Reinventing SoC Verification – It Is about Time

Simon Chang

cadence®

Tutorial Agenda

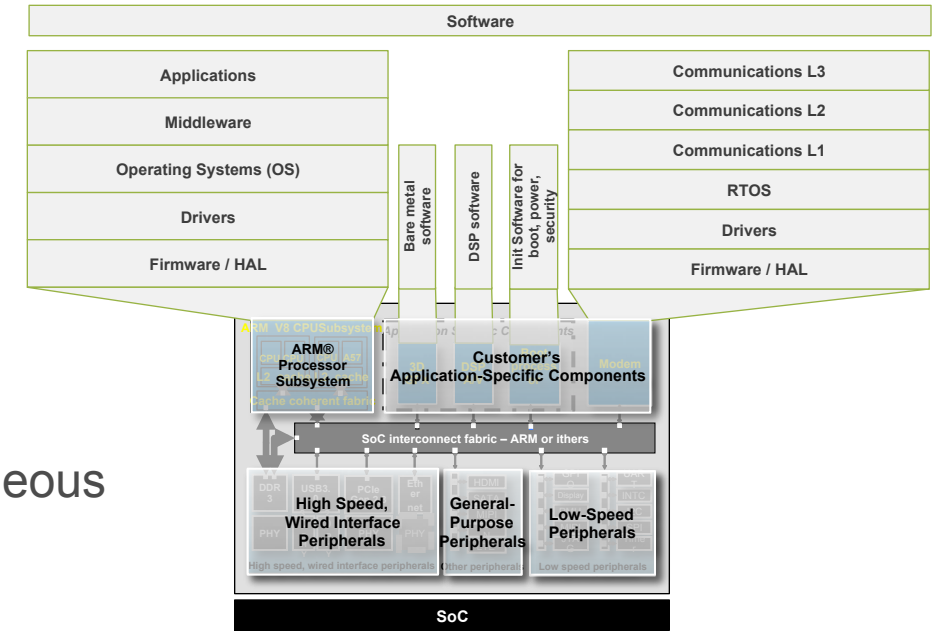
- Industry Status and Need for Change
- Integration Verification with Formal Technology
- Kicking Verification up a Notch with Multi-Core Simulation
- Leveraging Hardware: Acceleration, Emulation, and FPGA Prototyping
- Automating Test Creation with Portable Stimulus
- Bringing It All Together: Planning, Management, Coverage and Debug

Tutorial Agenda

- **Industry Status and Need for Change**
- Integration Verification with Formal Technology
- Kicking Verification up a Notch with Multi-Core Simulation
- Leveraging Hardware: Acceleration, Emulation, and FPGA Prototyping
- Automating Test Creation with Portable Stimulus
- Bringing It All Together: Planning, Management, Coverage and Debug

Challenges in HW/SW Systems

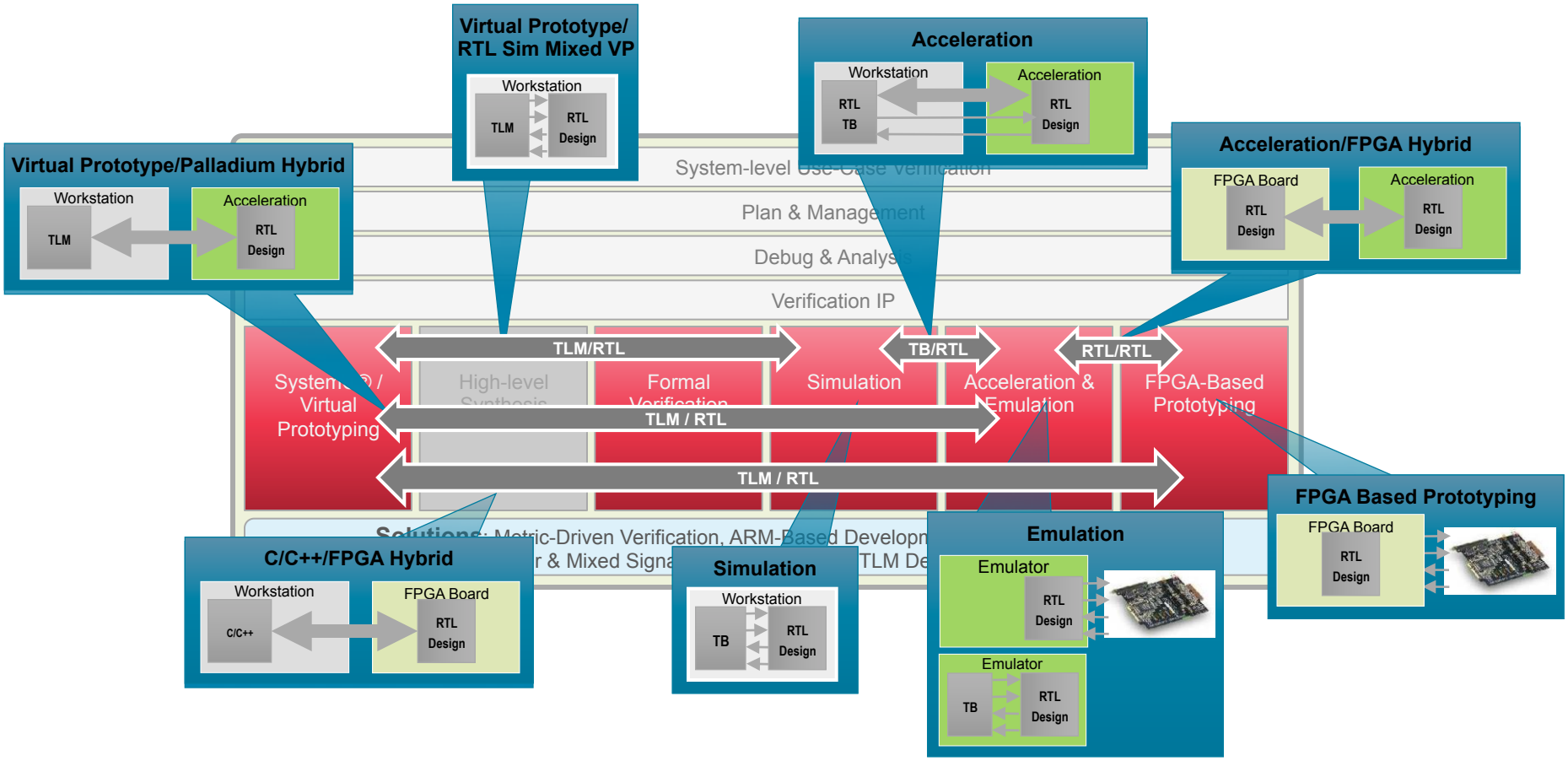
- Many IPs
  - Standard I/O
    - WiFi, USB, PCI Express® (PCIe®), etc.
  - System infrastructure
    - Interconnect, interrupts, UART, timers…
  - Differentiators
    - Custom accelerators, modem…
- Many cores
  - Both symmetric and asymmetric
  - Both homogeneous and heterogeneous
- Lots of software
  - Part of core functionality
    - Communication stack, DSP software, GPU microcode…
  - User application software infrastructure
    - Android, Linux…



cadence®

Common Customer Use Models Leveraging Integrated Suite

Enables optimized verification and SW development flows



Tutorial Agenda

- Industry Status and Need for Change
- **Integration Verification with Formal Technology**
- Kicking Verification up a Notch with Multi-Core Simulation
- Leveraging Hardware: Acceleration, Emulation, and FPGA Prototyping
- Automating Test Creation with Portable Stimulus
- Bringing It All Together: Planning, Management, Coverage and Debug

Why SoC Integration Verification

- Lint/super-lint are required before designers block go into verification
- Formal-based integration “Linting” before integrated subsystem/system go into system-level verification
- Similar to RTL linting, many issues can be cleaned up a lot quicker to enable system-level functional verification

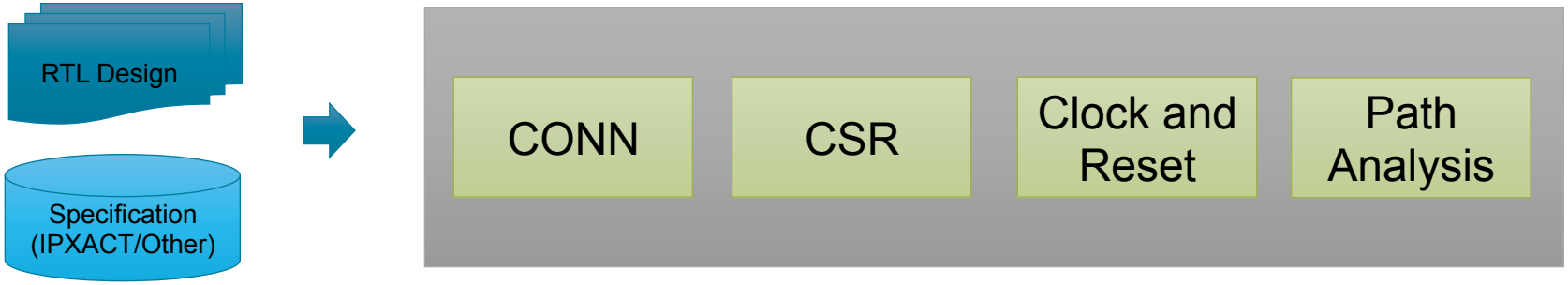
What Goes In / What Comes Out

- System specification (IP-XACT or other format) defining integration information
  - Direct connections (conditional or unconditional)
  - Indirect connection – how does one block talk to another without a direct connection
  - Address mapping for the different components
  - Control and status registers
  - Low-power descriptions (UPF, CPF, IP-XACT extension)
  - Clock and reset information
- Verify that the integrated system meets the specifications

SoC Integration Verification – What It Is and Is Not

- It is NOT SoC functional or system verification
- It verifies that the System level integration specification matches the RTL
- For example: Verify that a master is reaching a slave via expected path, we check:
  - The physical connection along the path
  - The address mapping, mostly to ensure that the transaction passes through the interconnect
  - The interconnect is configured correctly
  - There is no other paths that the expected path
  - Does not verify the full data-integrity

The Flow



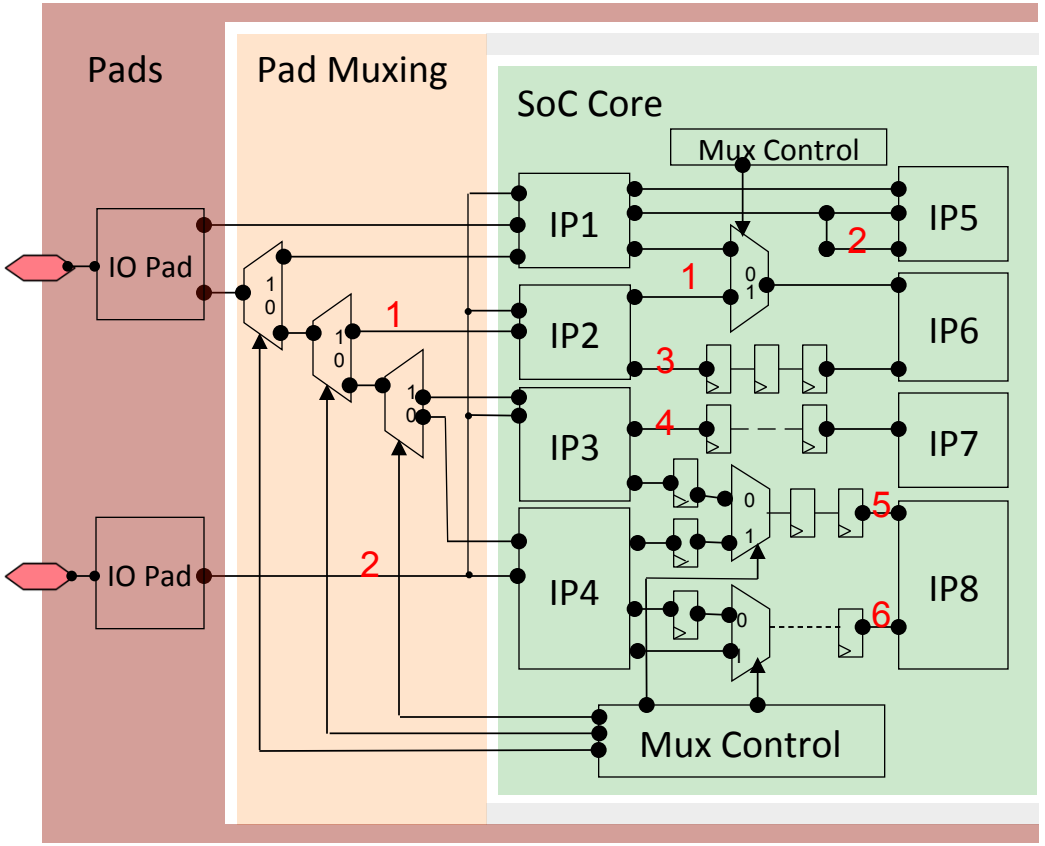
- Clock and Reset Verification
- Control & Status Register Verification
- Master/Slave Path Verification
- Connectivity Verification

The Problem: A Tangled Web

Connection Types

Combinational
1. Direct
2. Conditional

Pipelined
3. Fixed latency
4. Variable latency
5. Fixed latency and multiplexing logic
6. Variable latency and multiplexing logic



The Key: The 3 C's

- Proving all connections in the chip are correct
  - In a typical chip there may be 10,000+ connections to verify
- Traditional simulation methods lack efficient way to exercise the connections
  - Significant manual effort to apply stimulus → **Controllability**
- Very difficult to assess coverage in a simulation environment → **Coverage**
  - Have you really exercised all combinations?
- Finally, it's a big challenge to identify whether all connections are tested by the testbench → **Completeness**
  - Do you have checks for all connections?

The Solution: Formal Analysis

Solving the 3 challenges:

1. **Controllability**
2. **Coverage**
3. **Completeness**

Black-Boxing
By removing unnecessary logic, we can give the tool direct access to IP interfaces to apply stimulus

Formal Verification
The exhaustive nature of formal analysis ensures a comprehensive analysis for the connections being tested

Reverse Connectivity
Automatic RTL analysis identifies missing checks for existing connections

Problem Statement: Register Verification

- Critical validation goal of any programmable block is checking the design matches the register specification
  - Straightforward to verify under ideal conditions (i.e. simple read-write registers)

- Time consuming and difficult for more complex cases
  - Register interactions
  - Secure access
  - Dependency on status update
  - Simultaneous actions

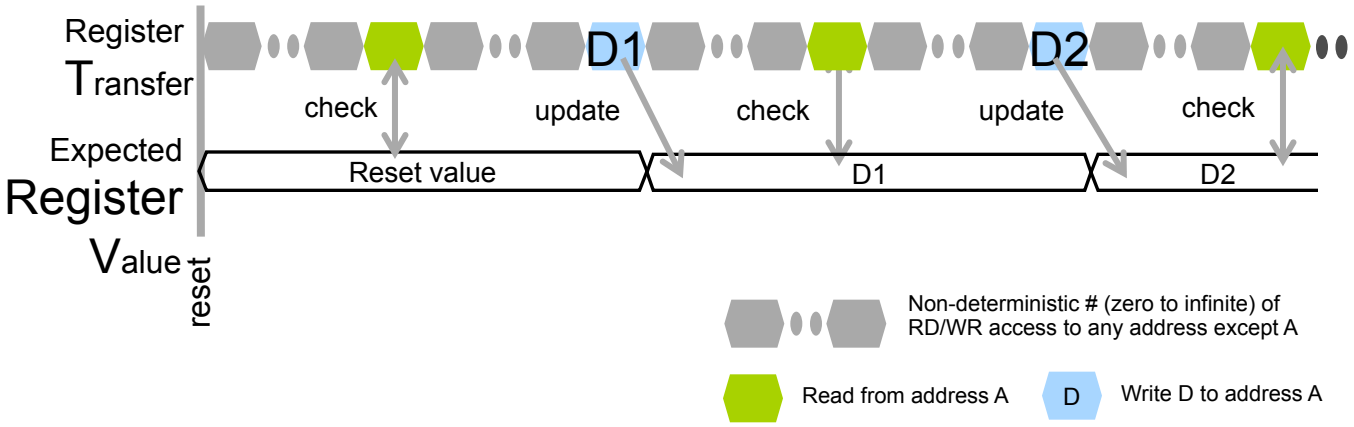
Problem Statement: Register Verification (cont.)

- Proliferation of registers
  - Even small IP blocks can have 100s
  - Unit level control & status registers can number in the thousands
  - Automation is key to effectively dealing with large numbers
- Prove the data integrity of register fields
  - Reset values – data read gets reset values until it is written to
  - Write values – data read gets values from most recent data write
  - Access types / attributes

Exhaustive Register Verification

CSR

- Formal proofs are exhaustive
  - Checks for all possible sequences of RD/WRs in any order
  - Checks for all register addresses, aliases, and interferences
- Conceptually, the following non-deterministic trace is considered by formal for proving address A:

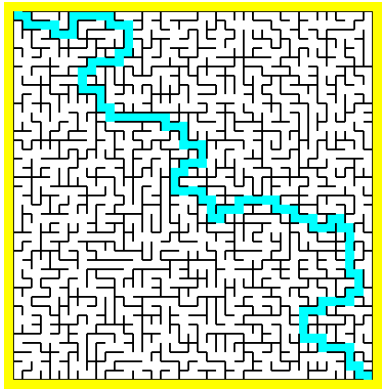


cadence

SoC Clock and Reset Verification

- Specification
  - Define clocks and reset pins at the top-level
  - Which flops get which type of reset
  - Obtained from IP-XACT or CSV specification
  - Identify generated clocks, gated clocks
- SoC Clock and Reset
  - Identify clock/reset drivers and connections between top-level and block-level
  - Verify the specification
  - Help create reset sequence
  - Extract clock and reset connections

Path Analysis

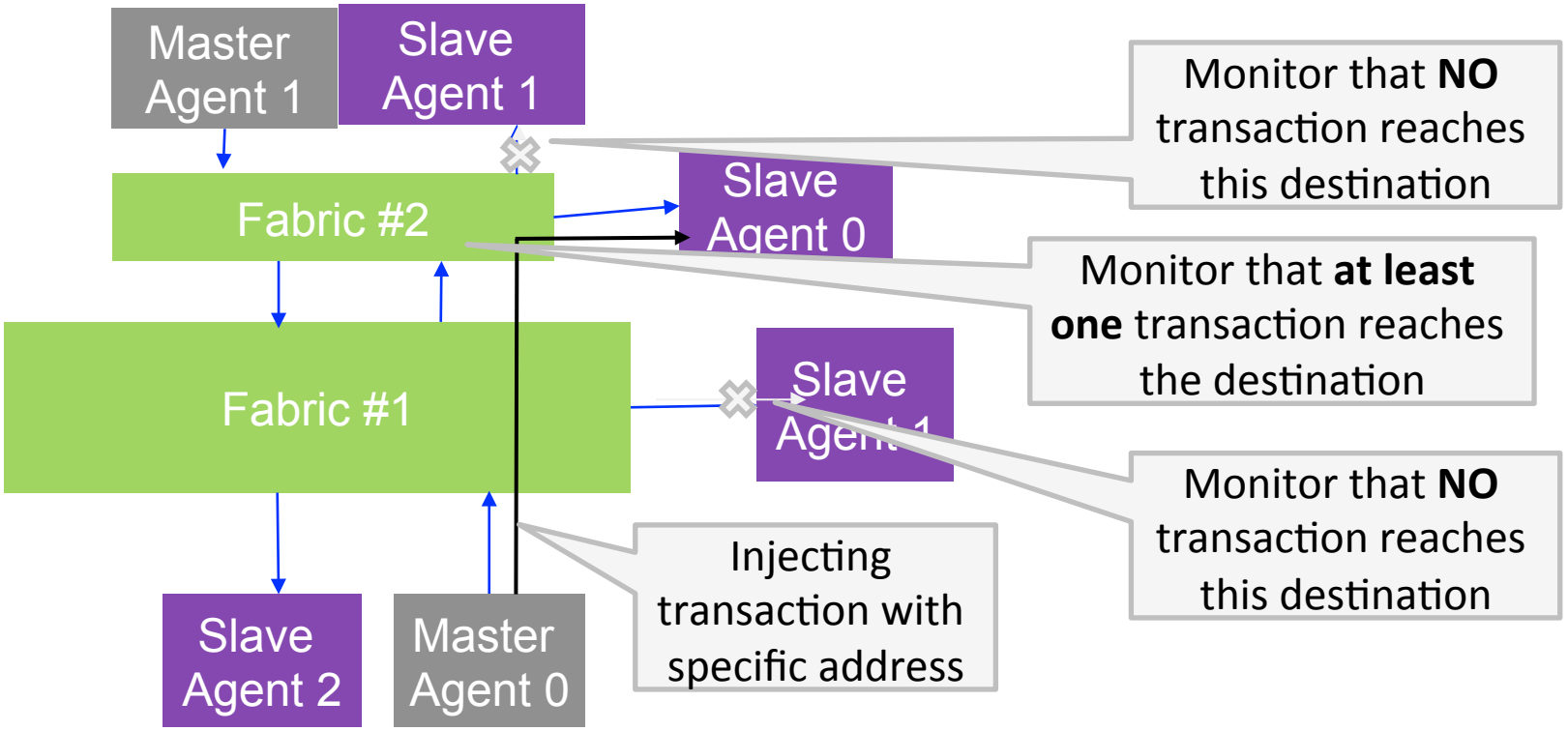


- User provides
  - RTL Master and Slave interface
    - ABVIP (standard or custom) instantiations for each port
  - Spec (IP-XACT or CSV/xls)
    - Address ranges for the master/slave agents connected to the fabric
    - Other information like: Interface clock, address bus width, instance of ABVIP for corresponding port, etc.
- Tool provides automation
  - Transactions show up at correct destination port (group of covers)
  - Ensure transactions do not show up at unintended destinations (assertion per unintended destination)
  - Create tasks that exercise master/slave permutations containing the properties above

Path Analysis Examples

- Considering the following

Repeat for all paths and address ranges

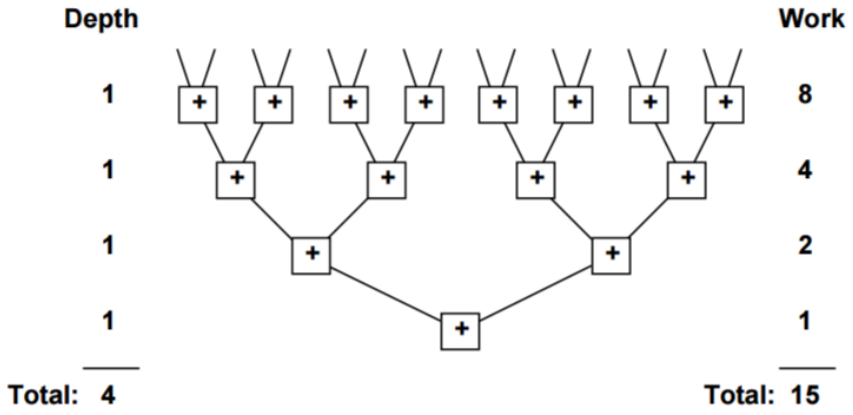


Tutorial Agenda

- Industry Status and Need for Change
- Integration Verification with Formal Technology
- **Kicking Verification up a Notch with Multi-Core Simulation**
- Leveraging Hardware: Acceleration, Emulation, and FPGA Prototyping
- Automating Test Creation with Portable Stimulus
- Bringing It All Together: Planning, Management, Coverage and Debug

Does Parallelism Really Require New Algorithms?

- In some cases, no
  - For example, a for-loop can sometimes be broken up to smaller tasks
  - “Embarrassingly parallel algorithms”
- In other cases, yes
  - Example: sum of a vector of elements
    - Single CPU algorithm: go over the vector elements one by one and accumulate them into a single memory location
    - Parallel CPUs algorithm: do a “binary tree” sum



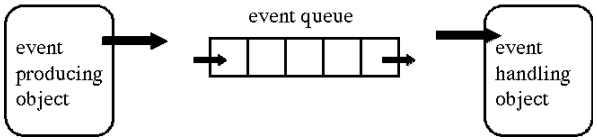
Multi-Core Isn't the Only Server Change: Now Cache Is King

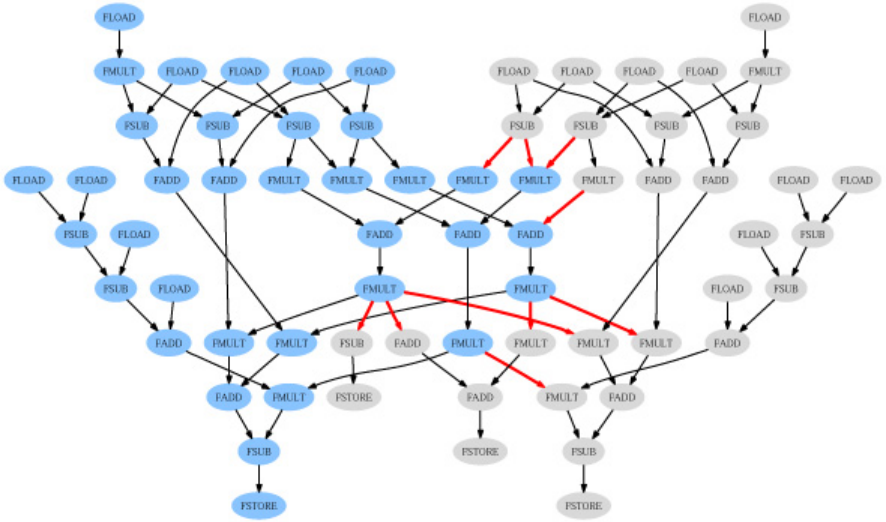
- Multi-core is not the only thing that's changed in the last 20 years
- Cache memory hierarchy has become increasingly important for performance optimizations
- Example:
  - In the 90's: linked-list was considered perfectly fine
  - Today: keeping your elements in a contiguous vector is much better!
- Careful consideration of modern server architecture is key

Why a Paradigm Shift?

- Optimizing any 20-year-old architecture for new servers is challenging
  - Change expense is high and performance yield is often low – which is the software headache generally observed
  - From engineering perspective it is also known that all architectures have a life cycle – the challenge is finding the right circumstance to trigger that new architecture
- A new simulation engine, re-architected from scratch, is a solution
- Modernization isn't enough – a different algorithm is required
  - Analyze the dependency graph at a global scale
  - Existing algorithms rely too much on global, cache-inefficient data structures

Parallel Simulation Requires a Paradigm Shift





cadence

Introducing Xcelium Parallel Simulator

High-performance third-generation parallel simulator

REVOLUTIONARY

rocketick

+

PROVEN

Incisive®
Enterprise Simulator

+

OPTIMIZED

- **2X** average single-core **speed-up**
- **Direct kernel** engine integration
- **New randomization** engine

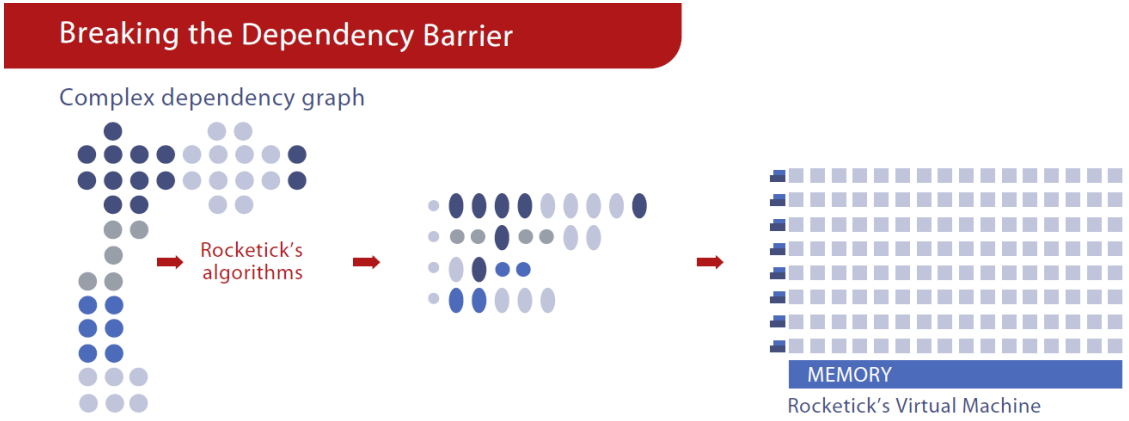
Xcelium™
Parallel Simulator

3X+ RTL **5X+** Gate **10X+** DFT

Xcelium™ Contains RocketSim® Multi-Core Simulation Technology

New core engine for third generation of simulation

- Unique compile/elaboration process analyzes each design's dependency graph
- Patented and patent-pending technology proven scalable beyond a billion gates
- Multi-core speed-up over single-core performance



- Automatically maps dependency graph to cores to maximize speed
- Users can set number of cores on command line without recompile

Xcelium Multi-Core Enables More Efficient SoC Verification

- Single-core simulation slows with more events
- To compensate, verification teams narrowed tests
- But SoC functions operate concurrently
- Multi-core simulation is more effective with higher event density

Simulation Function	Event Density	SoC HW Run Time	Single-Core	Multi-Core	Multi-Core Speedup
Boot Sequence	1.0	28 ms	16.8 hr	5.7 hr	2.9X
Test Scenarios	4.3	15 ms	37.3 hr	6.8 hr	5.5X
Overall	2.1	43 ms	54.1 hr	12.5 hr	4.3X

- Multi-core simulation enables test methodology better suited for SoC
- Perspec™ System Verifier can be used to create concurrent scenarios
- Creates test alignment between Xcelium™ multi-core and Palladium® Z1 acceleration

Concurrent Test Scenarios	Single-Core	Multi-Core
1	0.6 hr	0.2 hr
2	1.0 hr	0.3 hr
3	1.4 hr	0.4 hr
4	1.7 hr	0.5 hr
5	2.1 hr	0.6 hr

cadence®

Xcelium Summary

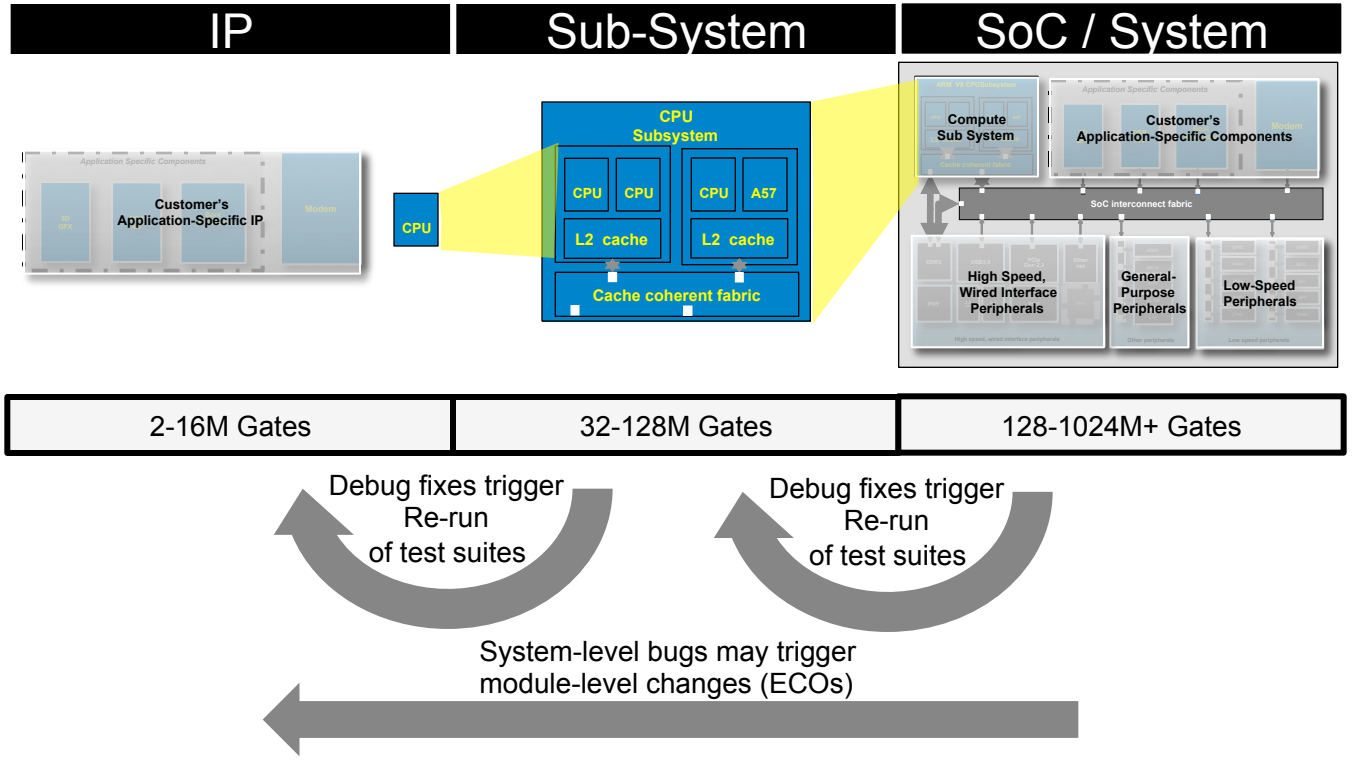
Xcelium™ Parallel Simulator, the industry's first production-ready third-generation simulator

- **Multi-core engine** architected for **fast SoC** simulation
  - **Patented** solution analyzes design and selects configuration for speed
  - Provides on-average simulation speed-up of **3X RTL**, **5X gate-level**, **10X DFT**
- **Single-core** engine refactored for **fast IP** simulation
  - Proven engine runs comprehensive set of simulation use cases
  - Provides on average **2X speed-up** over Incisive® Enterprise Simulator
- **Productivity** features enable efficient verification
  - Innovative solutions on top of fast engines reduce overall project time
  - Provides **enhanced smart exclusion flow and parallel multi-core**

Tutorial Agenda

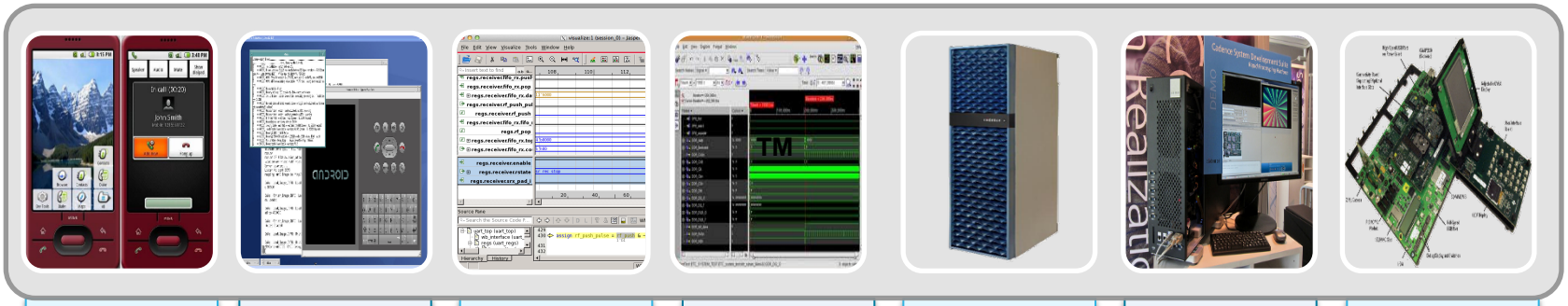
- Industry Status and Need for Change
- Integration Verification with Formal Technology
- Kicking Verification up a Notch with Multi-Core Simulation
- **Leveraging Hardware: Acceleration, Emulation, and FPGA Prototyping**
- Automating Test Creation with Portable Stimulus
- Bringing It All Together: Planning, Management, Coverage and Debug

Verification of Different Payload Sizes



There Is No “One Size Fits All”

Verification and software platforms need to interoperate



SDK OS Simulation

- Highest speed
- Earliest in the flow
- Ignore hardware

Virtual Platform

- Almost at speed
- Less accurate (or slower)
- Before RTL
- Great to debug (but less detail)
- Easy replication

Formal Verification

- Exhaustive
- Finds bugs early
- Great for IP
- Few apps scale to SoC
- No SW execution

HDL Simulation

- KHz range
- Accurate
- Excellent HW debug
- Broadly available
- Mixed-abstractions
- Limited SW execution

Acceleration Emulation

- MHz Range
- RTL accurate
- After RTL is available
- Good to debug with full detail
- Expensive to replicate

FPGA Prototype

- 10’s of MHz
- RTL accurate
- After stable RTL is available
- OK to debug
- More expensive than software to replicate

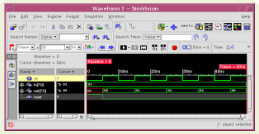
Prototyping Board

- Real time speed
- Fully accurate
- Post Silicon
- Difficult to debug
- Sometimes hard to replicate

Use Models for Hardware Based Verification

- Performance
- Cost
- Native interfaces

Early software development



- Fast compile
- Flexible stimuli
- Debug
- Use models

HW verification

- Capacity
- SpeedBridge
- Transaction I/F

HW regressions

- Performance
- Debug
- Transaction I/F

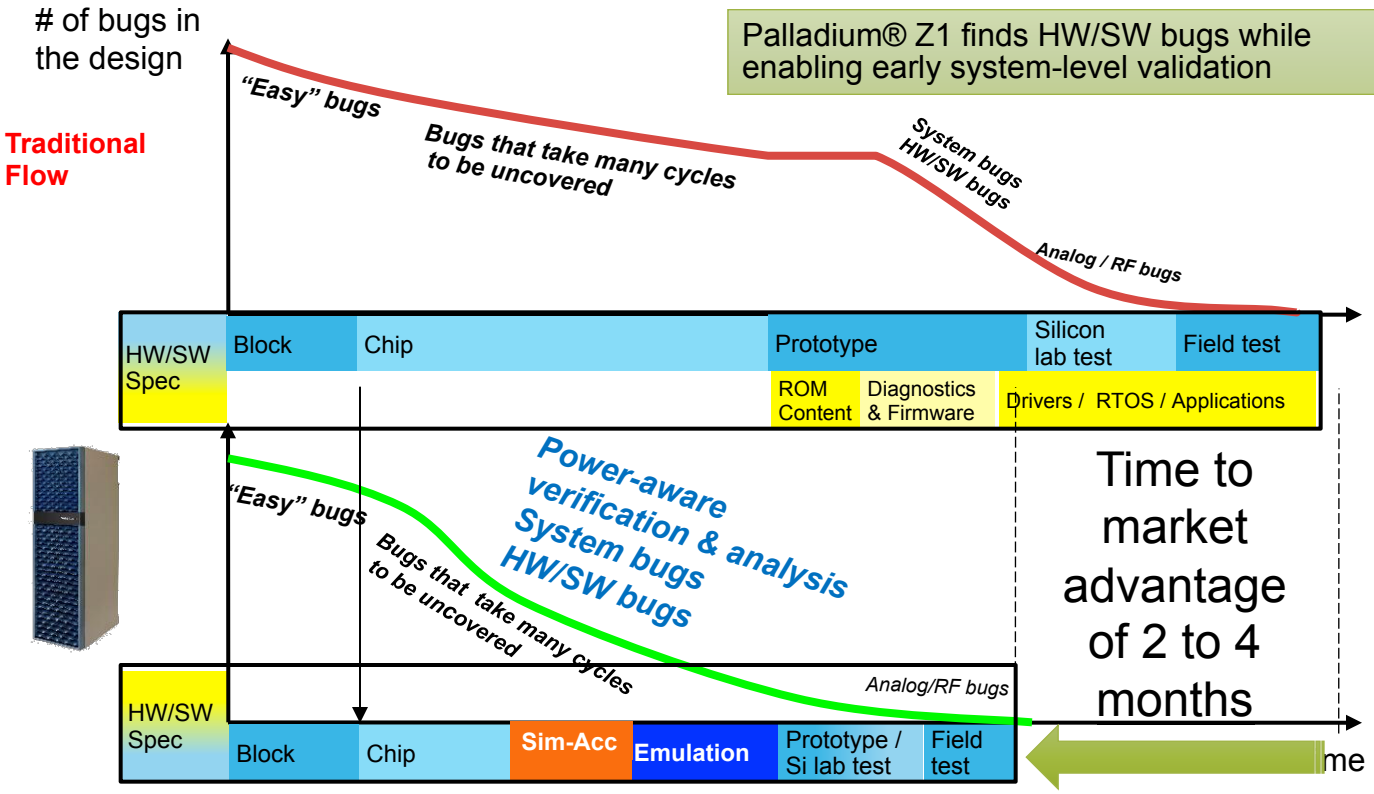
HW/SW integration

- Capacity
- Performance
- Any interface

System validation

Palladium Z1: Core Value Proposition

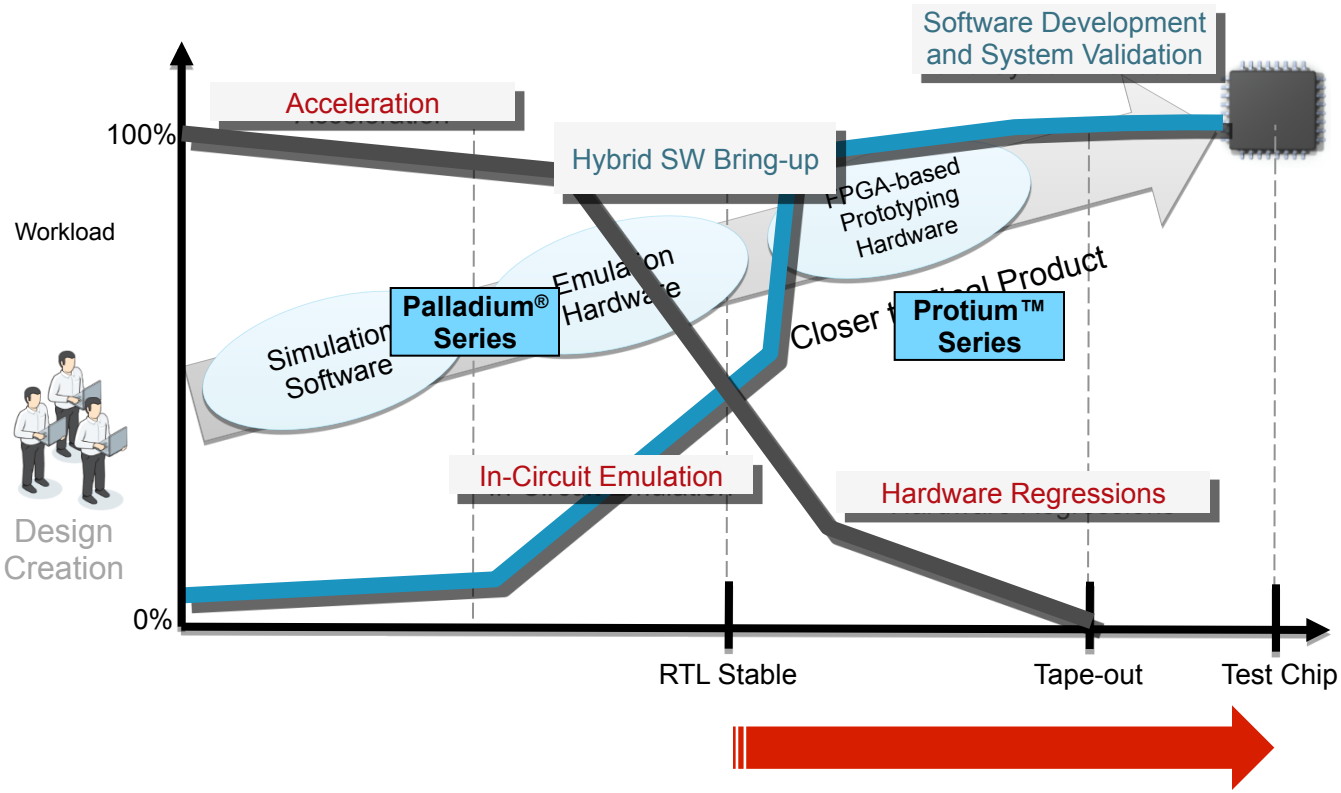
Bridging the productivity gap



Emulation Use Models

- Embedded software profiling
- Low power verification with CPF
- Low power verification with IEEE1801
- System & interface modeling
- Performance benchmarking
- Complementing ICE with virtualization
- Dynamic power analysis in a pre & post Si environment
- Coverage driven verification
- Fast OS validation with hybrid
- Verification acceleration with AVIP
- Accelerating pre-silicon SoC power profiling
- Transaction based acceleration

Emulation & FPGA Co-Exist



The roadmap is provided for informational purposes only and does not represent a commitment to deliver any of the features or functionality discussed in the materials.

Prototyping Use Modes

Early software development

#1

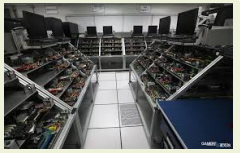
#2

HW/SW integration

System validation

#3

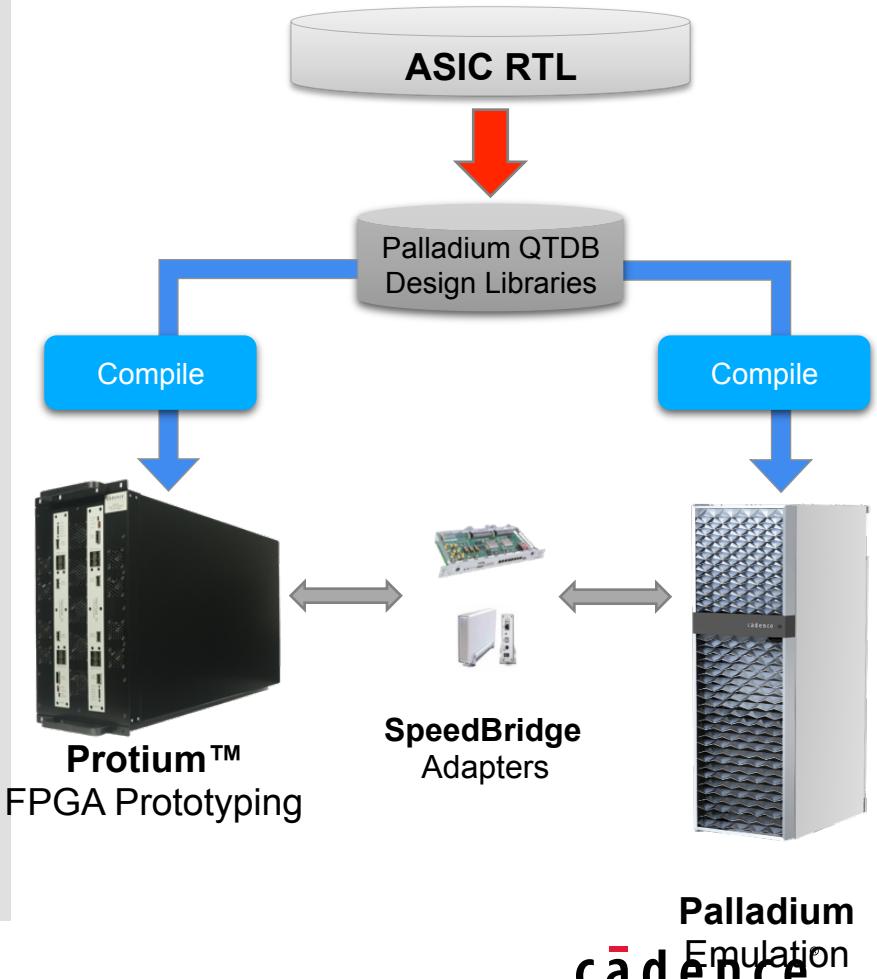
#4



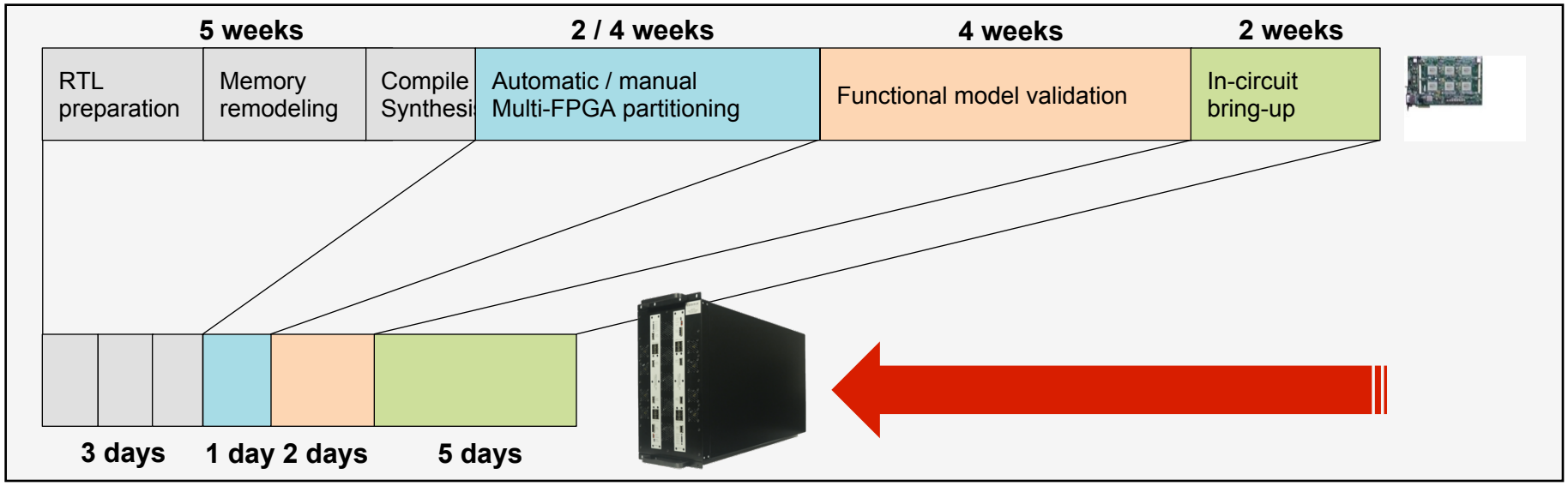
HW regressions

Common Compile Flow

- Reuse of the existing (Palladium®) emulation environment
  - Clock definition, memory models, clocking mode (CAKE1/2), scripts
- No learning of new tools and flows
- Congruency between emulation and prototyping
  - No guessing about which debug results are correct
  - Going back to emulation for detailed debug and failure analysis
- Fast & easy transition from emulation to prototyping
- Identical language coverage
  - No RTL changes required



Time to Prototype



- No FPGA -specific design/RTL modifications needed
  - Handling of any clocking structure and any number of clocks
  - Automatic memory compilation & modeling
- Fully automatic, multi-FPGA partitioning
  - Optional, manual guidance for performance optimization

- Pre FPGA P&R model validation
  - Multiple design integrations per day
  - Avoids time consuming FPGA P&R runs
- Fully integrated FPGA P&R
  - Automatic constraint generation
  - Guaranteed P&R success

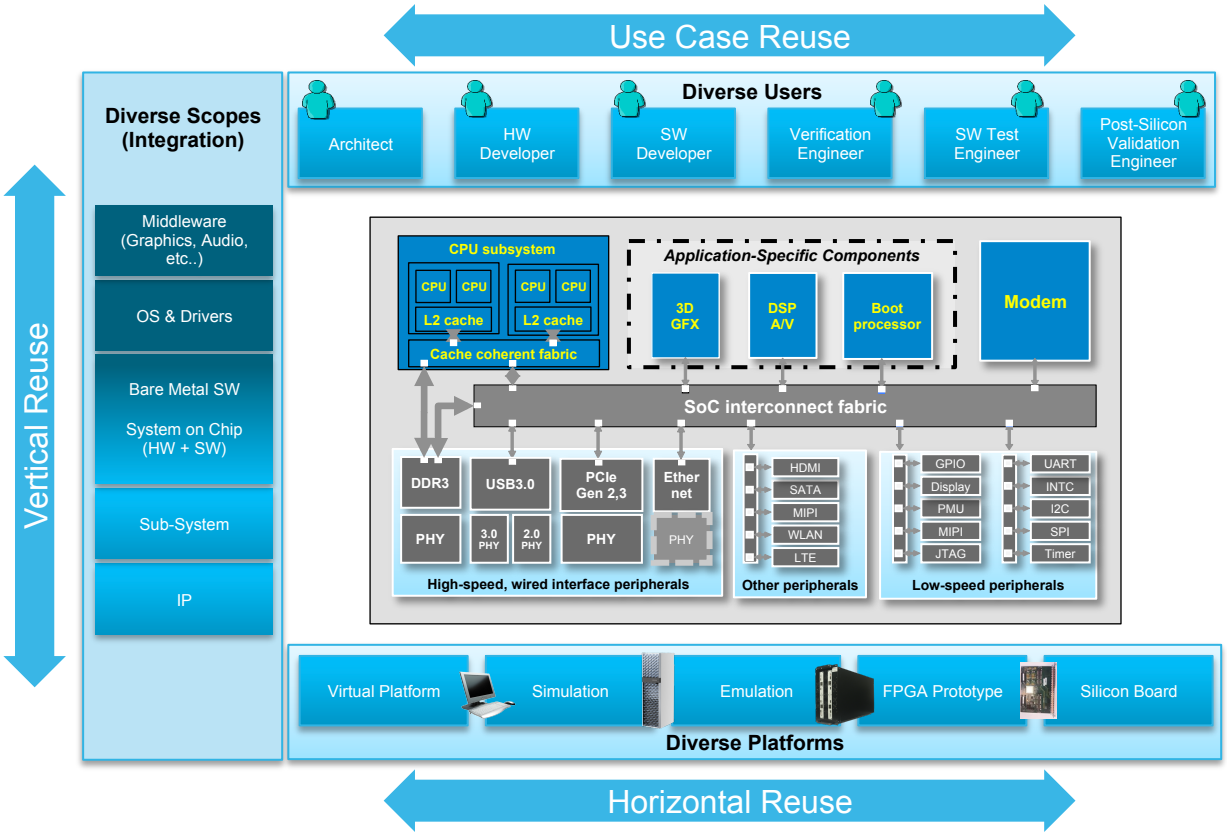
Tutorial Agenda

- Industry Status and Need for Change
- Integration Verification with Formal Technology
- Kicking Verification up a Notch with Multi-Core Simulation
- Leveraging Hardware: Acceleration, Emulation, and FPGA Prototyping
- **Automating Test Creation with Portable Stimulus**
- Bringing It All Together: Planning, Management, Coverage and Debug

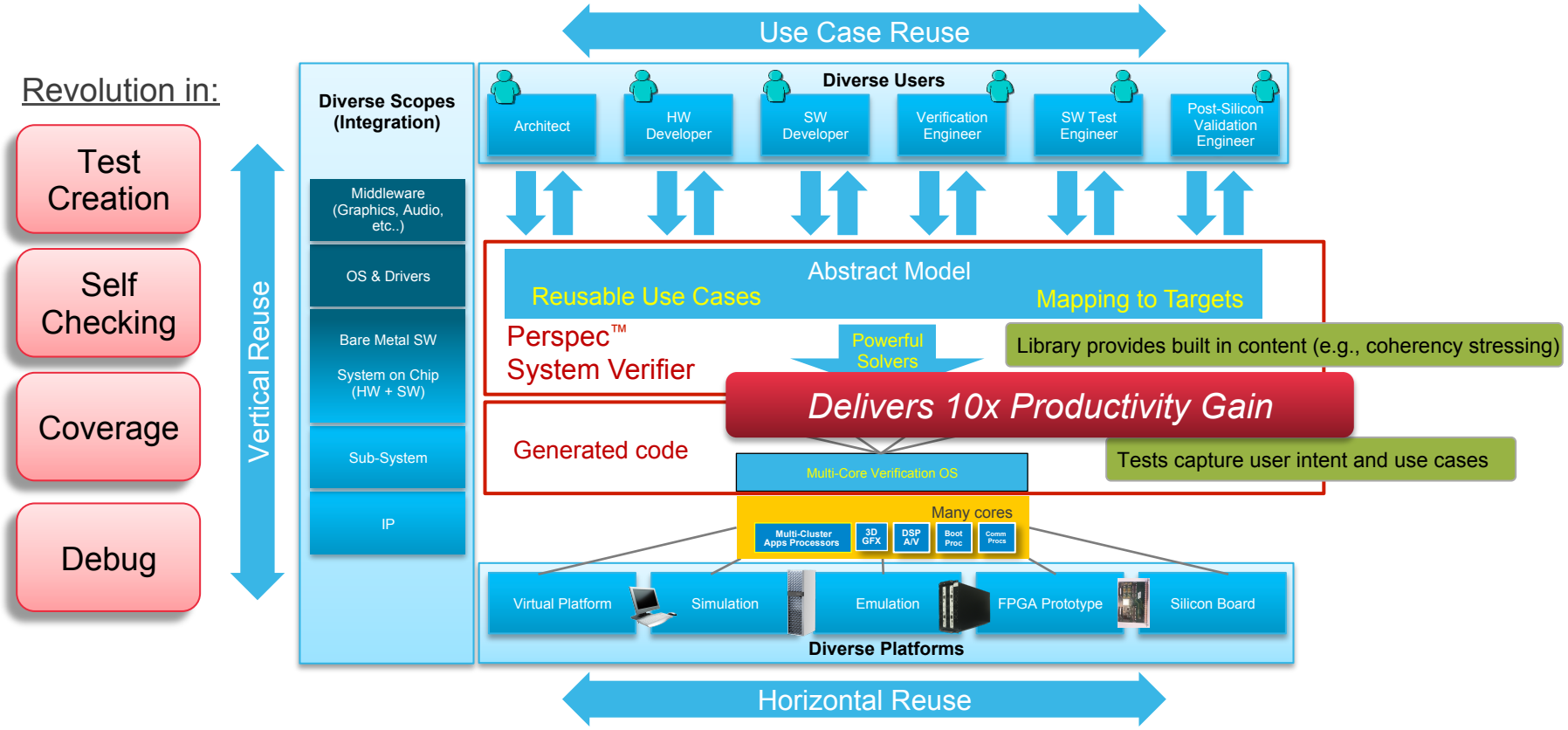
System-Level Scenarios

- The whole is greater than the sum of its parts!
  - And so are its bugs...
- Application use-cases involve multiple IPs interoperating
  - Example – read video off a mass-storage device, decode, split audio data from video frames, process by dedicated multi-media engines
- Stress and performance use-cases involve saturated utilization of shared resources
  - Example – all processors and DMA-enabled controllers access a certain memory controller in parallel
- System low-power use-cases need to be crossed with functional scenarios
- System coherency of caches/TLBs requires coordinated pattern of accesses from CPUs and non-processor masters

SoC Verification Needs to Address:

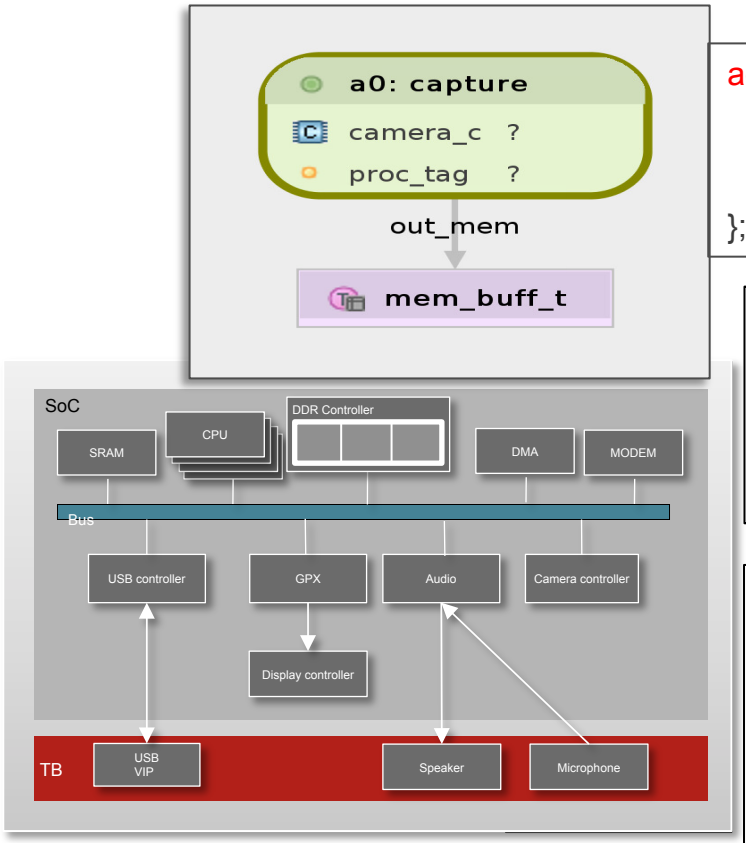


The Solution: Perspec™ System Verifier



All measurements as compared to hand-generated testcases on previous projects

Modeling Abstract System Behavior



SLN

```
action capture {
   output out_mem: mem_buff_t to mem;
   constraint out_mem.data.kind in [IMAGE, VIDEO];
   constraint out_mem.data.resolution in [LOW..MED];
};
```

C

Realization #1 (C):
call existing SW drivers

```
// Mapping to C
extend capture {
  exec body C #:
     capture_video(<(out_mem.addr)>, <(out_mem.length)>);
  end #;
};
```

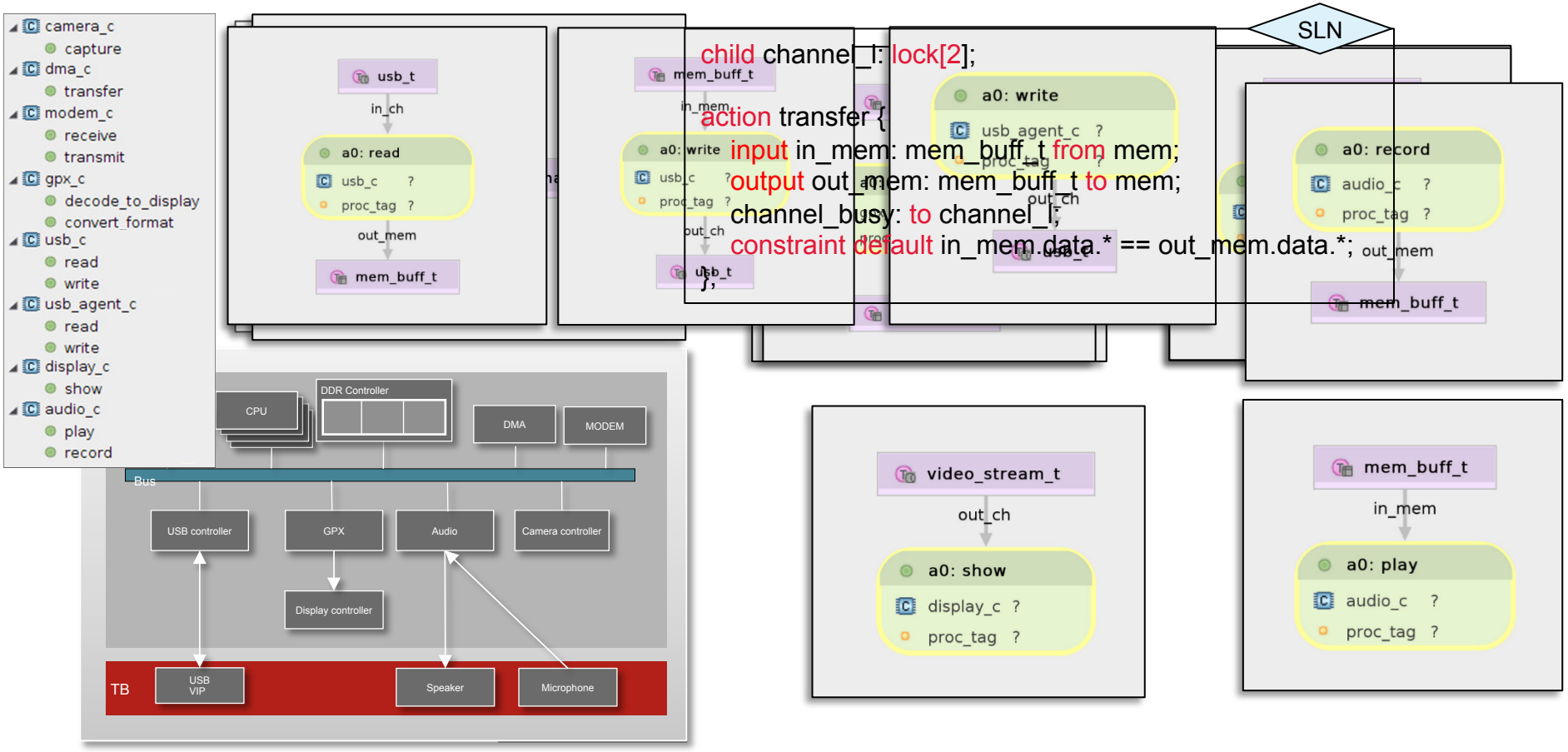
SV

Realization #2 (SV):
call existing SW drivers

```
// Mapping to SV
extend capture {
  exec body SV #:
     `uvm_do_on_with(capture_video, p_sequencer.ace_sequencer,
          {capture_video.addr ==(<(out_mem.addr)>;
          capture_video.length == <(out_mem.length)>;});
  end #;
};
```


cadence

Modeling Abstract System Behavior

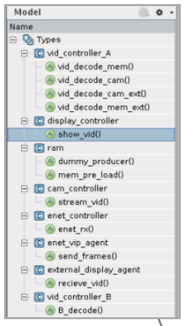


cadence

Productivity for System Verification

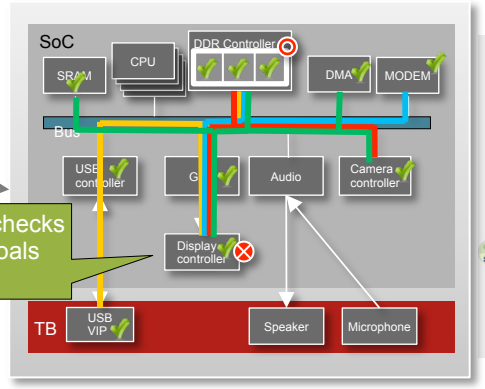
Auto-Completion of Partial Scenarios

SLN Models



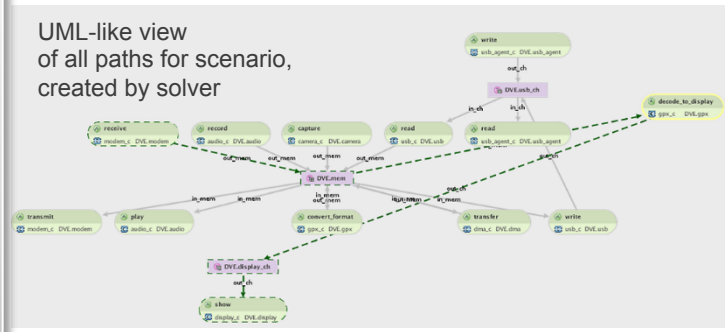
Desired scenario:

Decode video from the DDR and show on the display



Perspec solver checks feasibility of goals statically

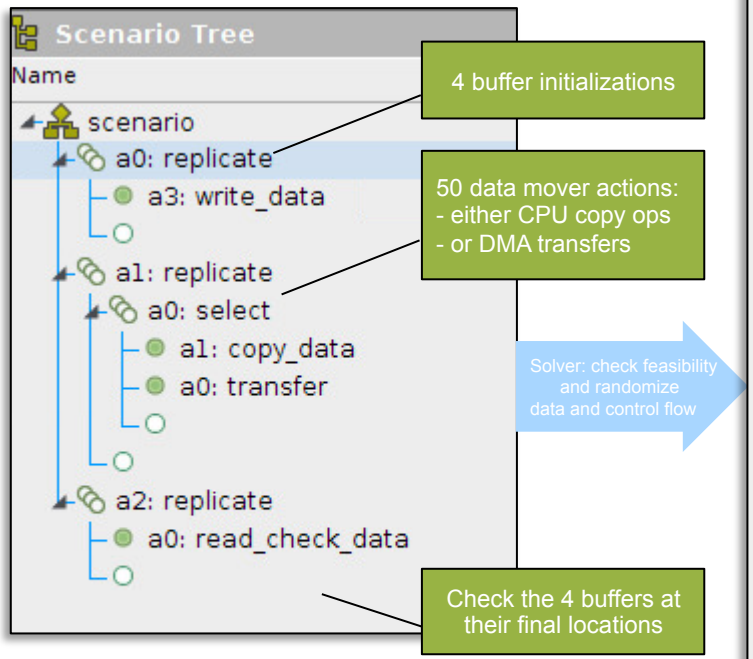
UML-like view of all paths for scenario, created by solver



***Automatically** and **exhaustively** complete the goals into full legal scenarios*

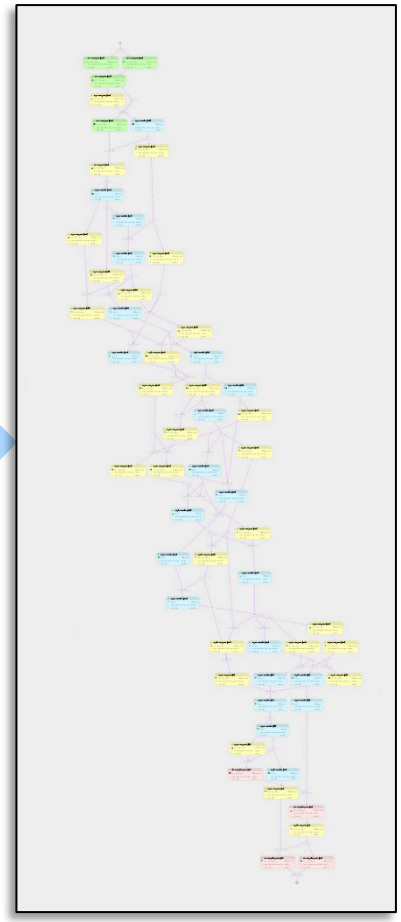
Use-Case Creation with Operators

Scenario Creation



User scenario specification

Solver: check feasibility and randomize data and control flow



Schedule elements

Data elements

Achieve massive high-quality system scenarios

Runtime (RT) Control Flow

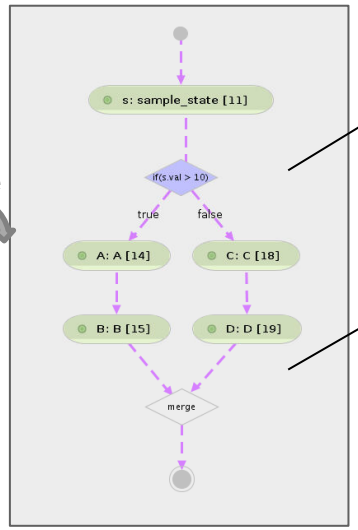
- Motivation
  - In post-silicon and ICE, we may need to run billions of actions
  - On all platforms, code may need to react to the state of the DUT
- RT operators enable scalability of tests for fast platforms
  - Combine intelligent solving with RT variations to achieve large scenarios
  - Reactivity and control flow in run-time
  - Stress condensed scenarios with minimal code size
- Enhanced ROI for user's fast platforms

Example:

runtime if

```
action cond_test {
  compound {
      > s: sample_state;
      > runtime_if (s.val > 10) {
        > then: serial {
          > A;
          > B;
        };
        > else: serial {
          > C;
          > D;
        };
      };
   };
};
```

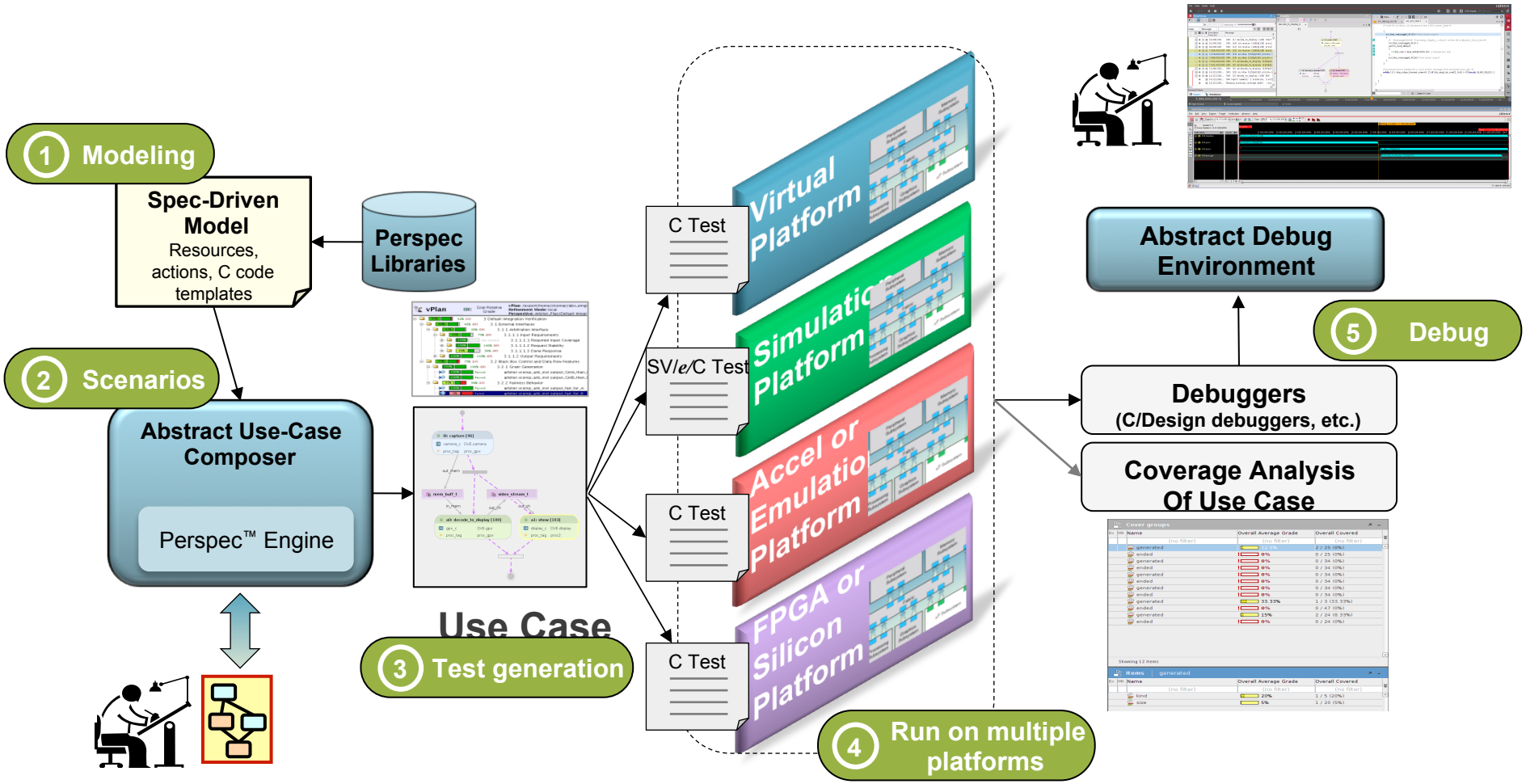
solve



Sub-activities under conditional execution branches

Rules are statically guaranteed for all possible execution flows

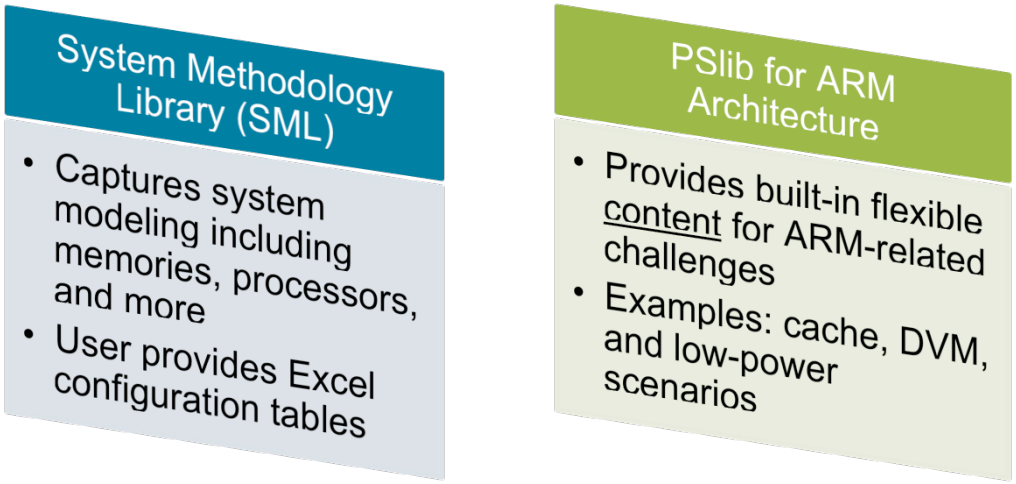
Portable Stimulus Usage Flow



cadence

Productivity from Built-In Content Libraries

- Requirements/opportunities:
  - Much of the SoC's logic is modeled the same way
    - It is possible to model the generic aspects of an SoC
  - Consistent coding style and methodology can improve readability and reuse
  - Can build libraries for cache, distributed virtual memory and low power logic
- Cadence libraries



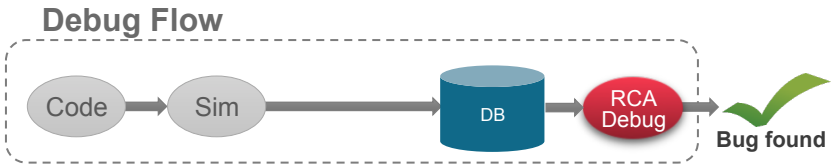
Tutorial Agenda

- Industry Status and Need for Change
- Integration Verification with Formal Technology
- Kicking Verification up a Notch with Multi-Core Simulation
- Leveraging Hardware: Acceleration, Emulation, and FPGA Prototyping
- Automating Test Creation with Portable Stimulus
- **Bringing It All Together: Planning, Management, Coverage and Debug**

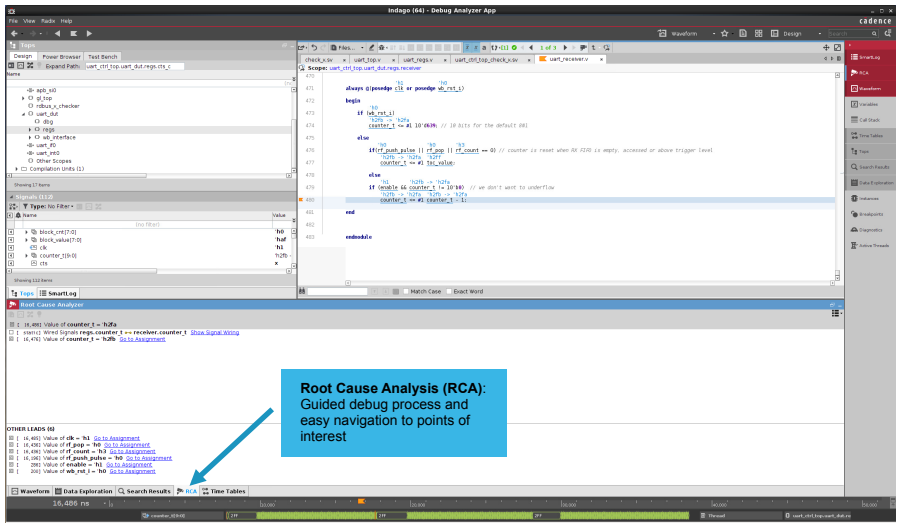
Debug Strategic Direction

- Provide the most productive debug tool in the industry by innovating
  - Patented Root Cause Analysis (RCA)
  - Smartlog identifies point of failure and more …
  - Leveraging market leading JasperGold® Formal technology in RTL debug
  - Optimized debug solution for multi-core simulation debug

- Significantly reduces debug effort via faster time to root cause
  - Full context views with one-click to point of interest
  - Synchronized waveform and source views
  - RCA to identify most likely causes

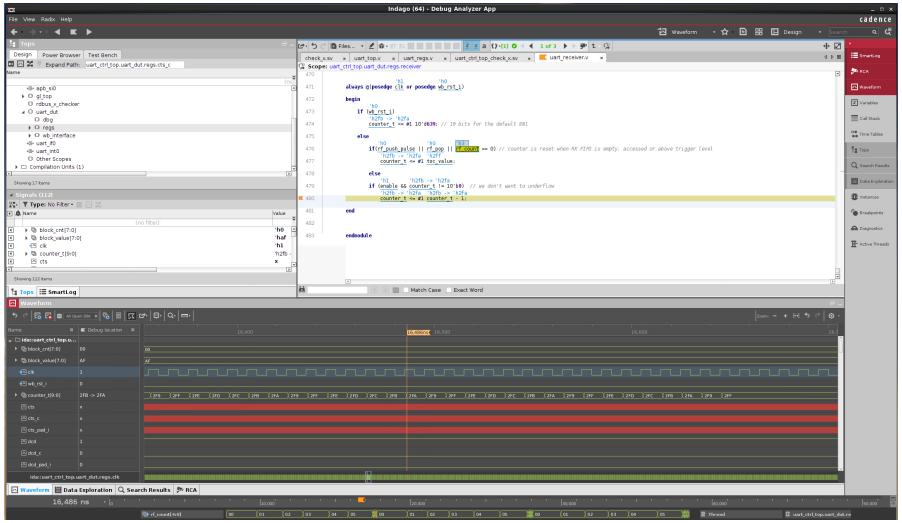


Faster time to bug found

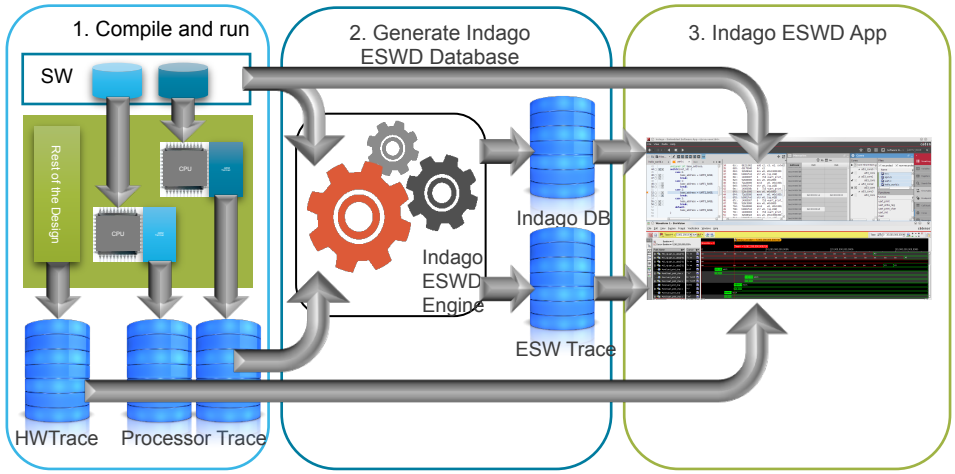


Saving debug time using innovative technologies

- Patented root cause analysis and data exploration speeds debug
- Formal technology powers expression calculator for “signal/assertion prototyping”
- Modern multi-threaded GUI prevents freezing on single operations
- Auto recognition of UVM errors to set debug starting point
- Pre-indexed, dedicated debug database for performance and scalability
- Smartlog of both SV and $display messages SV with single-click navigation to source and execution location



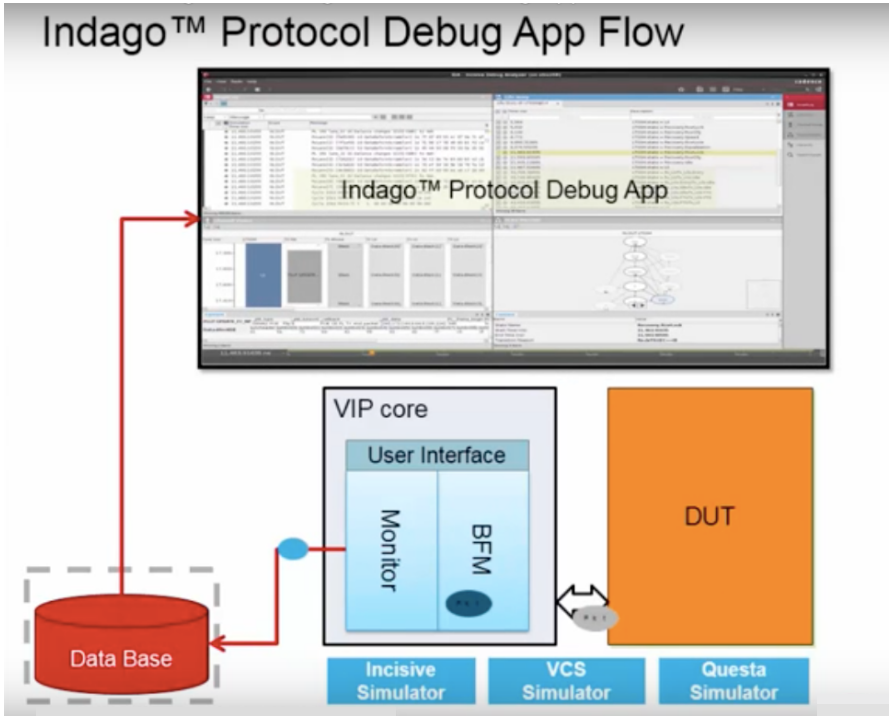
Embedded Software Debug Flow and Important Features



- Complete Context of SW execution
  - Timed view of SW source code execution by line
    - For all cores and all images
  - Source code lines executed
    - Navigate to next or last execution
- View of Memory from SW perspective
  - Memory view recorded based on processor register, instruction and memory transactions
- Ability to create custom visualization for faster debug
  - Calculated message – custom print of values and expressions on execution of any line of code
  - Color coded data patterns show in all source displays and waveform displays

cadence

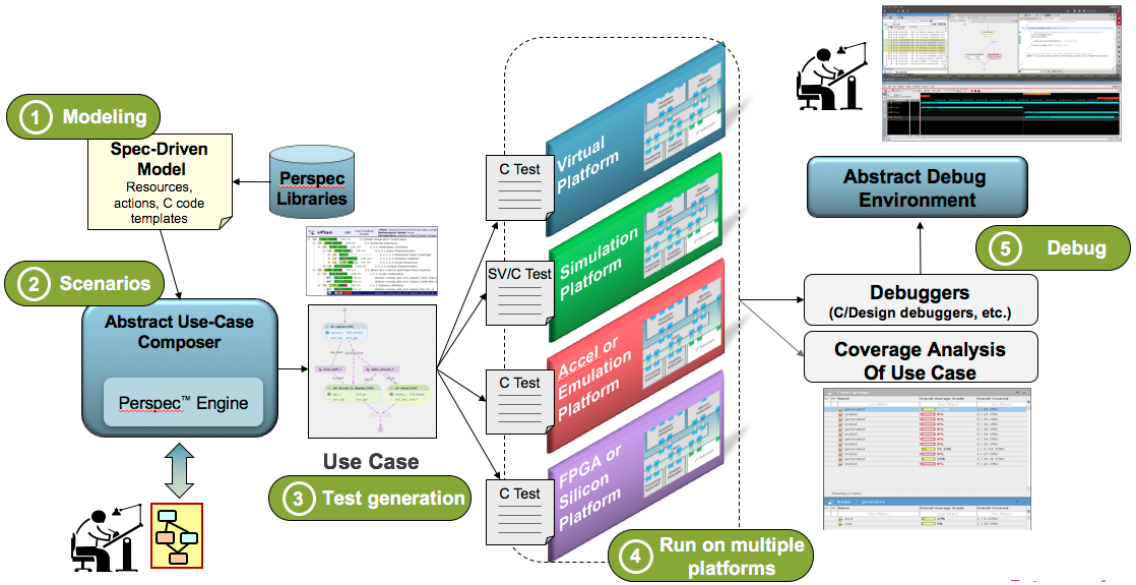
Protocol Debug Flow and Important Features



- Hierarchy view
  - Visibility into errors and warnings on all VIP instances
  - One-click navigation to error or warning location
- Smartlog
  - Messages have full execution scope and filtering
- Channel viewer
  - Visualization of protocol traffic
  - Highlight data resulting from specific command
- Life Story
  - How object changes during simulation
  - For transactions or statemachines

cadence

Portable Stimulus Debug Flow and Important Features



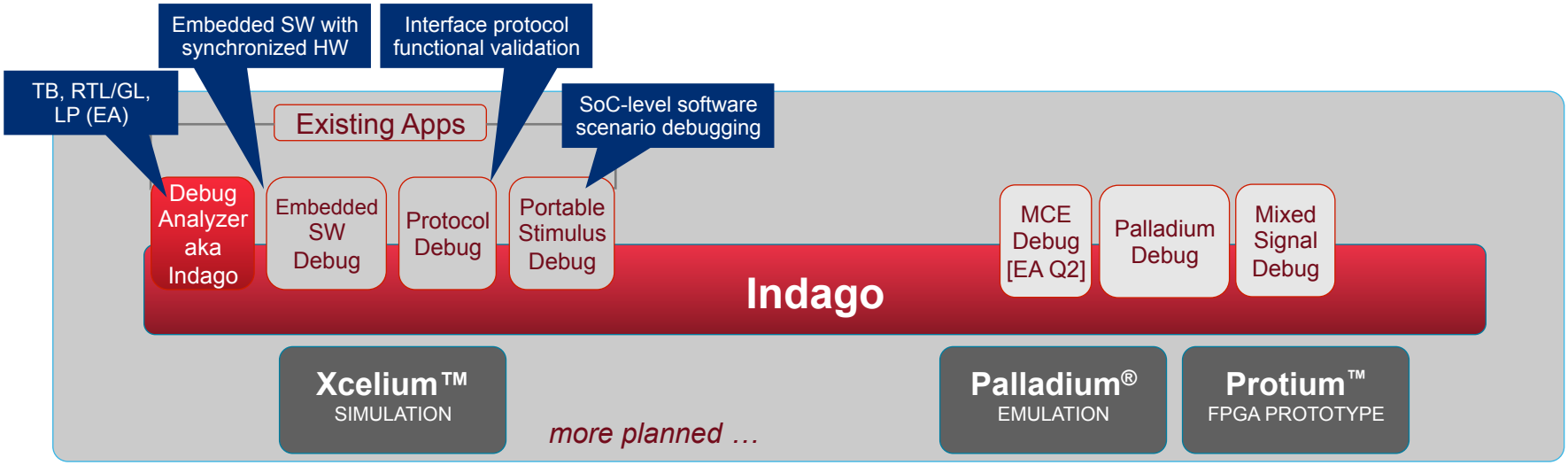
- UML activity Diagram visualization of multi-threaded test scenarios
- Waveform visualization of action executions in time by core
- Smartlog of all all action activations and completions

cadence

Indago Vision: The most productive debug tool in the industry

- 4 apps exist today

… with more features and apps on the way

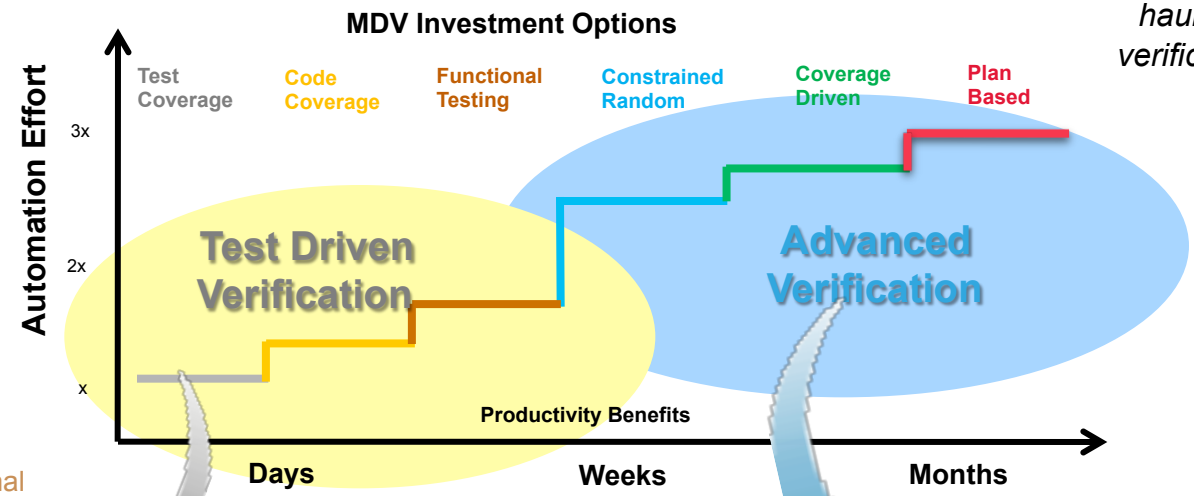


- Common platform with apps addressing specific debug tasks
- Indago is core design debug (RTL,GL,TB, LP, MCE) and common debug GUI

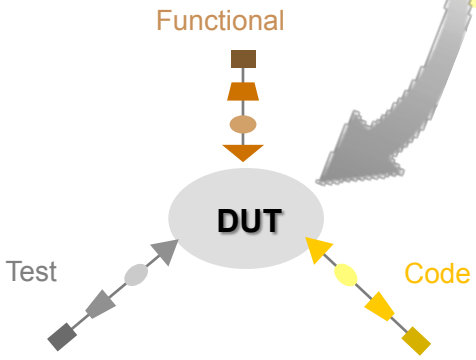
cadence

Getting Started with Metric-Driven Verification

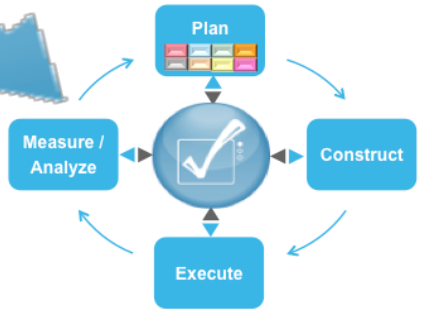
Users can invest incrementally or all at once when over-hauling the complete verification environment



- More effort, but more effective if resources permit
- Leverages advanced planning technology
- Greater ROI



- Quickly establish test area
- Take credit / track designers work
- Effective test driven verification environment



Metric-Driven Signoff

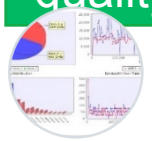
Quality via multi-engine metrics aggregated in vManager, IP to SoC

- Code **coverage**
- **Assertions**
- **IP** integration
- **Mixed signal** integration

Design quality

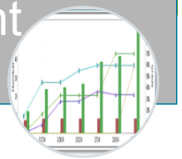
- **Power** management
- Interconnect **performance**
- **Protocol** compliance
- Requirement **traceability**

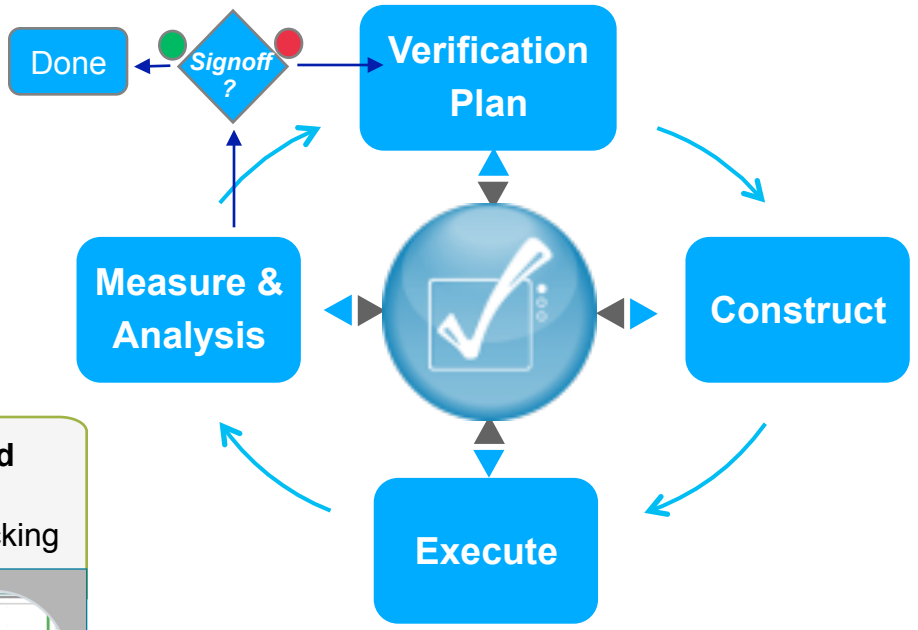
System quality



- Web-based **dashboard**
- Feature based **vPlans**
- Automated rollup / tracking

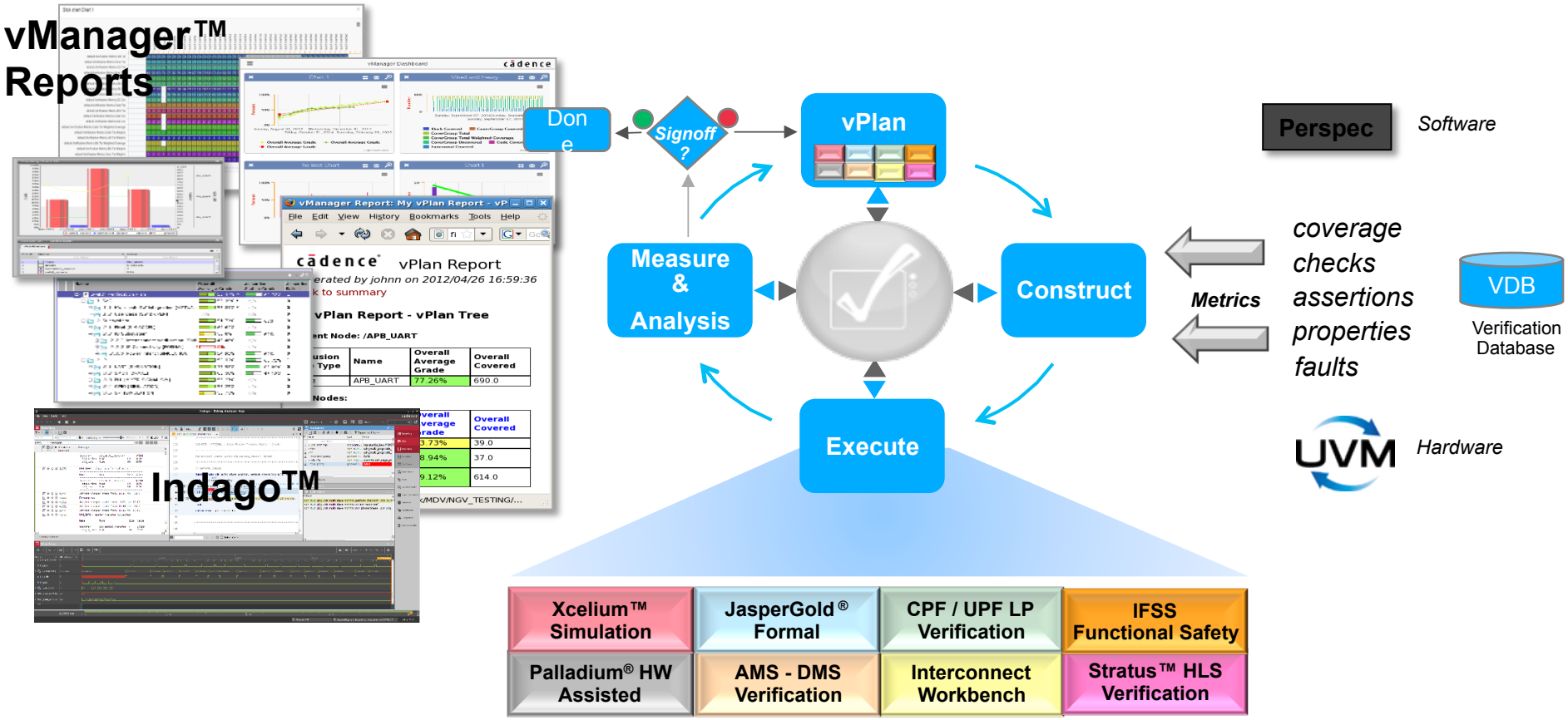
Management visibility



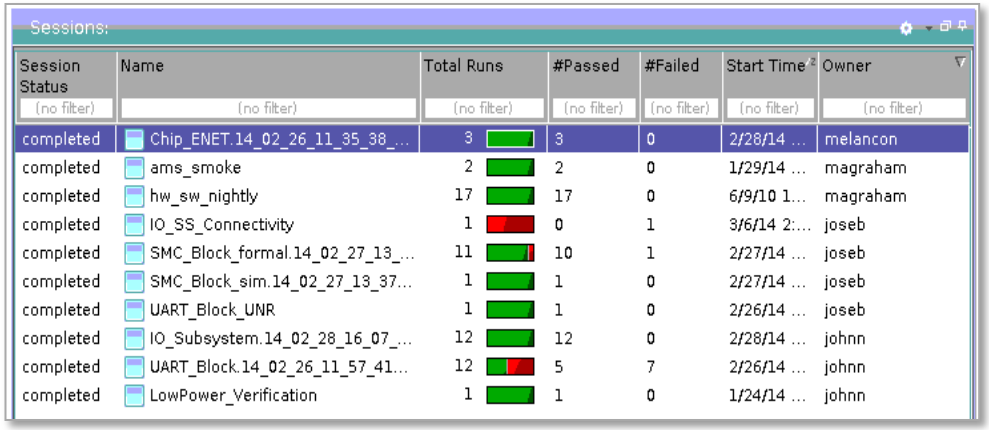


cadence

Plan-Based Multi-Engine Verification from Cadence – Complete!



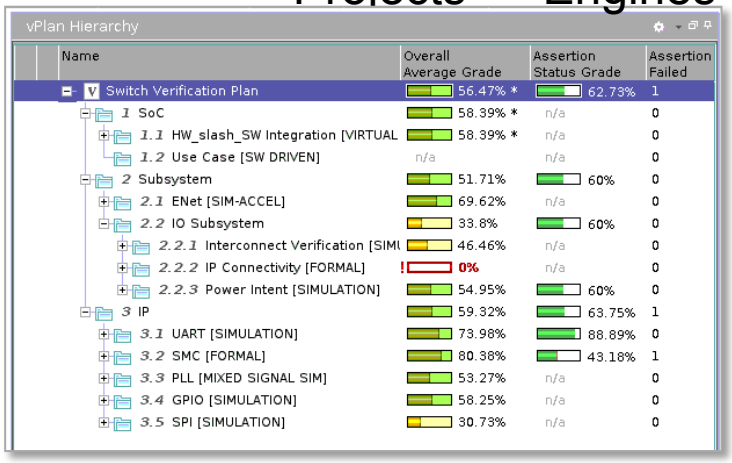
What an Integrated Metric-Driven Signoff Portal Looks Like



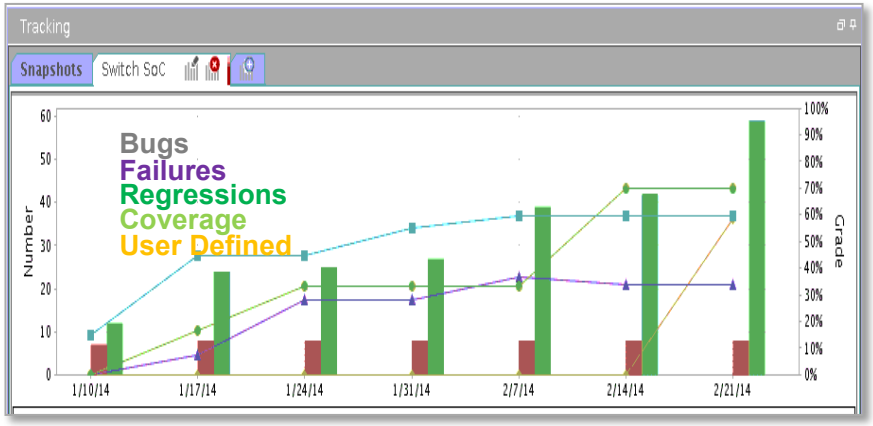
Multiple Users

Multiple Projects

Multiple Engines



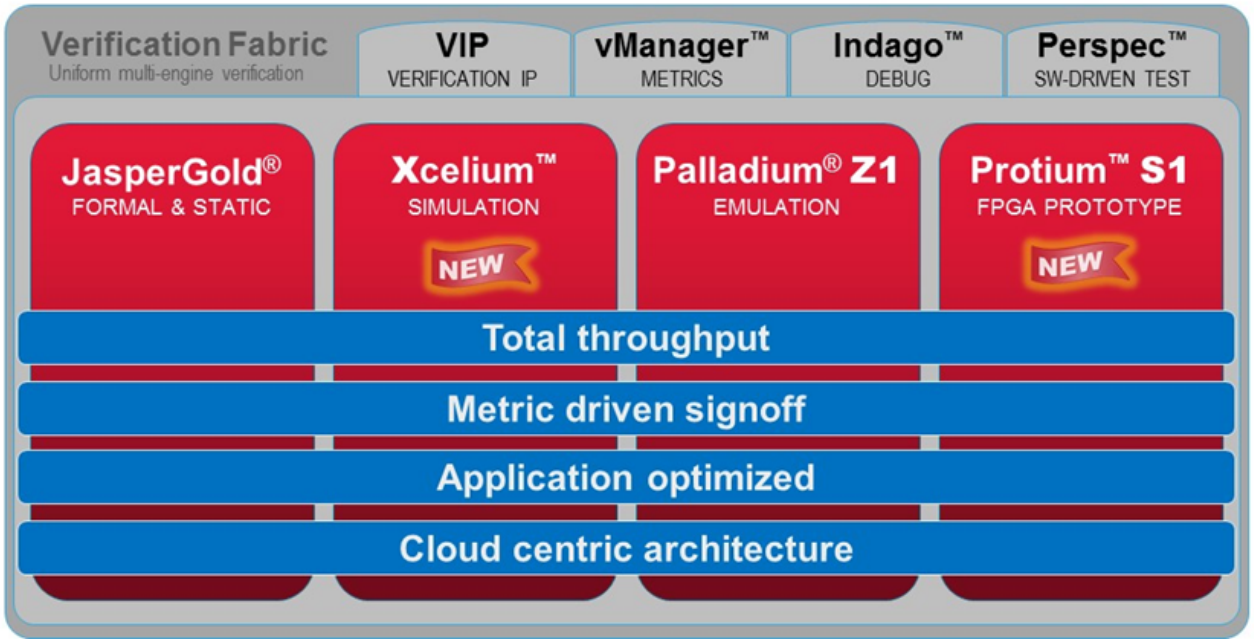
SIMULATION **FORMAL** **MIXED SIGNAL** **ACCELERATION**



Verification Signoff Data across Multiple Sites

cadence

Cadence Verification Suite



- ***Fast*** **Best-in-class engines**
- ***Smart*** **Flow-driven engine integrations**
- ***Optimized*** **comprehensive solutions**