Acceleration Startup Design & Verification

Tim Sun, Barry Yin, Haifeng Jiang
Agenda

• Enable SW-Driven Verification

• Accelerate Signoff with JasperGold RTL Designer Apps

• Speed Up Verification with Common Methodology For Emulation And Prototyping

• Accelerate SoC Performance Testing with System VIP
Enable SW-Driven Verification
Software cost is key challenge!

Source: IBS, July 2019
A Lot of Design Details Must Converge Successfully

Software

Hardware

Vehicle Software
- Vehicle Service Sensors, etc.
- Diagnostics, Calibration, Configuration
- Emergency Services

System Services
- IPC
- Linux

Infotainment Software
- UI with TTS & Speech Recognition

System Services
- IPC
- Android

DOMO
- Xen

Cloud
- Apps
- Wireless Display
- Sync
- DLNA
- Multi-media
- Radio

BT
- NAV
- Phone

Vehicle Service
- Sensors, etc.

Diagnostics,
- Calibration,
- Configuration

Emergency Services
- Phone
- BT
- NAV
- Cloud Apps
- Wireless Display
- DLNA
- Multi-media
- Radio

Xen
- Multi-media
- Radio
Hardware/Software Co-Verification During SoC Design

Applications (Basic to Complex)
- Virtual System Platform (VSP)
- Functional Simulation (Xcelium™)
- HW/SW Emulation (Palladium® Z1)
- FPGA Prototyping (Protium™ X1)

Middleware (Graphics, Audio)
- SystemC®
- RTL
- RTL
- RTL

Applications (Linux, Android)
- Bare-Metal SW

OS and Drivers (Linux, Android)
- Bare-Metal SW

Bare-Metal SW

Hardware/Software Co-Verification During SoC Design

Typical Duration:
- SoC Development: 6 month
- Front-End Design and Functional Verification: 12 month
- FPGA Prototyping: 4 month
- Chip Production: 12 month
- Silicon Bring-Up: 3 month

Applications (Basic to Complex)

Middleware (Graphics, Audio)

OS and Drivers (Linux, Android)

Bare-Metal SW

Software-driven design &

1st Silicon

Board
Virtual and Hybrid Platform Crucial for Software Speed

All Virtual
- Pre-RTL SW Development

IP Hybrid
- Pre-SoC IP / Driver Validation & Optimization
- RTL – Palladium or Protium

SOC Hybrid
- Pre-Tapeout Hardware/Software Validation

Emulation, FPGA Proto
- Pre-Tapeout Fully Accurate Hardware/Software Validation

All RTL, Silicon
- Final Hardware/Software validation

Index
- SW stack
- Virtual Models
- RTL Models

Shift Left
- Hardware/Software Validation
- IP Driver
- OS
- SoC Drivers
IP and SoC Hybrid Examples


- Linux >1B instructions
  - 2 min with virtualization
  - 45 min in RTL
- Android >20B instructions
  - ~45 min with virtualization
  - Hours in RTL
- Windows > 50B instructions
  - ~80 min with virtualization
  - Days in RTL

Source: DAC

SoC hybrid with faster OS boots, smoother SoC bring-up once silicon is back, SW ready to demo product earlier, OpenGL test suites pre-silicon
PreSilicon Software Validation Focus

- Fastest software execution across engines
  - Leveraging abstraction and connecting fastest engines
- Native software debug and configuration across engines and abstractions
  - Assembly and configuration for virtual and hybrid
  - Native SW debug across abstractions and run time engines
- Open platforms and models
  - Virtual Models based on standard SystemC TLM2
  - Run time / control framework natively integrated with a heterogenous model ecosystem
  - Tightly integrated with Cadence platforms – Palladium, Protium and Xcelium.
Leveraging Abstraction for Performance

- Virtualization (Models)
  - CPU and Interrupt Controller selection is key
  - Leverages TLM2 Direct Memory Interface (DMI)
  - Virtualize High Activity Peripherals
    - Timers, UARTs, Interrupt Controllers, …

- Virtual Reference Platform
  - Reference Platform running required software
    - Platform completeness depends on requirements
  - Start from required software and work back
    - Android -> which cpu -> which interrupt controller
  - Existing reference platforms
    - Android 9 and 10, Linux Multi-core, …

- Hybrid
  - Connecting Virtual Platform to RTL (Palladium/Protium)
  - Which interconnect, smart memory, …
Virtual Prototype Model Generation
Modeling Time Spent on Functionality

Register Descriptions & Configuration Options

- Cadence RDF
- IP-XACT

TLM Interface Generator

- IP-reg.h
- IP-test.c
- Reg.txt
- IP.CC

C-API for firmware register access

Pin change & IP register read/write test software

Register and IP function documentation

TLM 2.0 I/O, register definitions, and read/write functions

Generated TLM 2.0 software

Register Window GUI

Func.CC

Source functionality

Binary models
Virtual Platform Debug
SystemC/TLM Aware Debug

- SystemC Threads and Methods in sidebar
- Source Code View
- Call Stack
- Debug and Simulation Console
- SystemC Module Hierarchy
- Device Registers with Bit Fields
- SystemC/C++/C Variable Watch Window
- Activation of SystemC processes
Unified Hardware / Embedded Software Debugging
Fully Transparent and Integrated Across Both Domains

Combined Software / Hardware Debug View

- **Breakpoints & Probing**
  - Set break points in either hardware or software
  - Single step in hardware or software
  - Debug individual cores in multi-core system
  - View register and memory views for each core

- **Common Trace Views**

- **Software Debug View**

- **Hardware Debug View**

- **Source, Register, and Process Debugging**

- **System Memory & Memory Maps**
Ericsson use of Cadence Virtual System Platform

- Ericsson RBS 6000 Basestation
- Ericsson Helios Virtual Platform
  - Enables software developers to immediately begin SW development
  - High performance abstract TLM models available months before RTL
Intel Mobileye – Early SW development

Challenges in Pre-Silicon SW validation

- Software validation has a big impact on “time to market” of automotive products
- Pre-Si SW validation in HW/SW co-sim ensures a high SW quality on silicon arrival
- Requirements to shorten the SW turnaround time on Si:
  - Fast bring-up / High execution frequency / Quick turnaround time / Accuracy / Debug
- What is the best Pre-Si platform for fast SW convergence?
- What is the right methodology for the job?

<table>
<thead>
<tr>
<th>Platform</th>
<th>Benefit</th>
<th>Limitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual Simulator</td>
<td>Early, fast, high debug</td>
<td>Timing accuracy, not all IP available/effort porting IP to TLM models/ no co-verification of SOC – RTL/SI</td>
</tr>
<tr>
<td>RTL Simulator</td>
<td>Accurate, high debug</td>
<td>Extremely slow, impossible for complex software runs</td>
</tr>
<tr>
<td>RTL Emulator</td>
<td>Accurate, high debug</td>
<td>x1000 faster than RTL but still not enough for SW convergence</td>
</tr>
</tbody>
</table>

Results

- Full Linux OS boot in **32 seconds** instead of 2-3 hours on non-hybrid emulation
- **High level** of software validation and readiness even before Si arrival
- Validate **complex software and complex scenarios**
- Validate **SW drivers** for peripherals in OS context: SPI, eMMC and PCIe on RTL
- **Shorten** SW bring up time for Si
- **User friendly** environment for both HW and SW debug
- **Same SW** image used as for pure emulation
- **Quick retest** of new SW image
- Gate count of hybrid design is 37% lower

Hybrid advantages from the SW programmer’s point of view

- **vs. Pure Emulation**
  - Dramatically reduced boot time
  - Interactive mode of operation, control OS (Linux) via console
- **vs. Pure Simulator**
  - Much more precise model
  - No need to develop any model for peripherals
- **vs. Silicon**
  - Schedule “Shift-Left”: SW enablement before Silicon arrival
  - Enable error injection to test software bad path scenarios
  - Convenient waveform debug of HW signals/register values (key to debug HW/SW issues)
Commitment to Software-Driven Verification

- Enable early SW development and validation
  - Based on SystemC models or CPU fast models
  - When only some key IP(s) ready
- Native software debug and configuration across engines and abstractions
  - Assembly and configuration for virtual and hybrid
  - Native SW debug across abstractions and run time engines
- Open platforms and models
  - Virtual Models based on standard SystemC TLM2
  - Run time / control framework natively integrated with a heterogenous model ecosystem
  - Tightly integrated with Cadence platforms – Palladium, Protium and Xcelium.
  - Fastest software execution across engines
Accelerate Signoff with JasperGold RTL Designer Apps
Need for Early Design Checking

- Effort and cost to fix a bug increases significantly further into the development cycle

Effort to fix bug

Catch bugs as early as possible

Time

- RTL Design
- Testbench Development
- RTL Refinement
- IP Verification
- Integration
RTL Signoff by Designers: Big Picture

- Typical RTL signoff includes
  - Signoff the RTL against a comprehensive set of structural lint/DFT/CDC/RDC checks
  - All checks are ‘automatic’ and user provides RTL + constraints (for DFT and CDC/RDC)

- For comprehensive signoff, augment with automatic functional checks
  - High-value code reachability and functional checks
  - Functional CDC/RDC checks

- JasperGold® platform uniquely positioned to give true integrated solution for

  AUTOMATIC STRUCTURAL CHECKS

  AUTOMATIC FUNCTIONAL CHECKS
Auto-Formal Checks: FSM
Reachability/Deadlock Example

- Structural LINT cannot catch such issues
- These issues are high value to caught at RTL stage and can be automatically checked using auto-formal checks

Once FSM enters CHECKSUM and enq_sop_id never goes high, CHECKSUM will be deadlocked.
Domain-Crossing Verification – Gaps in Conventional Flow

1. Correctness of analysis constraints
   - User-specified constraints are considered golden
   - Reuse of constraints – a big source of bugs
   - Are my constraints correct?

2. Validity of waivers
   - Assumptions made about design functionality
   - Reviewed based on waiver comments
   - Are the underlying assumptions for my waivers valid?

3. Metastability-aware verification
   - How do I guarantee that my design is immune to metastability effects?
   - Is my functional verification environment metastability aware?
   - Generally modeled with randomized synchronizer delay in simulation – Pessimistic and incomplete

Linking CDC/RDC analysis with functional verification is the key
Validity of Waivers

• RDC violation: Destination flop (Q2) may go metastable if source reset (RST1) is asserted while destination reset (RST2) is de-asserted
• Considered safe if reset isolation logic is present
• Assumption: The RDC path (D2→Q2) cannot be sensitized when reset isolation is active
• Is this assumption valid?

Missing verification linkage to validate waiver related assumptions
JasperGold Superlint App

Structural Lint and DFT Checks

Automatic Formal Checks

DFT controllability
Sim-synth mismatch
Coding style
DFT observability
X assignment
Reachability
Bus contention
Livelock/deadlock
Arithmetic overflow
Range overflow
Combo loop analysis

Enabled by true formal technology
Best-in-class debug
Low-noise violation and waiver handling
JasperGold® Visualize™ Environment

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

Rules Configuration
- Custom categories
- Selected checks
- Custom parameters

Design Configuration
- Clocks
- Reset
- Signal configuration

Run
- Lint
- DFT
- Auto-Formal

Violation / Progress Report
- Analysis of run results
- Debug violation and failing properties

What was achieved
- What is pending

Signoff Report

Revise RTL
Revise Waivers

Auto-Formal violations come up in the same violation tree as LINT/DFT

Waivers
Persistent Waivers

Waivers are persistent across RTL code/file path and hierarchy changes, waivers are highly portable as those are stored as Tcl commands.
Observability-Enabled Debug

- Debugging auto-formal violations show where the violation propagates
  - Boundary signals are added to the debug window
  - The propagation cycle is highlighted – hints the user to perform “why”

Leverage the JasperGold® platform best-in-class Visualize™ debugging features

Use ‘why’ to debug why the signal gets the value which leads to failure

Failure trace is automatically extended by one cycle to show where the failure is observed

Analyze in the source browser, can trace loads and drivers
JasperGold CDC App - The Complete Picture

The only CDC solution with industry-leading formal technology for functional checks, asynchronous verification, and waiver handling.

Comprehensive Checks

- Wide range of synchronizers (NDFF, Mux, FIFO, Handshake, User-defined…)
- Automatic structural checks
- Functional checks (Gray, FIFO…)
- Low-noise violation handling
- Powerful auto-waiver feature
- Constraint Validation

Metastability-Aware Verification, Innovative Debug, and Waiver Handling

- Metastability Modeling and Injection
- Metastability Injection in Simulation
- Innovative debug with Visualize™ and Graph views

Enabled by true formal technology

Clock/Reset Tree

Convergence/ Glitch

Assertion generation and export to simulation
True CDC/RDC Signoff Flow with JasperGold CDC App

- **CDC Configuration**
  - Clock specifications
  - Reset specifications
  - Signal configurations

- **Structural Analysis**
  - Clock/reset tree checks
  - CDC synchronization checks
  - Convergence/glitch checks
  - Reset synchronization checks
  - Reset domain crossing checks

- **Functional Analysis**
  - Signal configuration validation
  - Clock relationship checks
  - Reset order/priority checks
  - CDC protocol checks
  - Waiver condition validation

- **Metastability Analysis**
  - Metastability-aware proof of:
    - User-written assertions
    - CDC protocol checks
    - Waiver conditions

- **Simulation**
  - Metastability-aware simulation
  - CDC protocol checks
  - Signal config checks
  - Waiver conditions

- **Auto/Conditional waiver flow**
- **Waivers**
CDC Configuration and Structural Analysis

- **CDC configuration**
  - Specify clock properties and relationships (using SDC or native commands)
  - Specify correct reset types (async/synchronized/synchronous) and reset priority
  - Declare signal configurations (constant/static/mutex/gray-code) with conditions

- **Comprehensive structural analysis**
  - Clock and reset tree checks
  - CDC synchronization checks
  - Convergence/glitch checks
  - Reset, RDC checks
  - Waivers added while dispositioning structural violations

**Garbage In = Garbage Out**

**Wrong Waivers = Masking Issue**

Signal configuration correctness and waiver-related functional assumptions should be verified
Functional CDC Analysis

Signal configuration validation
- Auto-generated assertions for verification
- Proven in formal verification environment

Verify *pseudo* nature of constraints
- Specify triggering event for signal to be constant/static
- Example:
  - Data on CLK1 domain can change only when destination is in reset (RST2 is asserted)
  - `check_cdc -signal_config -add_static D2 -condition RST2`
- Additional verification to ensure pseudo-static property

Export and run signal configuration checks in simulation

Verification of analysis constraint correctness leads to greater confidence!
Functional CDC Analysis (cont)

- Waiver condition validation
  - Validation of functional assumptions used in dispositioning structural violations
- Auto-waiver flow
  - Tool automatically detect conditions for dispositioning structural violations
  - Violations are waived if the condition is proven
  - Example: Gray encoded buses, unsynchronized paths but are metastability safe
- Conditional waiver flow
  - User provides SVA expression for waiver condition
  - User-selected violations are waived only if the condition is proven

% check_cdc -waiver -add -filter ..............
  -comment "stable Q2 during RST1"
  -expression {##1 $fell(RST1) && $stable(RST2) |-> $stable(Q2)}
% check_cdc -waiver -prove

Verification reduces noise and avoids errors in manual dispositioning of violations
Metastability Injection (MSI) Flow in Formal

- Pre-requisite: Formal Property Verification (FPV) environment
  - Passing functional assertions
- Push button flow with customized debug in visualize
  - Inject metastability in user properties
  - Verify user-defined properties in presence of metastability
- Metastability awareness in protocol checks
MSI Flow in Simulation

- Pre-requisite: Simulation environment with passing test cases
- Simple, easy to use, no instantiation flow in simulation
  - Timing violation monitors and injection modules exported from JasperGold® platform
  - Injection models can mimic both setup and hold violations
  - Not dependent on synchronizer types – all CDC crossings covered
  - Configurable setup, hold time windows for individual clocks
- Random/Always/No Injection modes
Conclusion

• JasperGold® Superlint App is industry-leading solution for RTL signoff by designers
  ✓ Comprehensive structural LINT and DFT checks
  ✓ High-value auto-formal checks
  ✓ Easy setup and feature-rich analysis and debug environment
  ✓ Designed to be low-noise, high-productivity application

• JasperGold® CDC App is a holistic CDC/RDC verification solution
  ✓ Comprehensive structural checks
  ✓ Functional CDC/RDC verification
    • Constraint validation
    • Waiver validation
    • CDC protocol verification
  ✓ Metastability-aware verification
Speed Up Verification with A Common Methodology For Emulation And Prototyping
System & Chip design trend 2021 - 2025

- Time-to-market
- Development cost reduction
- Multi-core design and verification complexity
- Integration of new designs and derivatives
- Software stack development
- Hardware-software convergence
- More than 80% re-use
- More than 60% of effort in software
Emulation & prototyping accelerate time to product (revenue)

- Early, embedded software & firmware development
- Initial systems and/or proof of concept
- Pre-silicon chip (ASIC) verification

Software is ready when 1st silicon comes in
- Full Android boot 30 min after silicon is back
- Full system demo to customer in 3 days

Source: Amlogic
The Right Tool, for the Right Job, at the Right Time!

Pre-Silicon

- RTL verification
- Performance Analysis
- Simulation Acceleration
  - Predictable fast build
  - Comprehensive debug
  - Multi-chip level capacity

Post-Silicon

- Power Analysis
- Early HW/SW Co-verification
- Firmware & software validation
  - HW regressions
  - Highest performance
  - Lowest cost
  - Scalable capacity

Usage

- Chip/System RTL Ready
- Tape Out
- 1st Silicon
- Product Release

Accellera Systems Initiative
Palladium Z1 to Protium X1 benefits

• Unified Flow
  o Reuse of the existing Palladium environment (clocks, memory models, scripts, AVIPs, SB/EDK, etc.)
  o Congruency between emulation and prototyping
  o Going back to emulation for detailed debug
Palladium Z1 Emulation Drivers

Multi-Project Use – Emulation Farms

Multi-Use-Model Versatility

Core Emulation

Virtualization

Architecture

Low Power

Acceleration

Hardware/Software

Post-Silicon

Palladium Z1 key characteristics

• Scalable datacenter-class emulation system
  o IP to full SoC emulation: 4 to 576 million per rack
  o Scales up to 9.2BG with up to 2,304 parallel jobs
  o Rack-based form factor: setup in existing data center
  o Built-in redundancy for reliability

• Virtualization
  o Virtual target relocation
  o Advanced job reshaping
  o Emulation Development Kits (EDK)
  o Virtual Emulation and Virtual Debug
Palladium Use Modes
Simulation Acceleration, Virtual and In-Circuit Emulation

**Simulation Acceleration**
- TB-driven HW verification

**Virtual Emulation**
- SW-driven HW verification
- Interface-driven HW verification
- HW/SW co-verification

**Palladium**
- Embedded Device Models

**In-Circuit Emulation (ICE)**
- SW-driven HW verification
- Interface-driven HW verification
- HW/SW co-verification

**Test Environment**
- Physical Devices
- Application / Tester

**Physical Interfaces**
- Embedded Device Models

**Virtual Interfaces**
- Test Environment
  - Simulator
    - Test-Bench
    - UVM-A, C/C++

**Test Environment**
- Virtual Device Models
- Application / Tester

**Test Environment**
- Fast Compile
- FullVision
- Coverage
- Multi-run Modes
- Low-power DPA
### IXCOM Modular Compiler, VXE21.02

<table>
<thead>
<tr>
<th>Compile time</th>
<th>Fclk</th>
<th>% Faster</th>
</tr>
</thead>
<tbody>
<tr>
<td>5:42:08</td>
<td>424KHz</td>
<td>71%</td>
</tr>
</tbody>
</table>

- **Main workstation:** PD02, 96-core, 2TByte
- **PPC workstation:** PD03, 72-core, 3TByte

### IXCOM Parallel Partition Compiler VXE 19.10

<table>
<thead>
<tr>
<th>Compile time</th>
<th>Fclk</th>
<th>% Faster</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:19:31</td>
<td>541KHz</td>
<td>57%</td>
</tr>
</tbody>
</table>

- **Main workstation:** PD02, 96-core, 2TByte
- **PPC workstation:** PD03, 72-core, 3TByte

Running 18 partitions on PPC workstation

### IXCOM standard compile VXE 19.10

<table>
<thead>
<tr>
<th>Compile time</th>
<th>Fclk</th>
<th>% Faster</th>
</tr>
</thead>
<tbody>
<tr>
<td>19:47:25</td>
<td>786KHz</td>
<td></td>
</tr>
</tbody>
</table>

- **Workstation:** PD02, 96-core, 2TByte

- **Z1 configuration:** 8-clusters
- **MC workstations:** Machine farm through LSF, 1TByte
Palladium Debug
Unparalleled levels of productivity and at speed

**FullVision** View any design signal at full speed. Debug design

**Dynamic Probe** Capture select signals with large traces (up to 80M samples).

**InfiniTrace** Capture extremely long trace depth for post-analysis replay, move forward & backward to debug any time window of interest

**Offline Debug** Offline concurrent debugging on workstation. Free up Palladium resource for other jobs, jump to any time window using a specific trigger

**Waveform streaming** Continuously view small number of signals at full rate

**Save/Restore** Re-start emulation run from a previous state. Save time by restoring, avoid repetitive initializations or sequences

**Hot-swap** Swap state of design back to simulator for interactive debugging to free up Palladium resources for non-interactive jobs

**Force** Change design function during runtime. Set system conditions to analyze design behavior under un-modeled corner cases

**SDL/DRTL** Specify multi-level complex trigger conditions. Use design events to detect scenarios,
Palladium coverage support

- Palladium supports assertions, code and functional coverage
  - Scored in hardware & viewed in software
  - Supports acceleration & in-circuit emulation

![Diagram](image)

- By applying coverage in Palladium
- By applying coverage in simulation

- Accelerate performance-challenged simulations with assertion, code or functional coverage requirements
- Detect gaps earlier and improve overall verification efficiency
- Visualize coverage in Xcelium Metric Center (IMC) or vManager
Protium X1 Enterprise Prototyping System

- **Performance**
  - Enabling early firmware and software development, automated bring-up
  - Up to 50MHz for single FPGA; up to 5MHz on billion gate designs

- **Capacity**
  - Advanced blade architecture scales to billions of gates
  - Ideal for AI, ML, 5G, mobile, and graphics applications

- **Fast Bring-up**
  - Unified Palladium® Z1 / Protium™ X1 compile ensures DUT congruency
  - Enables transition from emulation to prototyping in days

- **Multi-user**
  - Single-FPGA granularity assures high utilization and efficiency
  - Ideal for storage, automotive, image, consumer and medical applications
Technology Details – Faster Prototype Bring-up

- **Unified compile**
  - With Palladium® Z1, Protium™ S1
  - Bring-up in days or weeks (vs. months)
  - Re-use of existing environments
  - Easy transition from emulation to prototyping

- **Enhanced memory modeling**
  - Map virtually any design memory
  - New: LPDDR5, UFS 3.0 and HBM
  - Future protocols can be added quickly

- **SpeedBridge® Adapters**
  - Connect to peripherals quickly
  - Without manual rate-adaption
Technology Details – Advanced Debug Capabilities

• Hardware debug: bring-up design quickly, validate functionality
  • Force and release: for initialization and “what-if” analysis
  • Data capture card (DCC): 1000’s of signals, millions of cycles
  • Prototyping Full Visibility: observe any signal at any time

• Software debug: early firmware and software development
  • Memory (backdoor) upload and download
  • Clock control to stop and resume the hardware at any time

• Standard interfaces to industry-leading debuggers and software environments - use familiar tools
  • Joint Test Action Group (JTAG) and Universal Asynchronous Receiver/Transmitter (UART) interfaces

• Transaction interface
  • Directly connect to software models and virtual environments
Technology Details – Multi-user Functionality

• Prototyping of smaller IP or subsystems
• Flexible multi-user capability for up to 6 concurrent users per blade
  • Any single-FPGA granularity combination is possible – no restrictions
  • “Cloning” of design is supported (same design in multiple FPGAs) without recompile
• Enables optimal utilization for IP, IoT, storage, automotive, image, consumer and medical designs
Shared solutions and peripherals

- **EDKs and SpeedBridge Adapters**
  - Physical connection to real-world interfaces

- **Virtual Solutions**
  - VirtualBridge, AVIP, Hybrid, Virtual Debug
  - InfiniBand-based link to virtual interfaces

- **Memory Model Portfolio (MMP)**
  - For all on-chip and off-chip memories

- **Job Scheduler and vManager**
  - Manages workloads across Z1/X1 platforms

- **Palladium only**
  - Dynamic Power Analysis with Joules
  - Code and functional coverage

- **Protium only: native interfaces**
  - Highest throughput
  - “breaks” congruency
A Common Methodology Makes Life Easier

- **Unified Flow**
  - Reuse of the existing Palladium environment (clocks, memory models, scripts, AVIPs, SB/EDK, etc.)
  - Congruency between emulation and prototyping
  - Going back to emulation for detailed debug

- **Common Compile Front-end**
  - No learning of new tools and flows
  - Identical language coverage
  - New capabilities and fixes available on both platform simultaneously
Accelerate SoC Performance Testing with System VIP
Why Is SoC Performance Testing a Growing Challenge?

Number and variety of processing engines is growing to address More than Moore.

SoC Infrastructure has become a mass of coherent and non-coherent interconnects stitched together with clock and domain bridges to support multiple power domains.

To maintain growing demands for SoC performance the number of cache hierarchies is growing with each generation. Combined with high-speed coherent I/O creating complex scenarios and measuring results is tough.

To keep pace with system throughput demands, DDR subsystems are becoming more complex, multiple controllers allied to complex hashing in the controllers and interconnects to ensure balanced DDR loading.
Understanding Performance Throttling

Quality of Service (QoS) settings can adjust priorities when the system becomes more highly loaded.

If the situation persists it can cause back-pressure to build all the way back to one or more Initiators.

Once a Responder cannot keep up with demand, it creates “back-pressure”

If the situation persists it can cause back-pressure to build all the way back to one or more Initiators.

Quality of Service (QoS) settings can adjust priorities when the system becomes more highly loaded.

Once a Responder cannot keep up with demand, it creates “back-pressure”

If the situation persists it can cause back-pressure to build all the way back to one or more Initiators.

Quality of Service (QoS) settings can adjust priorities when the system becomes more highly loaded.

Once a Responder cannot keep up with demand, it creates “back-pressure”

If the situation persists it can cause back-pressure to build all the way back to one or more Initiators.

Quality of Service (QoS) settings can adjust priorities when the system becomes more highly loaded.

Once a Responder cannot keep up with demand, it creates “back-pressure”

If the situation persists it can cause back-pressure to build all the way back to one or more Initiators.

Quality of Service (QoS) settings can adjust priorities when the system becomes more highly loaded.

Once a Responder cannot keep up with demand, it creates “back-pressure”

If the situation persists it can cause back-pressure to build all the way back to one or more Initiators.

Quality of Service (QoS) settings can adjust priorities when the system becomes more highly loaded.

Once a Responder cannot keep up with demand, it creates “back-pressure”

If the situation persists it can cause back-pressure to build all the way back to one or more Initiators.

Quality of Service (QoS) settings can adjust priorities when the system becomes more highly loaded.

Once a Responder cannot keep up with demand, it creates “back-pressure”

If the situation persists it can cause back-pressure to build all the way back to one or more Initiators.

Quality of Service (QoS) settings can adjust priorities when the system becomes more highly loaded.

Once a Responder cannot keep up with demand, it creates “back-pressure”

If the situation persists it can cause back-pressure to build all the way back to one or more Initiators.

Quality of Service (QoS) settings can adjust priorities when the system becomes more highly loaded.

Once a Responder cannot keep up with demand, it creates “back-pressure”

If the situation persists it can cause back-pressure to build all the way back to one or more Initiators.

Quality of Service (QoS) settings can adjust priorities when the system becomes more highly loaded.

Once a Responder cannot keep up with demand, it creates “back-pressure”

If the situation persists it can cause back-pressure to build all the way back to one or more Initiators.

Quality of Service (QoS) settings can adjust priorities when the system becomes more highly loaded.

Once a Responder cannot keep up with demand, it creates “back-pressure”

If the situation persists it can cause back-pressure to build all the way back to one or more Initiators.

Quality of Service (QoS) settings can adjust priorities when the system becomes more highly loaded.

Once a Responder cannot keep up with demand, it creates “back-pressure”

If the situation persists it can cause back-pressure to build all the way back to one or more Initiators.

Quality of Service (QoS) settings can adjust priorities when the system becomes more highly loaded.

Once a Responder cannot keep up with demand, it creates “back-pressure”

If the situation persists it can cause back-pressure to build all the way back to one or more Initiators.

Quality of Service (QoS) settings can adjust priorities when the system becomes more highly loaded.

Once a Responder cannot keep up with demand, it creates “back-pressure”

If the situation persists it can cause back-pressure to build all the way back to one or more Initiators.

Quality of Service (QoS) settings can adjust priorities when the system becomes more highly loaded.

Once a Responder cannot keep up with demand, it creates “back-pressure”

If the situation persists it can cause back-pressure to build all the way back to one or more Initiators.

Quality of Service (QoS) settings can adjust priorities when the system becomes more highly loaded.

Once a Responder cannot keep up with demand, it creates “back-pressure”

If the situation persists it can cause back-pressure to build all the way back to one or more Initiators.
A Systematic Approach to Ensuring SoC Performance

Step 1 – Characterization
- Path-by-path maxBandwidth, minLatency analysis
- Sweep Outstanding Transactions to find sweet spot

Step 2 – Synthetic Workloads
- Traffic Generators for non-coherent scenarios
- Basic Coherency Tests – cache hit rate sweeps
- Advanced Coherency Tests – complex multi master scenarios

Step 3 – Sign-off
- Define sign-off performance scenarios
- Create sign-off checks
  - Self-checking scenarios
  - Post-processing performance checks
A Systematic Approach to Ensuring SoC Performance

Simulation or Acceleration Subsystem Verification

Reuse Test Content at Subsystem and SoC

IP Verification

Subsystem Verification

SoC Verification
How Does System VIP Help?

- **Generation and Stimulus**
  - Testbench Assembly
  - SoC Test libraries

- **Cadence® System VIP**
  - System Testbench Generator
  - System Traffic Libraries

- **Analysis and Checking**
  - SoC Performance analysis
  - Data / Cache coherency checkers

- **System Performance Analyzer**

- **System Verification Scoreboard**
What Does an Automated SoC Performance Flow Look Like?

Flow

1. Generate Testbench
   - Configure CSV for Simulation
   - Generate SV UVM or C testbench

2. Scenario Creation
   - Initialization of DUT
   - Basic ATP Test
   - Basic Coherency Tests
   - Advanced Coherency Tests

3. Run Tests
   - Simulation or Acceleration

4. Analyze and Debug
   - SPA Performance Analysis
   - SVD Correlation
What Does an Automated SoC Performance Flow Look Like?

Scenario creation

Drag and drop "Actions" from the gallery which shows libraries and user code:
- Increment Bandwidth from 100MBs to 400MBs

Fields can have fixed values or in this case in a repeat loop to create 4 sequential ATP Actions

ATP FIFO Configuration models bursty IP behaviour

Fields with a "?" indicate that the Solver will generate a random value unless the user overrides with a contrained value.

400MBs is the max possible on this interface, therefore there is some minor throttling going on

FIFO creates the initial "burst" of transactions which has a bigger effect at lower bandwidths
What Does an Automated SoC Performance Flow Look Like?

DDR analysis

The DDR analysis tools help identify potential areas of concern for DDR performance.
What Does an Automated SoC Performance Flow Look Like?

Debug – Root causing problematic transactions
Renesas has used Cadence VIP for many years and values Cadence’s leadership in advanced SoC verification technologies. By adding the new System VIP to our existing verification environment based on the Cadence Xcelium and Palladium platforms, and improving stimulus re-use and automation, we’ve further accelerated the SoC verification process with 10X more efficiency, enabling us to deliver innovative, high-quality products to our customers faster.

Tetsuya Asano
Director, Design Methodology Department, Shared R&D EDA Division at Renesas Electronics Corporation

Through our collaboration with Cadence, we’ve reduced some of the complex SoC verification challenges, especially around I/O peripherals. By using Cadence System Traffic Libraries and System Performance Analyzers, Arm was able to automate complex test generation processes, enabling a quicker PCIe integration verification and performance analysis.

Tran Nguyen
Director of Design Services at Arm
Summary

- Ensuring SoC performance targets are met is a growing challenge
  - Bigger testbenches, multiple runtime engines
  - Building richer scenarios
  - Harder to analyze and debug

- Testbench automation is essential to be productive

- Building all the required scenarios is becoming a large task
  - Reuse of the scenario content is essential
  - Easy porting of content to different SoCs and execution engines is a must

- Analyzing and debugging performance requires domain-specific tools
  - On-chip bus, DDR, and other domains need special analytics
  - Tracing the transaction lifecycle across an SoC is extremely challenging without tools

System VIP addresses the challenges
Find and fix the most bugs per $ compute per day

Smart Verification Management
vManager™ – Indago™ – VIP – System VIP – Perspec™

Smartest Apps

Fastest Engines

Most Choice of Compute

Formal
JasperGold™

Simulation
Xcelium™

Emulation
Palladium™

Prototyping
Protium®

X86 or Arm® CPU

X86 or Arm CPU

Custom Processor

FPGA