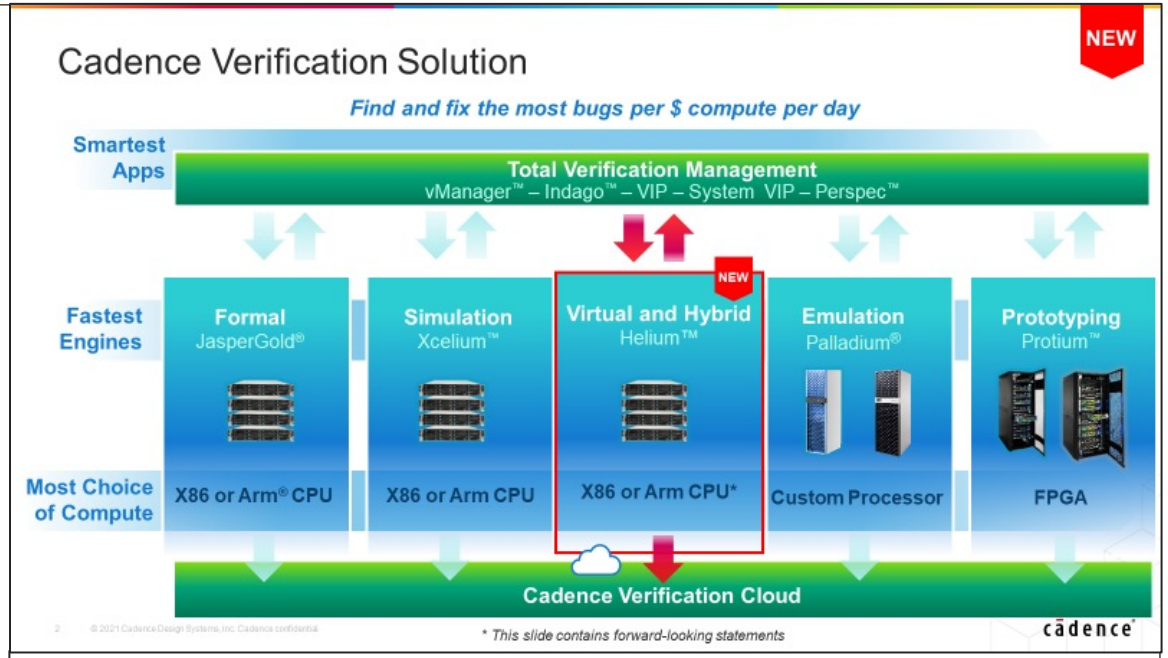
Next Steps to Being the Low-Latency Athlete

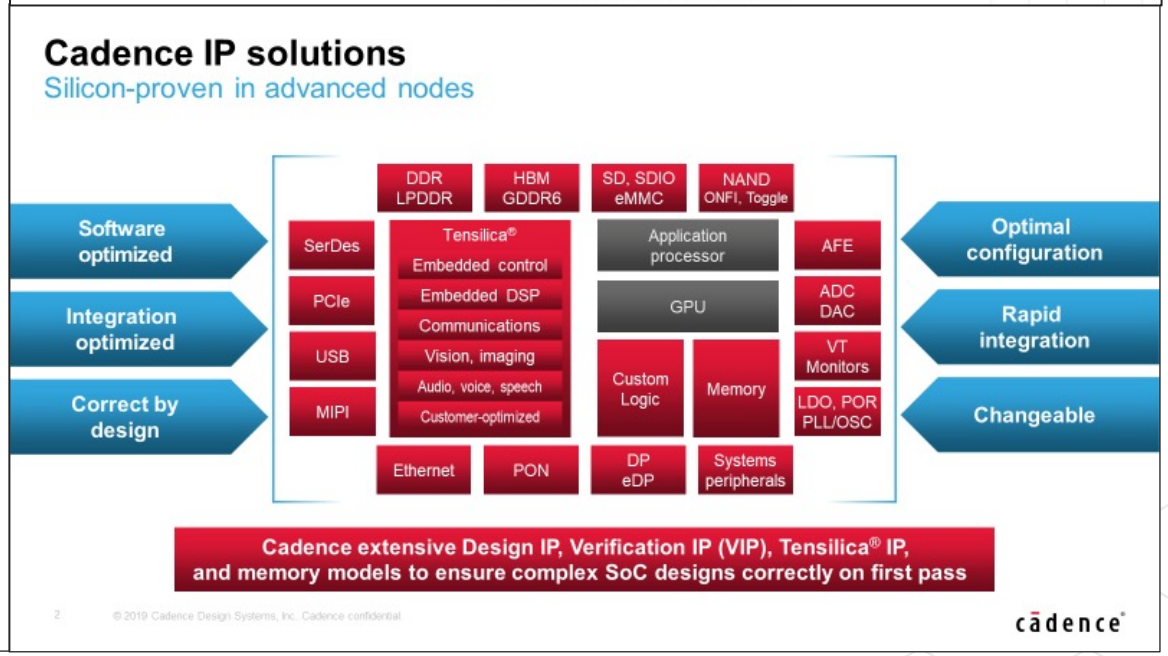
- Do your core work!
  - Profile even if you don’t see a performance bottleneck – everyone has a need for speed
  - Examine low-latency design approaches and how you can factor those into existing designs
- Maintain your performance health!
  - Increase use of simulation and formal methods to measure and correct any bottlenecks
- Break through your plateau!
  - Consider new low-latency design approaches when you implement a new architecture

Cadence is Your Design Partner

- Verification solution: apply objective analysis to improve FPGA and ASIC
  - Link to Tech Brief
- ASIC solution: broadly adopted in high-speed comms and mission-critical applications
- Tensilica IP: proven processor technology used in autonomous drive and other high-reliability apps
- High-perf IP: proven in leading comms systems
- Services: expert RTL to GDS design services







cadence