Verification Futures Conference

Austin Marriott South

Thursday 14 September 2023

Sponsored by

TESSOLVE A HERO ELECTRONIX VENTURE cadence imperas

BREKER DOULOS

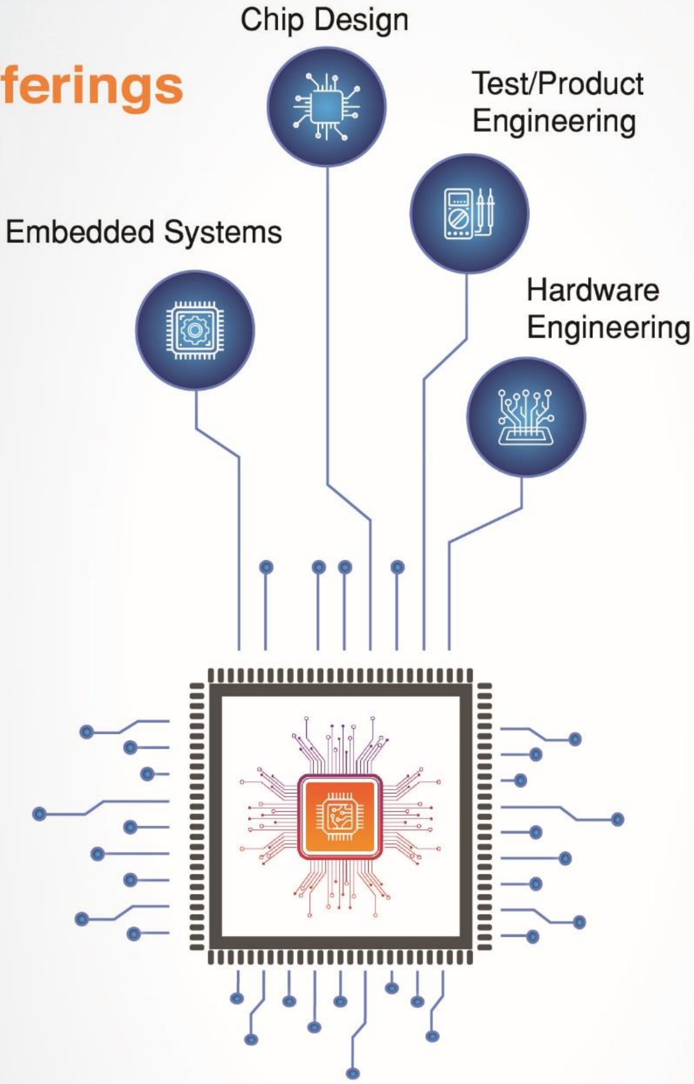
Participating Companies

ARM Bitstar BROADCOM ERICSSON intel NXP RENESAS SigmaSense TESSOLVE A HERO ELECTRONIX VENTURE SynthWorks tenstorrent

Media Sponsor

SemiWiki The open forum for semiconductor professionals

Offerings



TESSOLVE

A **HERO ELECTRONIX** VENTURE

Engineering Tomorrow's Solutions

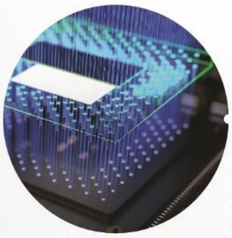
Your Capable Partner Today

Robust Infrastructure
Test Floor | Reliability & STPI Smart Lab

Expert Engineering Team
90% Technical Staff

Quality Processes
Upto 30% Cycle Time Reduction

CoE Technologies



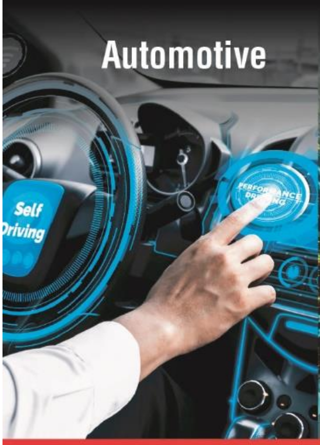
ARM, RISC-V Subsystem and Analog Block Development

5G and High Frequency RF Solutions

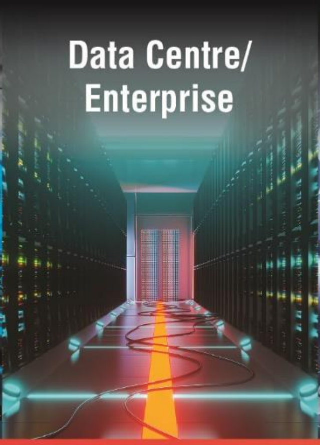
High Performance Compute Solution

Automotive Compliance Solution

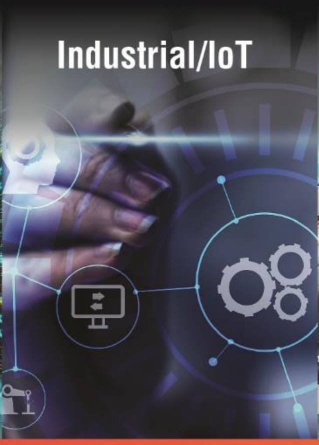
Let's Talk About Engineering New Possibilities



Automotive

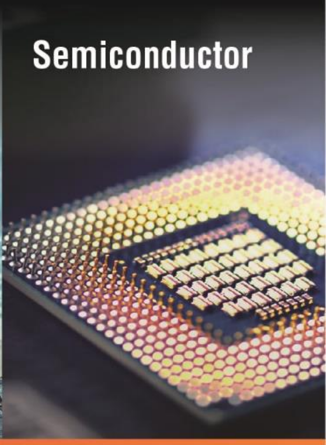


Data Centre/ Enterprise



Industrial/IoT

Avionics



Semiconductor

www.tessolve.com | Email: sales@tessolve.com

Agenda (AM)

08:30 Arrival: Breakfast and Networking

09:25 Welcome: Mike Bartley, Tessolve Semiconductor Ltd

09:30 **Keynote Speakers**
Vivek Vedula, ARM Ltd

10:15 User Top Verification Challenges

10:15 Alex Duhovich, Ericsson

10:30 Bahadir Erimli, Cadence Design Systems

11:00 Refreshments and Networking

11.30 **Multi-Track Session**

Track 1 - User Presentations on Formal Verification ***[Lonestar Ballroom – Salon A+B]***

11:30 Mike Bartley, Tessolve Semiconductor Ltd

11:40 Divyang Agrawal, Tenstorrent, Inc

12:10 Suneil Mohan, Intel Corporation

Track 2 - Training Session 1 ***[Lonestar Ballroom – Salon C]***

11:30 Doug Smith, Doulos

Track 3 - UVM for AMS Verification ***[Lonestar Ballroom – Salon D]***

11:30 Peter Grove & Steven Holloway, Renesas *Remote Presentation*

12:30 Lunch and Networking

Agenda (PM)

13:30 Larry Lapides (Imperas Software Ltd.)

14:00 Adnan Hamid (Breker Verification Systems)

14:20 Balram Naik Meghavath (Broadcom Ltd)

14:40 Hemendra Talesara (Bitstar Technologies)

15:00 Refreshments and Networking

15:30 **Multi-Track Session**

Track 1 – Latest Topics in Verification *[Lonestar Ballroom – Salon A+B]*

15:30 Aditya Devarakonda, NXP Semiconductor

15:50 Bill Tiffany, SigmaSense LLC

16:10 Benjamin Delsol, UVMGEN

Track 2 - Training Session 2 *[Lonestar Ballroom – Salon C]*

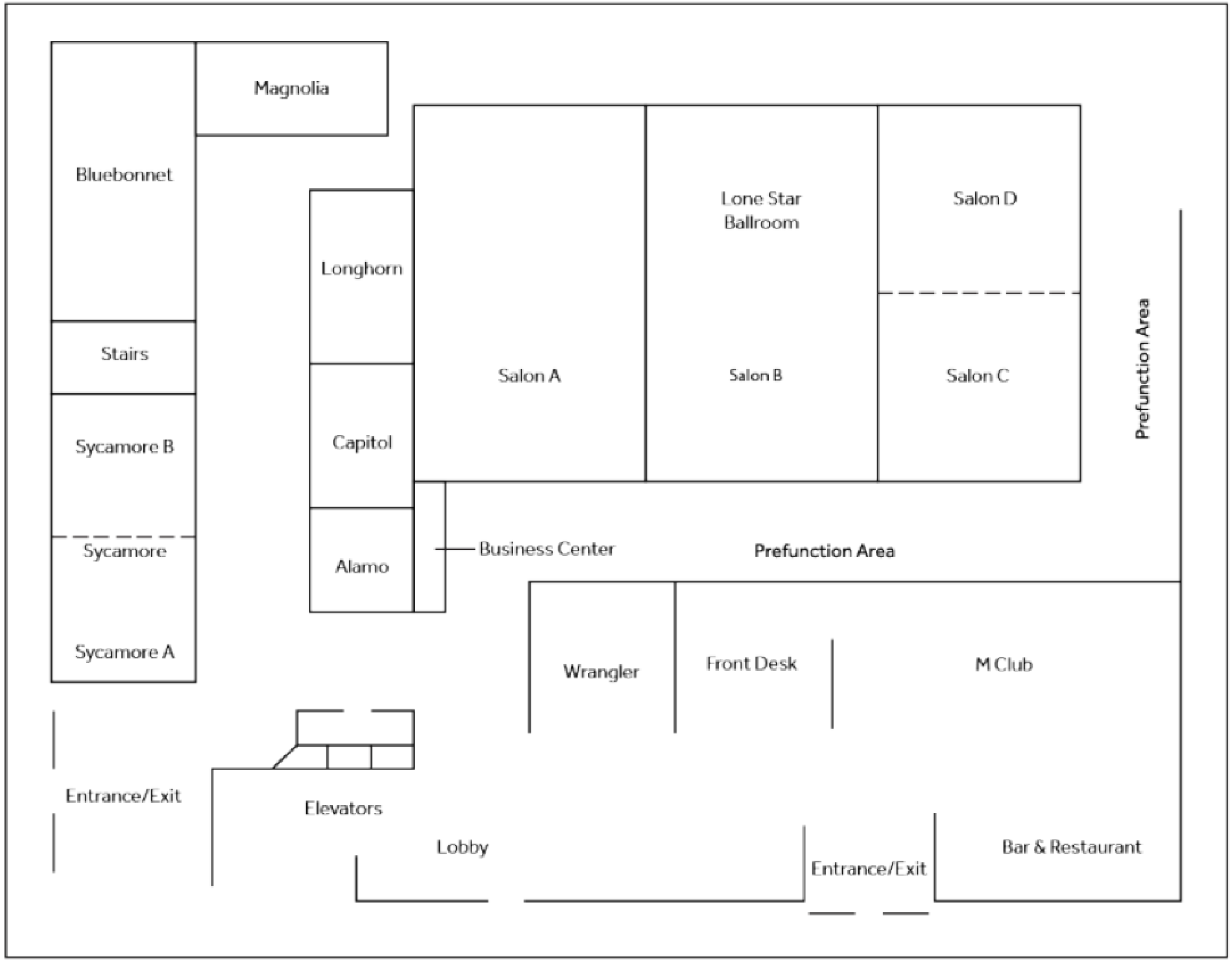
15:30 Doug Smith, Doulos

Track 3 – VHDL Verification *[Lonestar Ballroom – Salon D]*

15:30 Jim Lewis, SynthWorks Design Inc

16:30 Event Closes

Floor Plan



Magnolia
Bluebonnet
Stairs
Sycamore B
Sycamore
Sycamore A
Longhorn
Capitol
Alamo
Salon A
Lone Star
Ballroom
Salon B
Salon D
Salon C
Prefunction Area
Business Center
Prefunction Area
Wrangler
Front Desk
M Club
Entrance/Exit
Elevators
Lobby
Entrance/Exit
Bar & Restaurant

Notes

Mike Bartley

Tessolve Semiconductor Ltd

Senior Vice President – VLSI Design

Welcome Message

Biography

Dr Mike Bartley has over 30 years of experience in software testing and hardware verification. He has built and managed state-of-the-art test and verification teams inside a number of companies (including STMicroelectronics, Infineon, Panasonic and start-up ClearSpeed) and also advised a number of companies on organisational verification strategies (ARM, NXP and multiple start-ups).

Mike successfully founded and grew a software test and hardware verification services company to 450+ engineers globally, delivering services and solutions to over 50+ clients in a wide range of technologies and industries. The company was acquired by Tessolve semiconductors, a global company with 3000+ employees supporting clients in VLSI, silicon test and qualification, PCB and embedded product development in multiple vertical industries.

Mike has a PhD in Mathematics (Bristol University), and 8 MSc's in various subjects including management, software engineering, computer security robotics and AI. He is currently studying (remotely) for an MSc in Blockchain and Digital Currency at the University of Nicosia, Cypress

Tessolve would like to thank the sponsors and participants of the 2023 Verification Futures Conference

TESSOLVE
A **HERO ELECTRONIX** VENTURE

Notes

Vivek Vedula

Arm Ltd
Technical Director

Safety and Security challenges in hardware IP development

Keynote Speaker

Abstract

Ensuring the trustworthiness in computing is increasingly becoming a challenge in the inter-connected world relying on electronic systems. Security and safety provide assurance that these systems are resilient to malicious attacks and malfunctioning components, respectively. Given the diverse and rapidly evolving market demands, the requirements for both new features and performance significantly increases the probability of security- and safety-related design flaws to remain undetected. This talk will describe the challenges during IP development in efficient identification of relevant risks, and their effective mitigation for safe and secure computing.

Biography

Vivek Vedula leads the SDL methodology architecture and development for hardware IPs at Arm. Prior to this, he held several roles at Intel, NXP and Oracle Labs spanning the areas of formal verification, post-silicon validation and HW-SW co-verification. Vivek holds a PhD degree in Electrical and Computer Engineering from the University of Texas at Austin.

Slides will be shared during presentation.

Notes

Alex Duhovich

Ericsson
PEU Silicon – IP Verification Methodology Lead

Ericsson's Challenges of IP Development and Verification for Products with a Long Shelf Life

Challenge Paper

Abstract

Ericsson develops ASICs for radios which have a long shelf life and an even longer life cycle. It's hard to have IP roadmaps with on-time requirements to allow for IP-centric planning and execution. This presentation will outline some of the challenges Ericsson IP teams have been facing in their quest to IP driven in a product driven market.

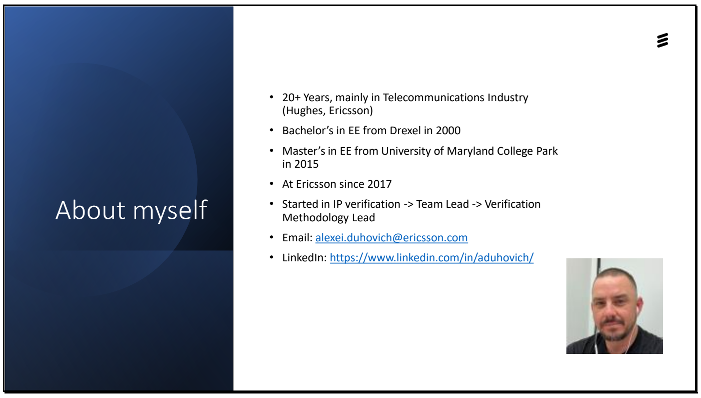
Biography

- 20+ years of ASIC/SoC design and verification experience
- BSEE from Drexel University, MEEE from University of Maryland College Park
- Most of career spent in the telecommunications industry
- Started at Ericsson in 2017
- Methodology lead since 2021

Notes

Ericsson's Challenges of IP Development and Verification for Products with a Long Shelf Life

Alex Duhovich – PEU Silicon IP Verification Methodology Lead



About myself

- 20+ Years, mainly in Telecommunications Industry (Hughes, Ericsson)
- Bachelor's in EE from Drexel in 2000
- Master's in EE from University of Maryland College Park in 2015
- At Ericsson since 2017
- Started in IP verification -> Team Lead -> Verification Methodology Lead
- Email: alexei.duhovich@ericsson.com
- LinkedIn: https://www.linkedin.com/in/aduhovich/

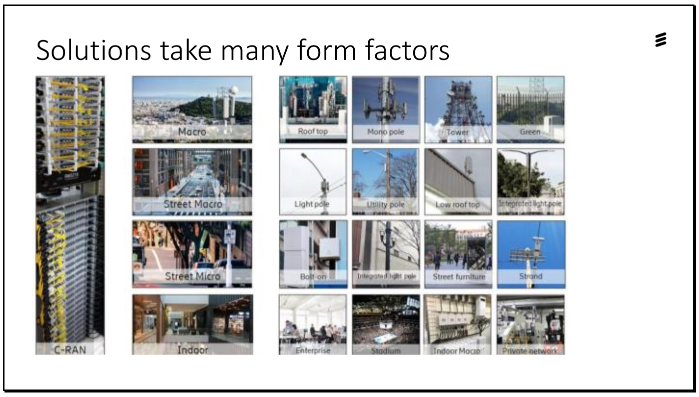
Ericsson at a glance:
A world leader in ICT and 5G

Purpose:
- To create connections that make the unimaginable possible

Vision:
- A world where limitless connectivity improves lives, redefines business, and pioneers a sustainable future

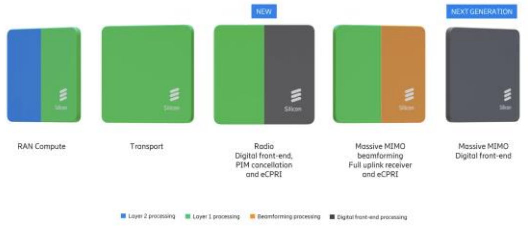
History:
- 140+ years of delivering ground-breaking solutions and innovative technology for good

Leader in Technology:
- Leading provider of Information and Communication Technology (ICT) to service providers
- 227.2 b. SEK (~ $ 27b) in Sales
- 54,000 patents



Solutions take many form factors

Ericsson Silicon Portfolio



Challenges

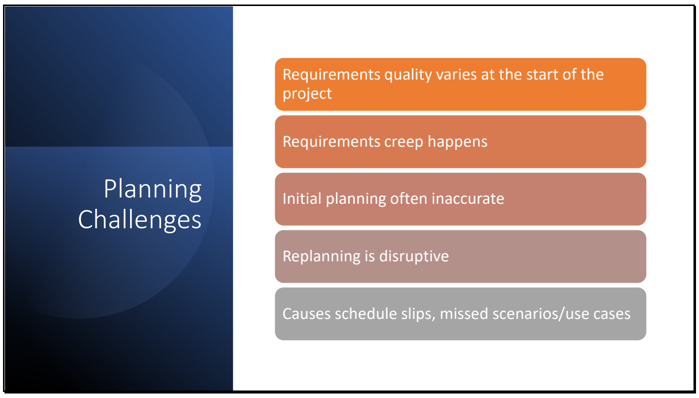
- Product-centric development
- Long product shelf lives lead to requirements creep
- Requirement quality gaps lead to planning challenges and schedule slips
- General increasing complexity challenge with verification: Methodology scaling, Power, Security

Product Centric Development

- IP requirements based on product
- Requirements come in at the start of a project
- Conflicting requirements possible between projects
- Team to close all milestones for each project
- Difficult to maintain code base and version control
- Difficult to deliver to multiple projects at once

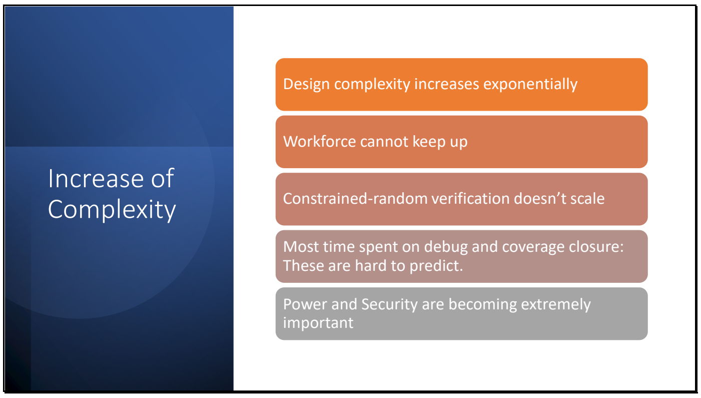
Long Product Shelf Lives

- Product lives of ~10 years
- Customer expects longevity
- Products are overdesigned to support future standards
- Cannot iterate on fixes between generations. Must be right the first time.



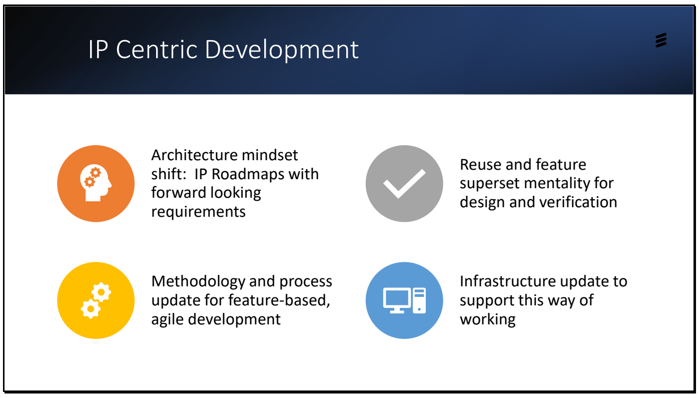
Planning Challenges

- Requirements quality varies at the start of the project
- Requirements creep happens
- Initial planning often inaccurate
- Replanning is disruptive
- Causes schedule slips, missed scenarios/use cases



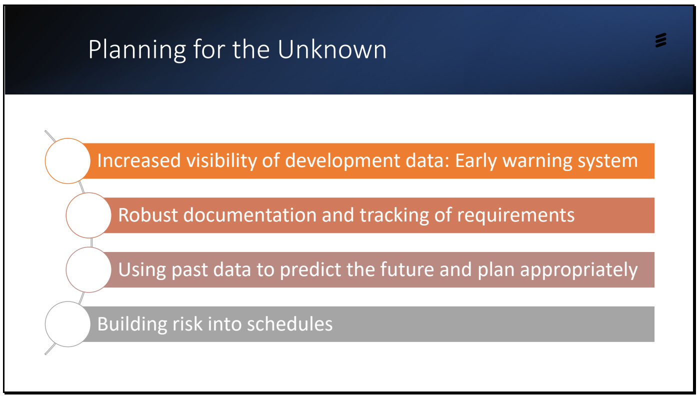
Increase of Complexity

- Design complexity increases exponentially
- Workforce cannot keep up
- Constrained-random verification doesn't scale
- Most time spent on debug and coverage closure: These are hard to predict.
- Power and Security are becoming extremely important



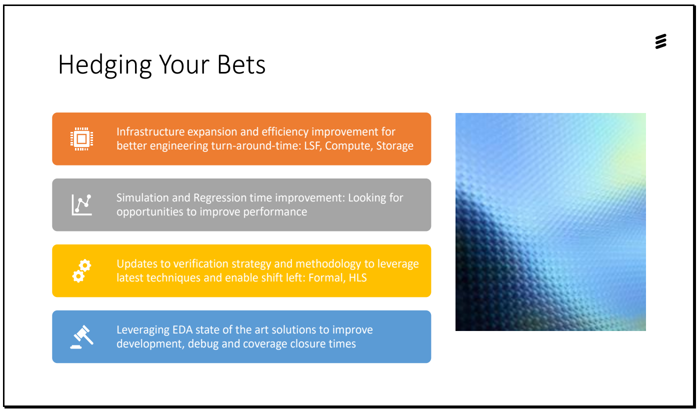
IP Centric Development

- Architecture mindset shift: IP Roadmaps with forward looking requirements
- Reuse and feature superset mentality for design and verification
- Methodology and process update for feature-based, agile development
- Infrastructure update to support this way of working



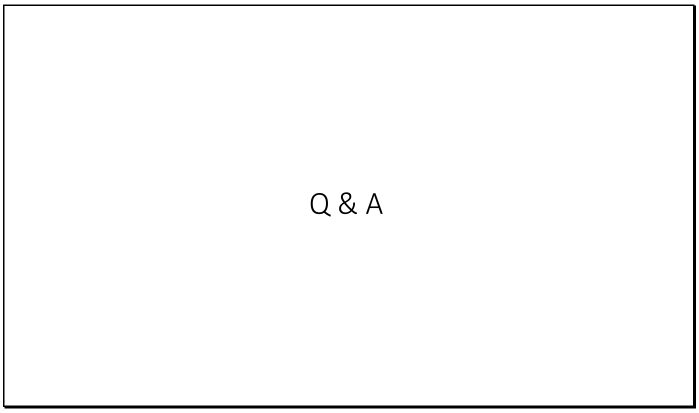
Planning for the Unknown

- Increased visibility of development data: Early warning system
- Robust documentation and tracking of requirements
- Using past data to predict the future and plan appropriately
- Building risk into schedules



Hedging Your Bets

- Infrastructure expansion and efficiency improvement for better engineering turn-around-time: LSF, Compute, Storage
- Simulation and Regression time improvement: Looking for opportunities to improve performance
- Updates to verification strategy and methodology to leverage latest techniques and enable shift left: Formal, HLS
- Leveraging EDA state of the art solutions to improve development, debug and coverage closure times



Q & A

Bahadir Erimli

Cadence Design Systems

Group Director – Verification Application Engineering

Engines, Logistics and AI

Platinum Sponsor

Abstract

As the verification problem continues to grow, the key metric that many verification teams must closely consider is “Total Verification Throughput.” While verification engines like simulation, formal, emulation and so on have a key part to play in total verification throughput, additional concepts like verification logistics and the utilization of AI can have significant impact and potentially benefit as well. This presentation will introduce the concept of verification logistics and how AI is, and will be, applied.

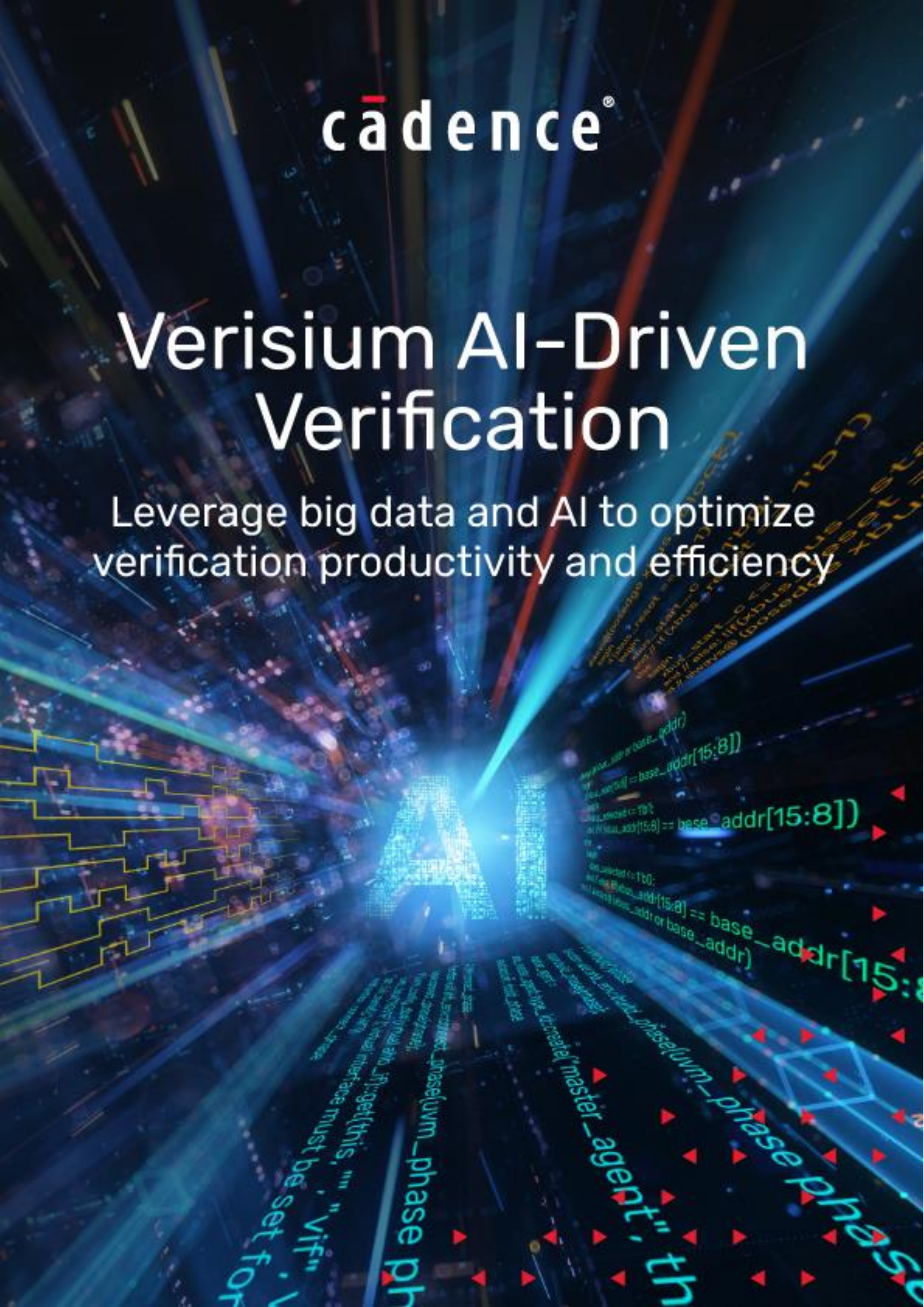
Biography

Bahadir Erimli is a member of the Cadence Worldwide Field Operations team where, as a Group Director he leads the Verification Applications Engineering team primarily in California including Silicon Valley. Before joining Cadence nearly 12 years ago, Bahadir held a number of senior engineering positions at consume and biotech semiconductor companies. Bahadir is based in San Diego, and holds a Bachelor’s degree in Electrical Engineer from Middle East Technical University in Turkey, as well as advanced degrees in electrical engineer from Caltech.

cadence®

Verisium AI-Driven Verification

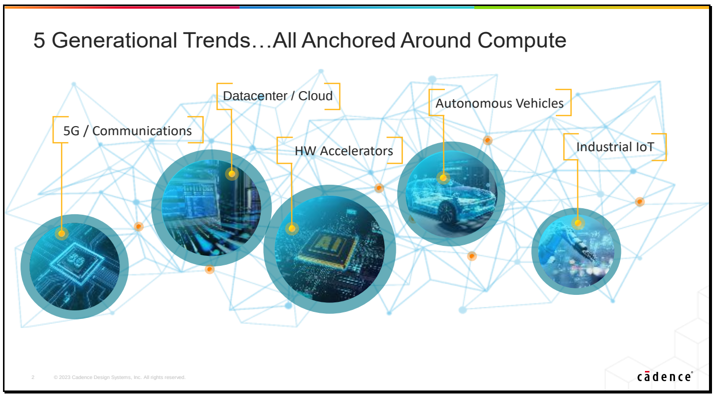
Leverage big data and AI to optimize verification productivity and efficiency



Engines, Logistics, and AI

Verification Futures Austin

Bahadir Erimli
Group Director, Verification Application Engineering



5 Generational Trends…All Anchored Around Compute

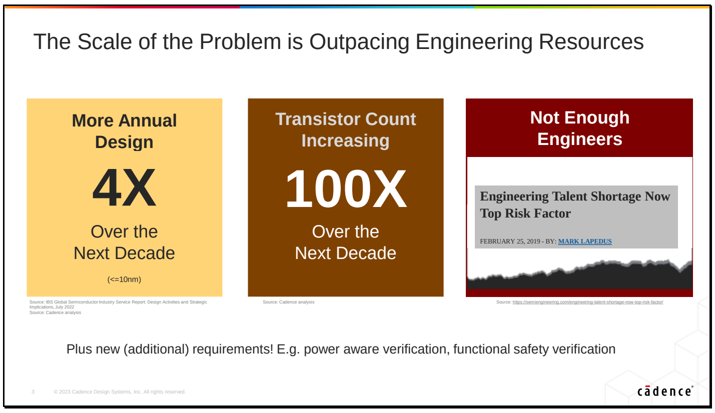
5G / Communications

Datacenter / Cloud

HW Accelerators

Autonomous Vehicles

Industrial IoT



The Scale of the Problem is Outpacing Engineering Resources

More Annual Design

4X

Over the Next Decade

(<=10nm)

Source: IBS Global Semiconductor Industry Service Report: Design Activities and Strategic Implications, July 2022
Source: Cadence analysis

Transistor Count Increasing

100X

Over the Next Decade

Source: Cadence analysis

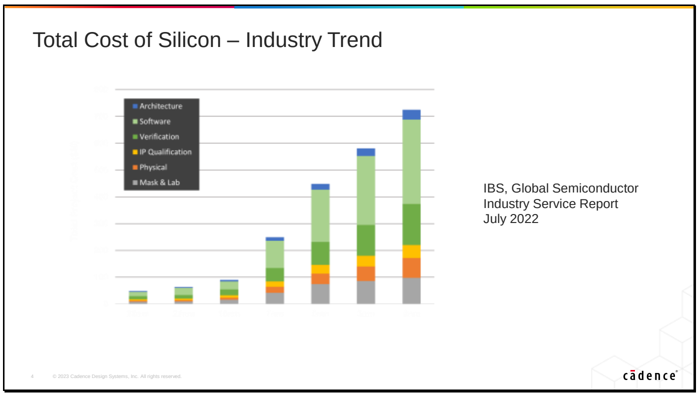
Not Enough Engineers

Engineering Talent Shortage Now Top Risk Factor

FEBRUARY 25, 2019 - BY: MARK LAPEDUS

Source: https://semiengineering.com/engineering-talent-shortage-now-top-risk-factor/

Plus new (additional) requirements! E.g. power aware verification, functional safety verification

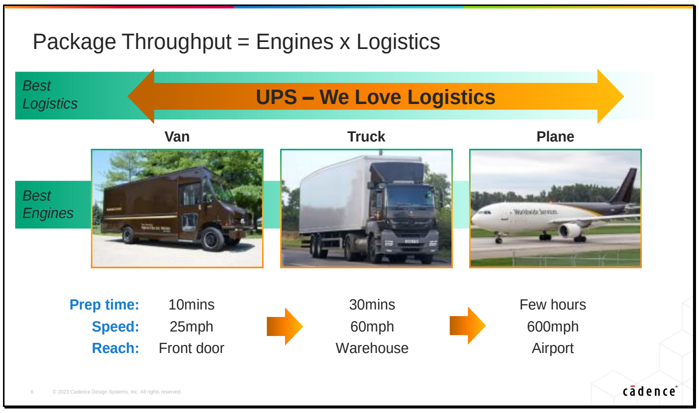


Total Cost of Silicon – Industry Trend

IBS, Global Semiconductor Industry Service Report July 2022

> “Amateurs talk about strategy…
> …professionals talk about logistics.”
>
> *(most commonly attributed to)*
> ***Omar Bradley***
> *General, United States Army*

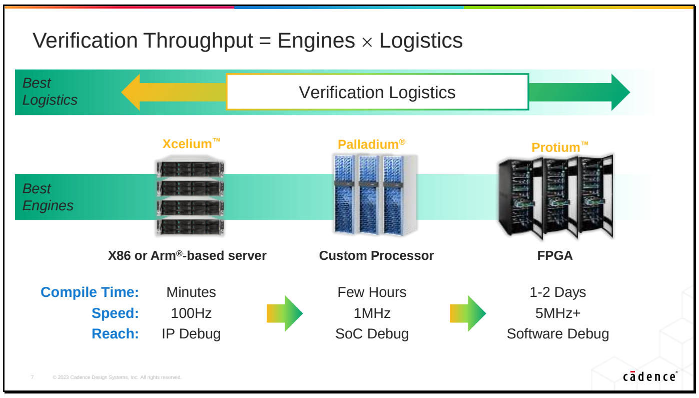
cādence



Package Throughput = Engines x Logistics

	Van	Truck	Plane
Prep time:	10mins	30mins	Few hours
Speed:	25mph	60mph	600mph
Reach:	Front door	Warehouse	Airport

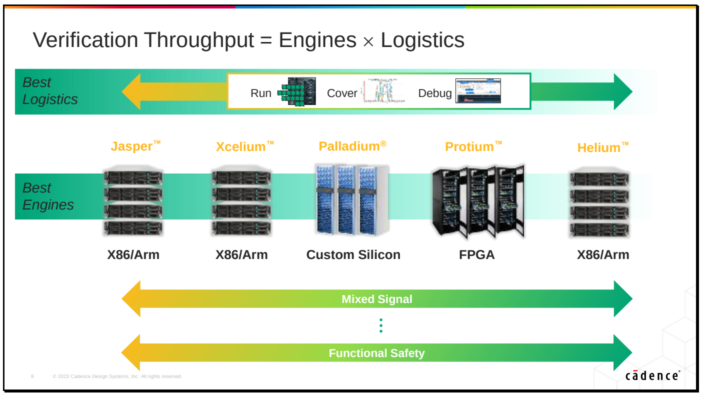
cādence



Verification Throughput = Engines × Logistics

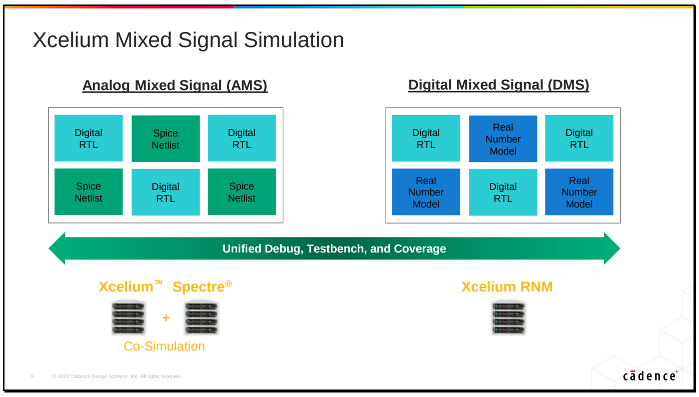
	X86 or Arm®-based server	Custom Processor	FPGA
Compile Time:	Minutes	Few Hours	1-2 Days
Speed:	100Hz	1MHz	5MHz+
Reach:	IP Debug	SoC Debug	Software Debug

cādence



Verification Throughput = Engines × Logistics

cādence



Xcelium Mixed Signal Simulation

Analog Mixed Signal (AMS)

Digital RTL	Spice Netlist	Digital RTL
Spice Netlist	Digital RTL	Spice Netlist

Digital Mixed Signal (DMS)

Digital RTL	Real Number Model	Digital RTL
Real Number Model	Digital RTL	Real Number Model

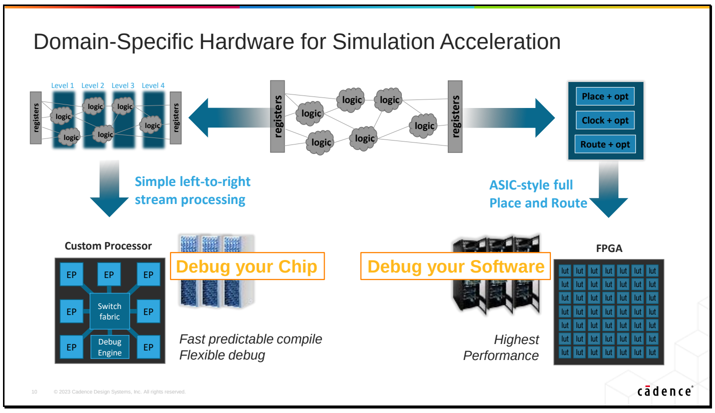
Unified Debug, Testbench, and Coverage

Xcelium™ Spectre®

+

Co-Simulation

Xcelium RNM



Domain-Specific Hardware for Simulation Acceleration

Simple left-to-right stream processing

ASIC-style full Place and Route

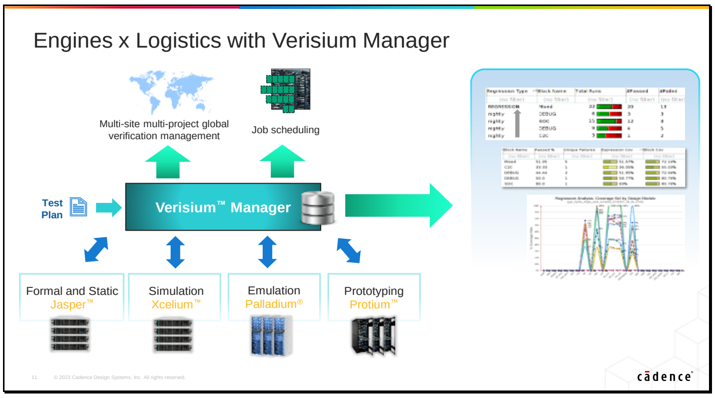
Custom Processor

Debug your Chip

Debug your Software

Fast predictable compile
Flexible debug

Highest Performance



Engines x Logistics with Verisium Manager

Multi-site multi-project global verification management

Job scheduling

Test Plan

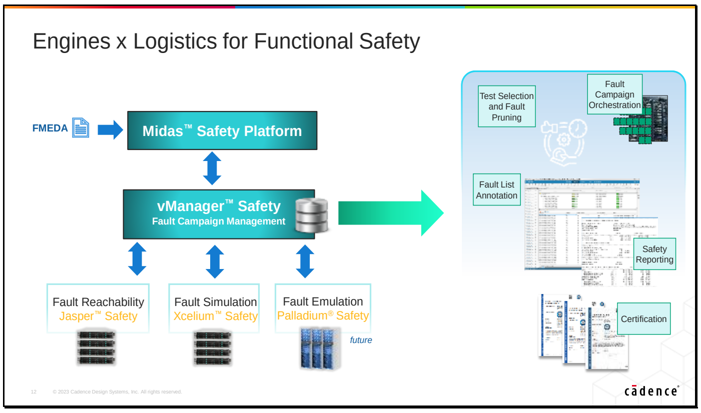
Verisium™ Manager

Formal and Static
Jasper™

Simulation
Xcelium™

Emulation
Palladium®

Prototyping
Protium™



Engines x Logistics for Functional Safety

FMEDA

Midas™ Safety Platform

vManager™ Safety
Fault Campaign Management

Fault Reachability
Jasper™ Safety

Fault Simulation
Xcelium™ Safety

Fault Emulation
Palladium® Safety

future

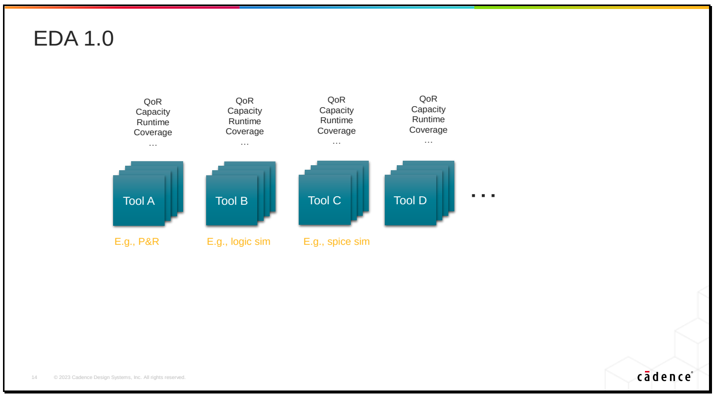
Test Selection and Fault Pruning

Fault Campaign Orchestration

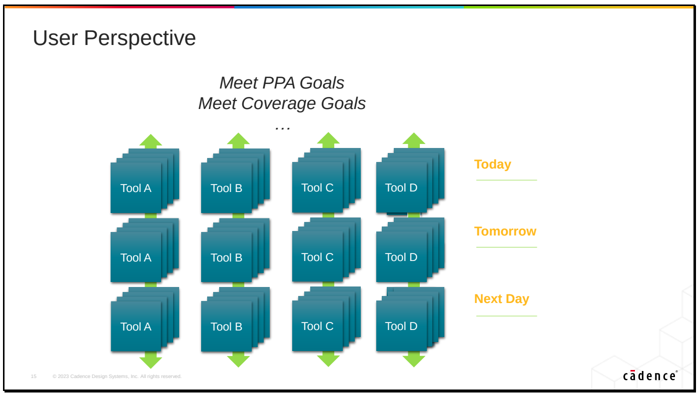
Fault List Annotation

Safety Reporting

Certification

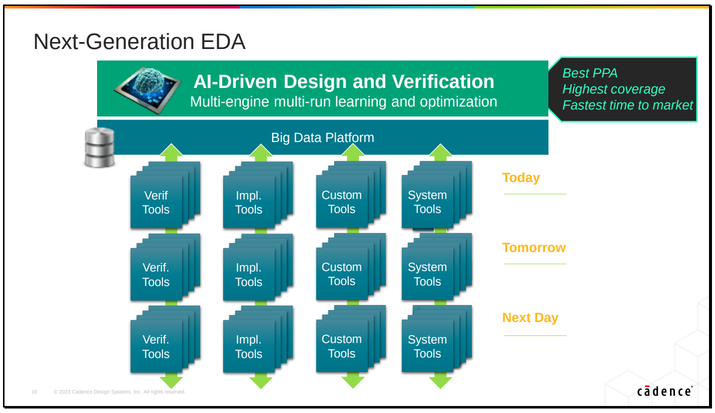


EDA 1.0

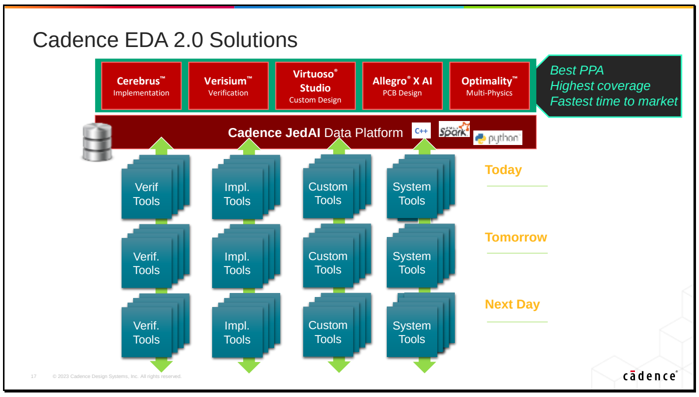


User Perspective

Meet PPA Goals
Meet Coverage Goals
…

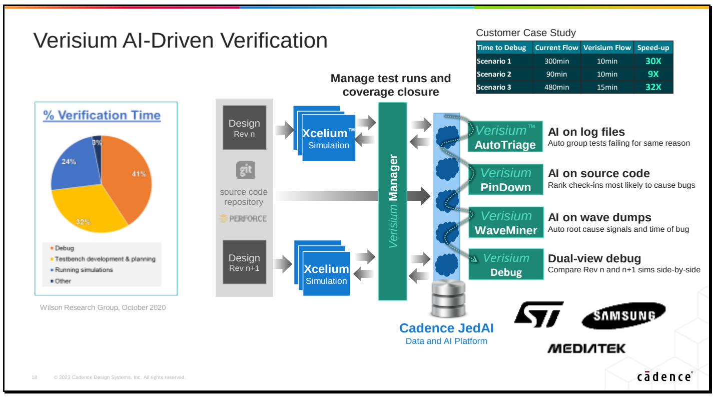


Next-Generation EDA



Cadence EDA 2.0 Solutions

Best PPA
Highest coverage
Fastest time to market



Verisium AI-Driven Verification

Customer Case Study

Time to Debug	Current Flow	Verisium Flow	Speed-up
Scenario 1	300min	10min	30X
Scenario 2	90min	10min	9X
Scenario 3	480min	15min	32X

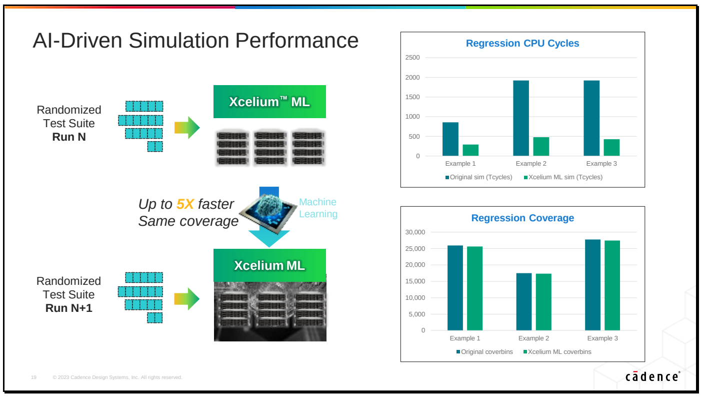
Wilson Research Group, October 2020

AI on log files
Auto group tests failing for same reason

AI on source code
Rank check-ins most likely to cause bugs

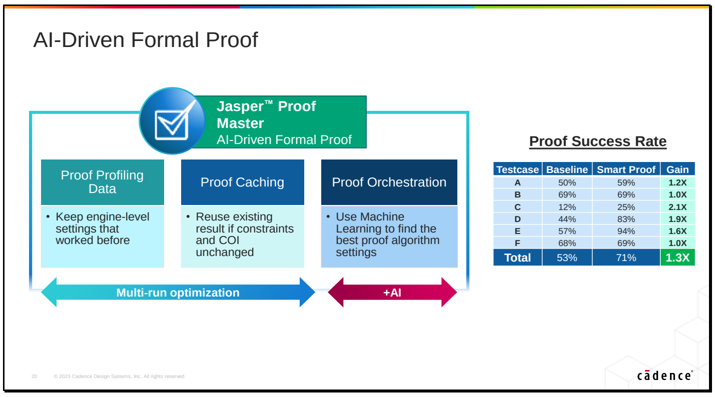
AI on wave dumps
Auto root cause signals and time of bug

Dual-view debug
Compare Rev n and n+1 sims side-by-side



AI-Driven Simulation Performance

Up to 5X faster
Same coverage



AI-Driven Formal Proof

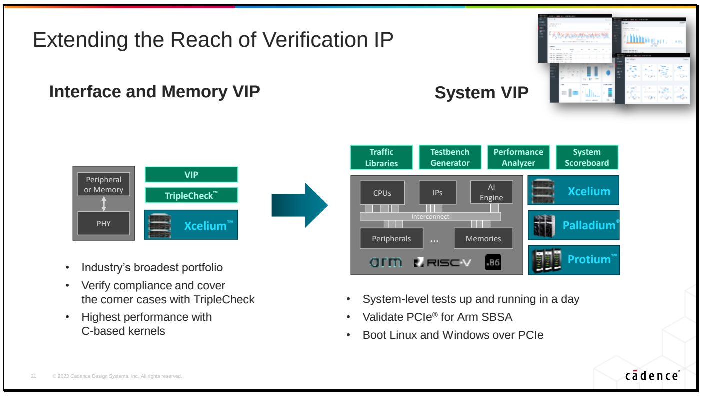
Proof Profiling Data
- Keep engine-level settings that worked before

Proof Caching
- Reuse existing result if constraints and COI unchanged

Proof Orchestration
- Use Machine Learning to find the best proof algorithm settings

Proof Success Rate

Testcase	Baseline	Smart Proof	Gain
A	50%	59%	1.2X
B	69%	69%	1.0X
C	12%	25%	2.1X
D	44%	83%	1.9X
E	57%	94%	1.6X
F	68%	69%	1.0X
Total	53%	71%	1.3X



Extending the Reach of Verification IP

Interface and Memory VIP

Peripheral or Memory

PHY

VIP

TripleCheck™

Xcelium™

- Industry’s broadest portfolio
- Verify compliance and cover the corner cases with TripleCheck
- Highest performance with C-based kernels

System VIP

Traffic Libraries

Testbench Generator

Performance Analyzer

System Scoreboard

CPUs

IPs

AI Engine

Interconnect

Peripherals

...

Memories

arm

RISC-V

.86

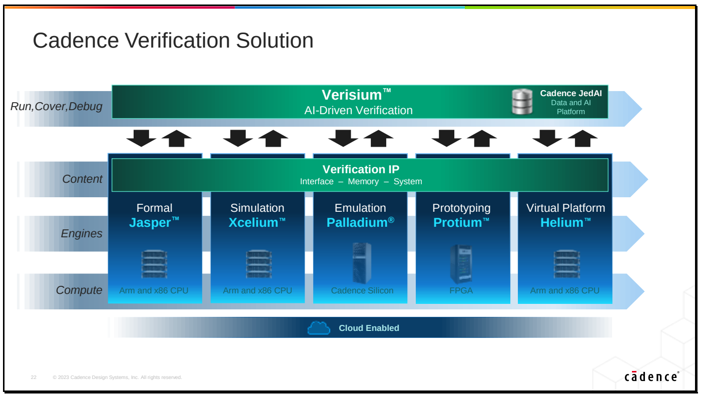
Xcelium

Palladium®

Protium™

- System-level tests up and running in a day
- Validate PCIe® for Arm SBSA
- Boot Linux and Windows over PCIe

cādence®



Cadence Verification Solution

Run,Cover,Debug

Verisium™
AI-Driven Verification

Cadence JedAI
Data and AI Platform

Content

Verification IP
Interface – Memory – System

Engines

Compute

Formal
Jasper™
Arm and x86 CPU

Simulation
Xcelium™
Arm and x86 CPU

Emulation
Palladium®
Cadence Silicon

Prototyping
Protium™
FPGA

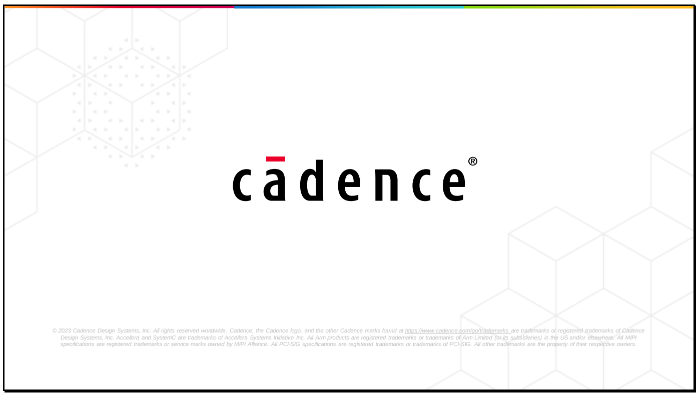
Virtual Platform
Helium™
Arm and x86 CPU

Cloud Enabled

cādence®

Thank You

cādence®



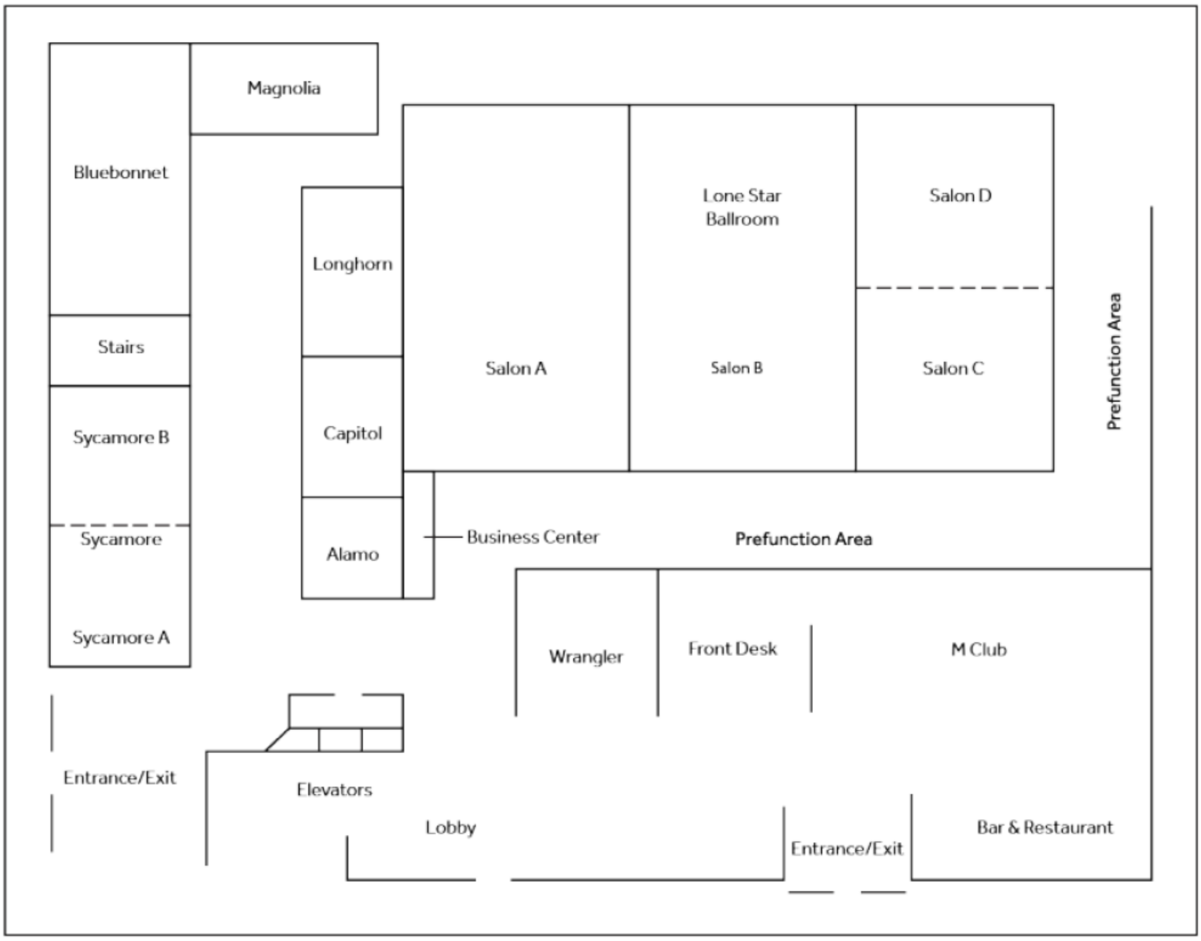
cādence®

Track Session

User Presentations

Lonestar Ballroom – Salon A+B

FLOOR PLAN



We would be grateful if you could move to the track session as quickly as possible.

Notes

Mike Bartley

Tessolve Semiconductor Limited
Senior Vice President - VLSI

10 years of Verification Challenges

User Paper

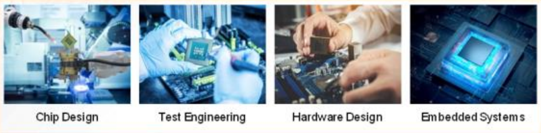
Abstract
Verification Futures has been running for more than 10 years and in that time more than 25 verification managers have given their views on their main challenges in verification. This talk will summarise those challenges and the main solutions organisations have put in place.

Notes

TESSOLVE
A HERO ELECTRONIX VENTURE

Verification Futures Austin 2023

12 years of challenges = 100+ challenges from 30+ experts



Historic summary

TESSOLVE
A HERO ELECTRONIX VENTURE

Challenge	Mentions
Complexity	12
Debug	12
Resources	12
Integrating Methods, Languages and Tools	10
Completeness	10
Safety verification	9
Scalability	8
Mixed Signal	7
Power Verification	6
Productivity	5
Reuse	5
HW/ SW	5
Security verification	5
Using AI/ML in verification	4

⇔ Non-mover Design complexity stabilising

⇔ Non-mover 70% projects use UVM
All other methods <10%

⇧ On the rise 44% of IC/ ASIC projects are safety-critical

⇧ On the rise 58% of IC/ ASIC projects add security features to their designs

⇦ New entry

2022 Wilson Research Group IC/ ASIC functional verification trends

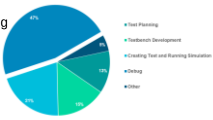
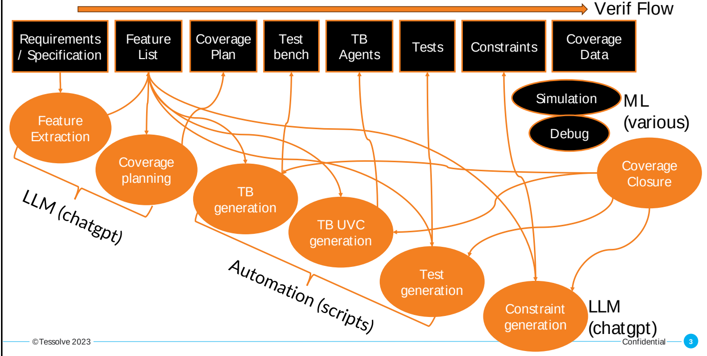


Fig. 9. Where IC/ASIC verification engineers spend their time.

©Tessolve 2023 Confidential

AI/ ML Experiments in IP Verification

TESSOLVE
A HERO ELECTRONIX VENTURE



©Tessolve 2023 Confidential

Notes

Divyang Agrawal

Tenstorrent, Inc

Sr. Director, RISCV Cores

RISCV CPU Verification - Opportunities and Challenges

User Paper

Abstract

The highly configurable nature of RISCV ISA makes it uniquely suited for a hierarchical verification methodology covering both architectural and microarchitectural complexity. This technical talk will focus on how Tenstorrent leveraged on the lessons from x86 and ARM to build a modular and scalable CPU verification framework. It will also preview how design complexity has to be tackled looking at silicon as a starting point. And ultimately why robust open source RISCV verification collateral is essential for broader adoption of the ISA from microcontrollers to high performance datacenter class products

Biography

Divyang Agrawal is a Senior Director at Tenstorrent where he works on RISCV Cores focusing on design verification, emulation, architectural tools and methodologies. He has previously worked on x86 and ARM architectures. Prior to Tenstorrent, Divyang worked at AMD where he held leadership roles within AMD's CPU Cores team working on several generations of high-performance cores. He also led the CPU Power Management IPs and Silicon Validation for all AMD cores. Divyang has a BTech in EE from Nagpur, India and an MBA from University of California at Berkeley

Slides will be shared during presentation

tenstorrent

Notes

Suneil Mohan

Intel Corporation
SOC Design Engineer

Validation of Hybrid Architectures

User Paper

Abstract
Intel’s 12th generation processors (code named Alderlake) introduced a new asymmetrical design that combines a mix of Performance Cores (P-cores) and Efficient Cores (E-cores), delivering scalable, efficient, multi-threaded performance in a single package. The validation challenge for this asymmetrical design spanned both Pre Silicon and Post Silicon phases. To meet the challenge of validating thoroughly the new asymmetrical design, our validation methodology had to be overhauled; this ranged from updating existing test generators all the way to developing new testing methodologies. In this presentation, we will cover key aspects of our asymmetrical design validation methodology in both Pre and Post Silicon phases, the strategies we adopted and the challenges that we had to overcome.

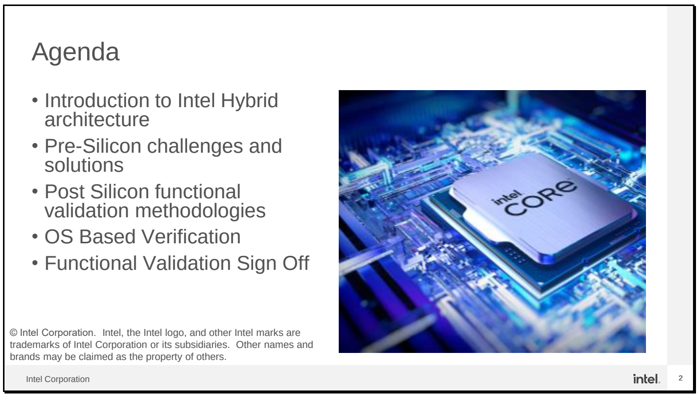
Biography
Dr. Suneil Mohan received his BE from Anna University in India in 2006 and PhD from Texas A&M University in 2012. He is a senior validation engineer in the Intel E-core team with deep expertise in both Emulation and Post silicon validation. He is currently the Post Silicon debug lead for the E-core team. He has worked on multiple generations of the E-core product line including those that are part of the most recent 13th Generation Intel® Core™ processors. In addition, he has experience working on the ISO26262 standard.

Notes

Verification Futures Conference 2023 US

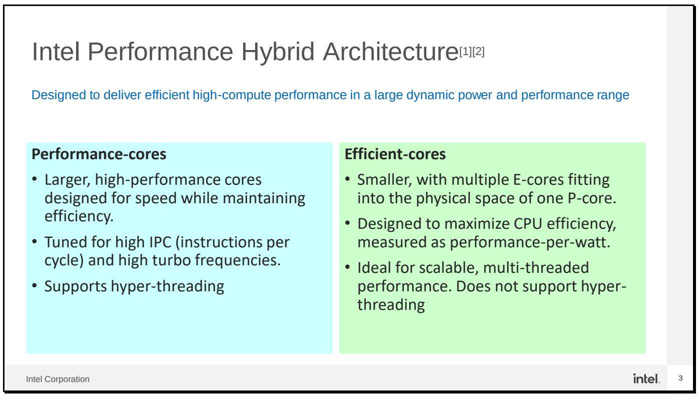
Hybrid Architecture Validation

Suneil Mohan



Agenda

- Introduction to Intel Hybrid architecture
- Pre-Silicon challenges and solutions
- Post Silicon functional validation methodologies
- OS Based Verification
- Functional Validation Sign Off



Intel Performance Hybrid Architecture[1][2]

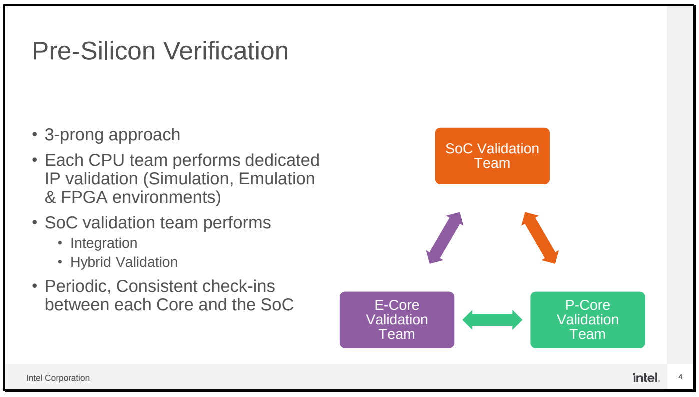
Designed to deliver efficient high-compute performance in a large dynamic power and performance range

Performance-cores

- Larger, high-performance cores designed for speed while maintaining efficiency.
- Tuned for high IPC (instructions per cycle) and high turbo frequencies.
- Supports hyper-threading

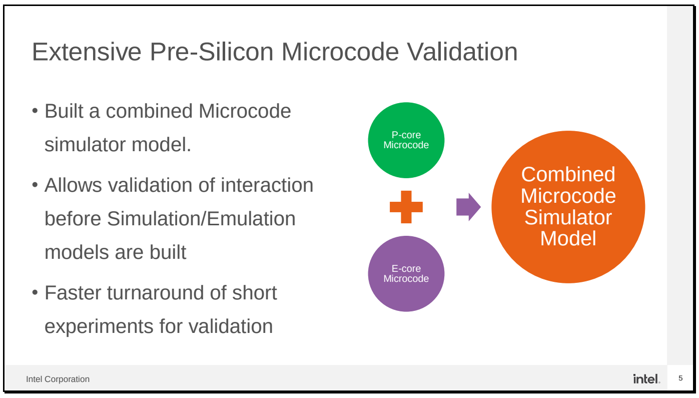
Efficient-cores

- Smaller, with multiple E-cores fitting into the physical space of one P-core.
- Designed to maximize CPU efficiency, measured as performance-per-watt.
- Ideal for scalable, multi-threaded performance. Does not support hyper-threading



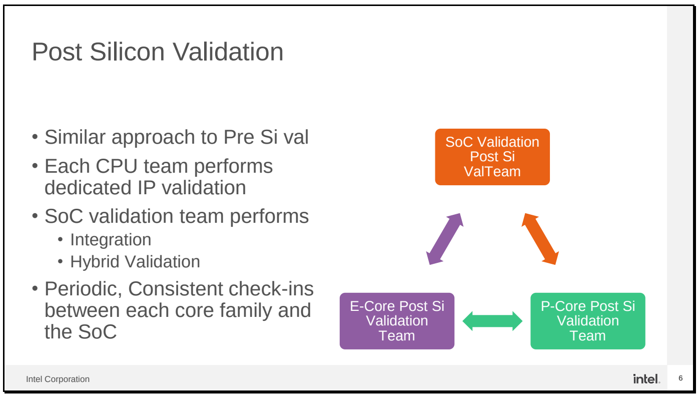
Pre-Silicon Verification

- 3-prong approach
- Each CPU team performs dedicated IP validation (Simulation, Emulation & FPGA environments)
- SoC validation team performs
  - Integration
  - Hybrid Validation
- Periodic, Consistent check-ins between each Core and the SoC



Extensive Pre-Silicon Microcode Validation

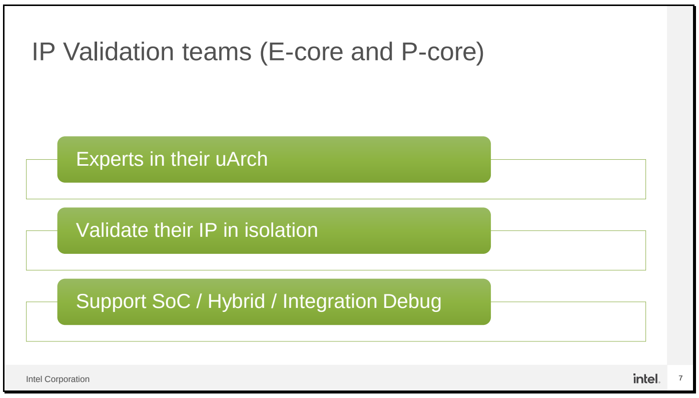
- Built a combined Microcode simulator model.
- Allows validation of interaction before Simulation/Emulation models are built
- Faster turnaround of short experiments for validation



Post Silicon Validation

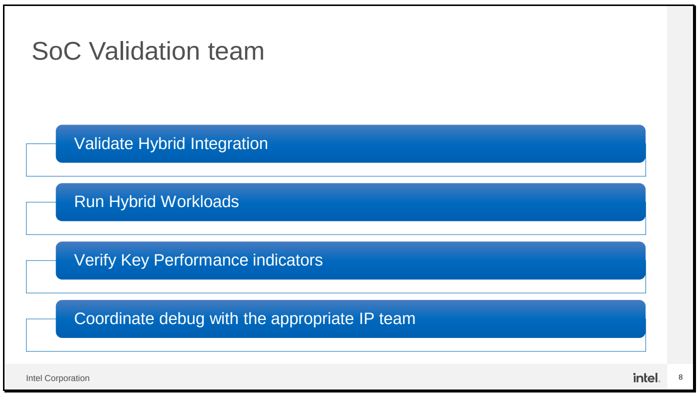
- Similar approach to Pre Si val
- Each CPU team performs dedicated IP validation
- SoC validation team performs
  - Integration
  - Hybrid Validation
- Periodic, Consistent check-ins between each core family and the SoC

SoC Validation Post Si ValTeam
E-Core Post Si Validation Team
P-Core Post Si Validation Team



IP Validation teams (E-core and P-core)

- Experts in their uArch
- Validate their IP in isolation
- Support SoC / Hybrid / Integration Debug



SoC Validation team

- Validate Hybrid Integration
- Run Hybrid Workloads
- Verify Key Performance indicators
- Coordinate debug with the appropriate IP team

OS Based Verification

What if Synthetic Workloads are not stressful enough?

- External Software may have corner case behaviors that are not always modelled.
- External Software may do things that are not completely to 'spec'
- Need to see how random task switching might behave

1. Analyzed Open-Source OS scheduler source code for task switching
2. Ran Task Switching OS Scheduler code on the combined microcode simulator model to understand the behavior
3. Built a randomized OS task switcher for Post Silicon validation

Functional Validation Sign off

Pre-Silicon

- Pass rates for synthetic test content.
- All failures accounted for, understood and dispositioned
- Coverage data analysis complete.
- Power and Perf data collected, analyzed and within expected ranges.

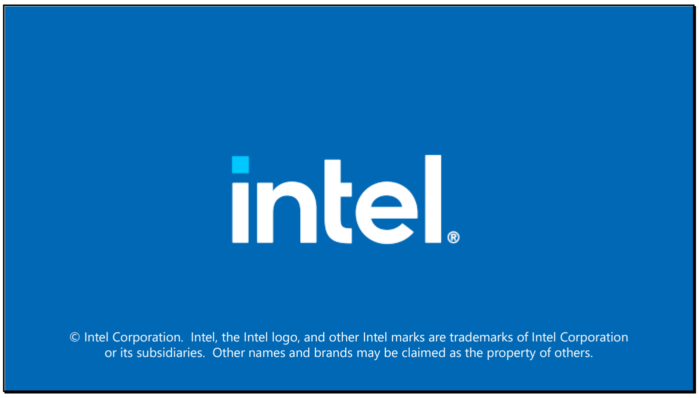
Post Silicon

- Acceptable pass rates for synthetic test content
- All failures accounted for, understood and dispositioned
- Power and Performance data meeting projections
- 100% pass rate for the OS task switching tests.
- Thread Director working as expected

References and Additional Resources

[1] E. Rotem et al., "Intel Alder Lake CPU Architectures," in IEEE Micro, vol. 42, no. 3, pp. 13-19, 1 May-June 2022, doi: 10.1109/MM.2022.3164338.	
[2] How 13th Gen Intel® Core™ Processors Work: https://www.intel.com/content/www/us/en/gaming/resources/how-hybrid-design-works.html	
Lakefield	S. Khushu and W. Gomes, "Lakefield: Hybrid cores in 3D Package," 2019 IEEE Hot Chips 31 Symposium (HCS), Cupertino, CA, USA, 2019, pp. 1-20, doi: 10.1109/HOTCHIPS.2019.8875641
Meteor Lake	W. Gomes, S. Morgan, B. Phelps, T. Wilson and E. Hallnor, "Meteor Lake and Arrow Lake Intel Next-Gen 3D Client Architecture Platform with Foveros," 2022 IEEE Hot Chips 34 Symposium (HCS), Cupertino, CA, USA, 2022, pp. 1-40, doi: 10.1109/HCS55958.2022.9895532
Intel Validation Lab	"I've never had so much fun - intel development center deep dive," YouTube, 25-May-2022. [Online]. Available: https://youtu.be/BtFdraQWVtM [Accessed: 16-Jul-2023]
Intel 12th Gen Validation Platform	"I saved the best for Last - Intel Design Center Development Motherboard," YouTube, 06-Jul-2022. [Online]. Available: https://youtu.be/pyVZ05SO0Ic [Accessed: 16-Jul-2023]

Q & A

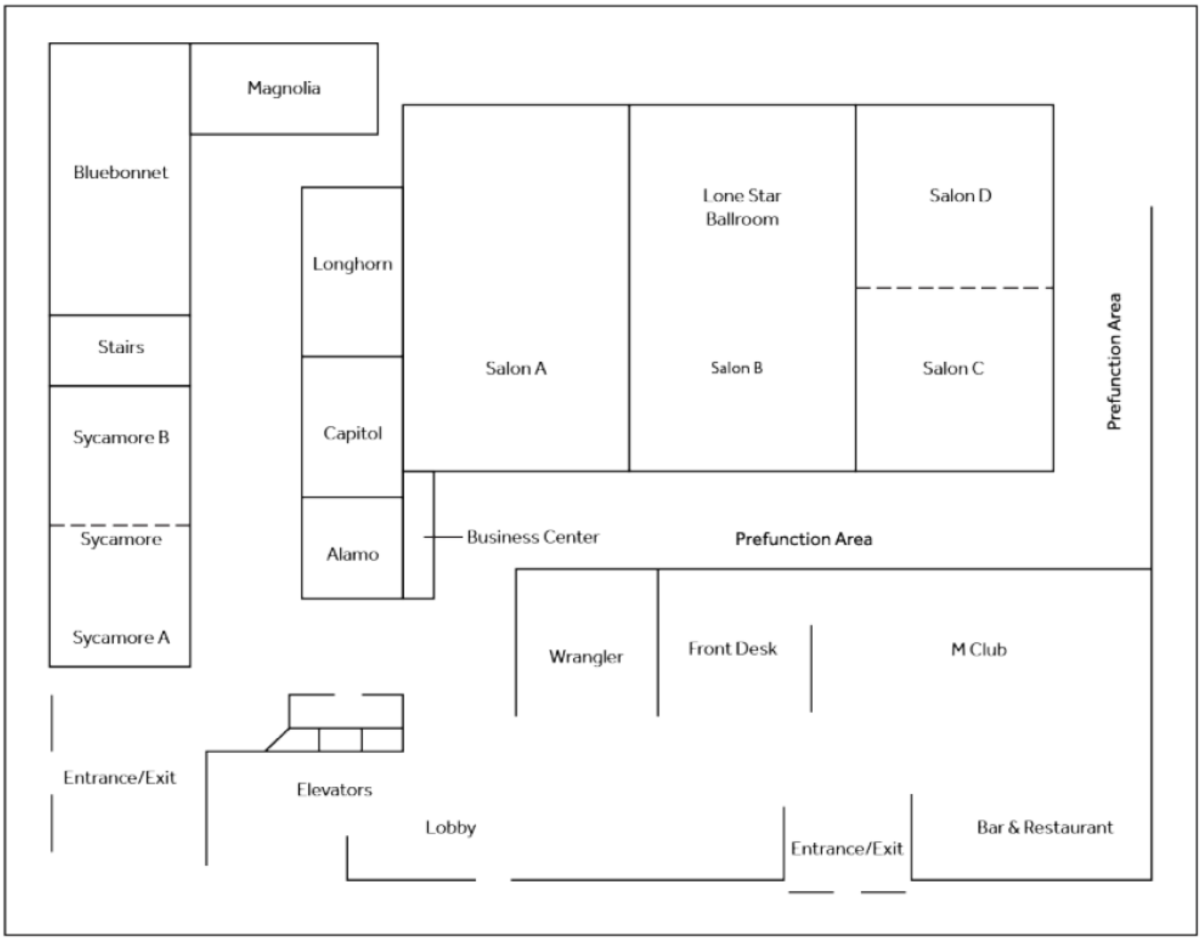


Track Session

Training Session - 1

Lonestar Ballroom – Salon C

FLOOR PLAN



We would be grateful if you could move to the track session as quickly as possible.

Notes

Doug Smith

Doulos

Engineer / Instructor

What Can Formal Do For Me?

Gold Sponsor

Abstract

We know formal can prove things, but where do we apply it? Did you know you can use formal to generate simulation testbenches for covering coverage holes or have it visualize your design without writing a single line of testbench code? Formal can be used for identifying metastability, X propagation, fault propagation and detection, equivalence, and so much more. In this tutorial session, we'll have a look at the many ways formal helps out your design verification process.

Biography

Doug Smith is a verification engineer and instructor for Doulos based in the Austin Texas area with expertise in UVM and formal technologies. He has been using formal technology for several decades, performing formal verification on many kinds of designs and formal applications. Likewise, he has provided formal application support at both Jasper and Mentor/Siemens EDA. At Mentor/Siemens EDA, he served as a formal specialist and verification consultant, where he provided both formal consulting and developed an automotive functional safety formal app for performing formal fault campaigns. At Doulos, he delivers training in verification methodologies like UVM, SystemVerilog, and formal technology.

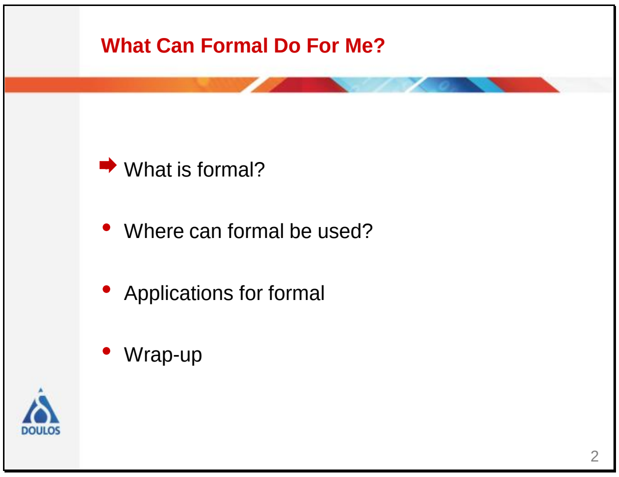
Doug holds a masters degree in Computer Engineering from the University of Cincinnati and a bachelors degree in Physics and Biology from Northern Kentucky University. Currently, he resides in Paige Texas with his wife and family on a small farm where he raises bees, cows, horses, chickens, and pigs and loves driving a tractor.

Delivering KnowHow www.doulos.com

What Can Formal Do For Me?

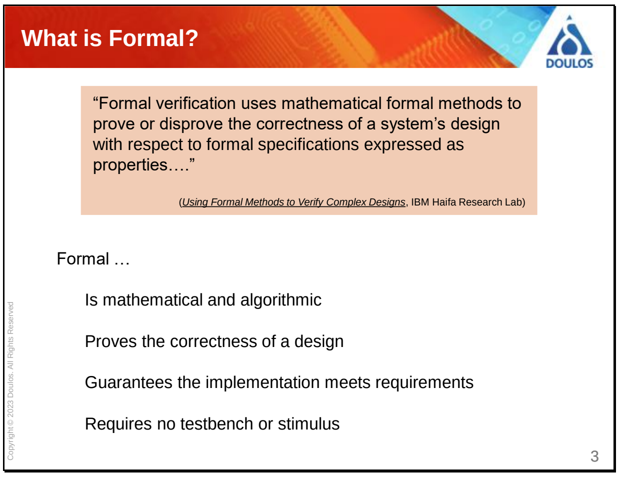
Presenter: Doug Smith
Engineer / Instructor

Webinar partner:



What Can Formal Do For Me?

- What is formal?
- Where can formal be used?
- Applications for formal
- Wrap-up

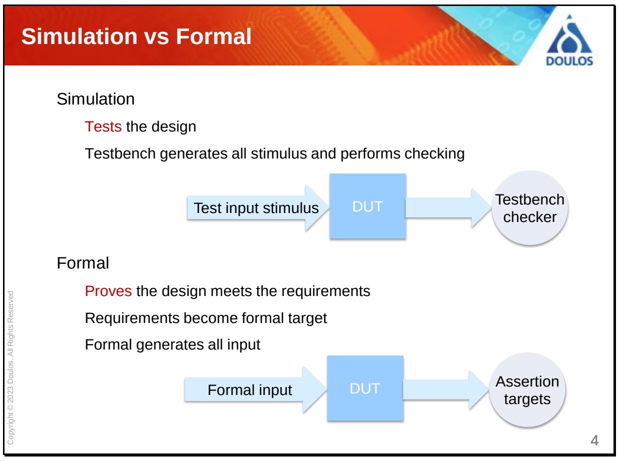


What is Formal?

> "Formal verification uses mathematical formal methods to prove or disprove the correctness of a system's design with respect to formal specifications expressed as properties…."
>
> (*Using Formal Methods to Verify Complex Designs*, IBM Haifa Research Lab)

Formal …

- Is mathematical and algorithmic
- Proves the correctness of a design
- Guarantees the implementation meets requirements
- Requires no testbench or stimulus



Simulation vs Formal

Simulation

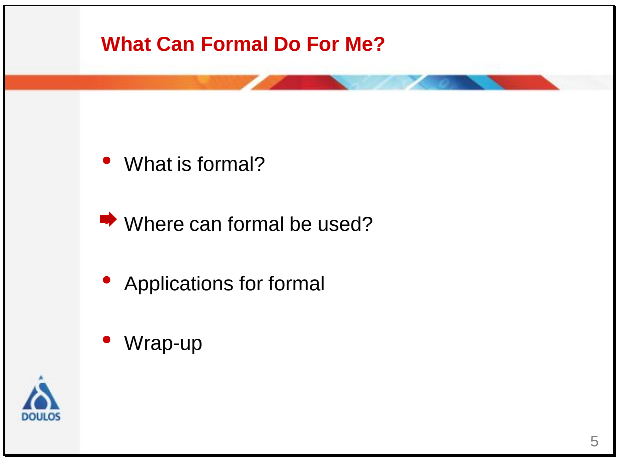
- Tests the design
- Testbench generates all stimulus and performs checking

Test input stimulus
DUT
Testbench checker

Formal

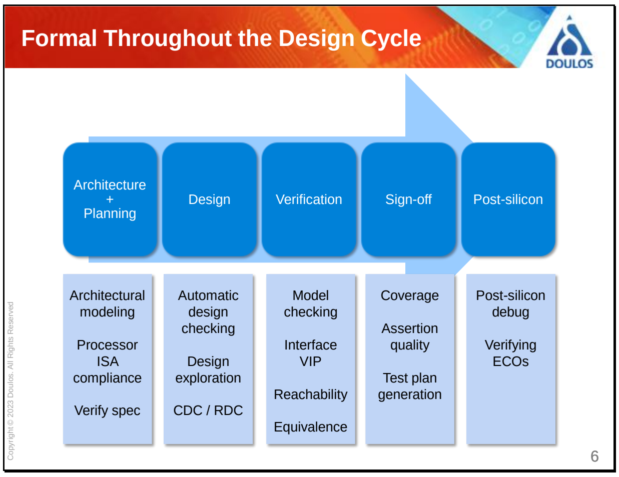
- Proves the design meets the requirements
- Requirements become formal target
- Formal generates all input

Formal input
DUT
Assertion targets



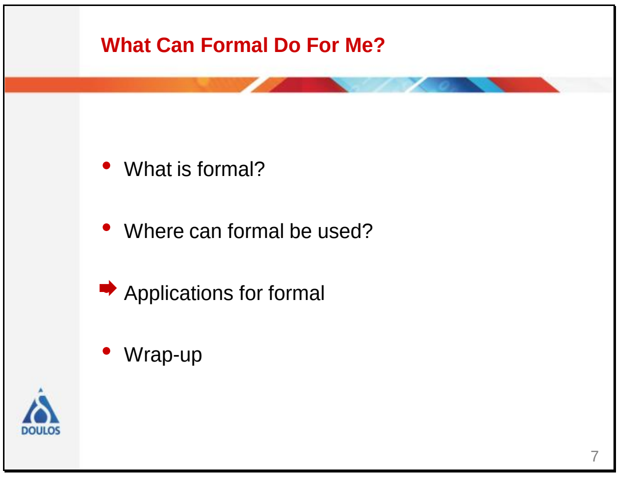
What Can Formal Do For Me?

- What is formal?
- ➡ Where can formal be used?
- Applications for formal
- Wrap-up



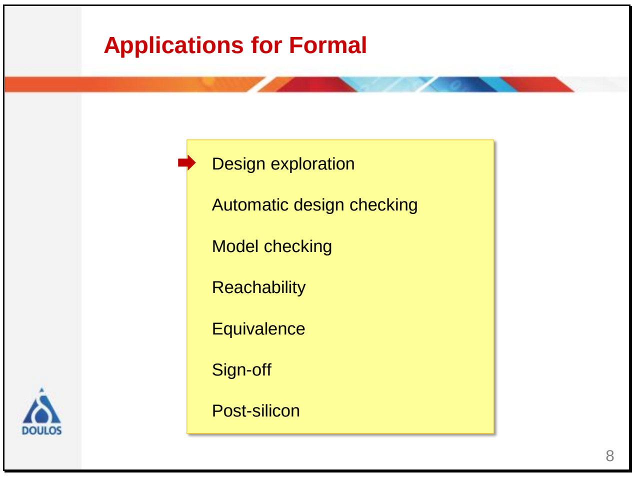
Formal Throughout the Design Cycle

Architecture + Planning	Design	Verification	Sign-off	Post-silicon
Architectural modeling Processor ISA compliance Verify spec	Automatic design checking Design exploration CDC / RDC	Model checking Interface VIP Reachability Equivalence	Coverage Assertion quality Test plan generation	Post-silicon debug Verifying ECOs



What Can Formal Do For Me?

- What is formal?
- Where can formal be used?
- ➡ Applications for formal
- Wrap-up



Applications for Formal

- ➡ Design exploration
- Automatic design checking
- Model checking
- Reachability
- Equivalence
- Sign-off
- Post-silicon

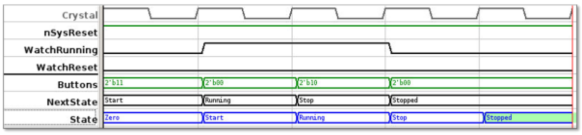
Design Exploration

```
unique case (State)
        Zero: if (Buttons[1]) NextState = Start;
       Start: begin
                 WatchRunning = 1;
                 if (!Buttons) NextState = Running;
              end
     Running: begin
                 WatchRunning = 1;
                 if (Buttons[1]) NextState = Stop;
              end
        Stop: if (!Buttons  ) NextState = Stopped;
     Stopped: if (Buttons[1]) NextState = Start;
              else if (Buttons[2]) NextState = Reset;
       Reset: begin
                 WatchReset = 1;
                 if (!Buttons) NextState = Zero;
              end
   endcase
```

```
cover property ( State == Stopped );
```

Formal Generated Trace

```
cover property ( State == Stopped );
```

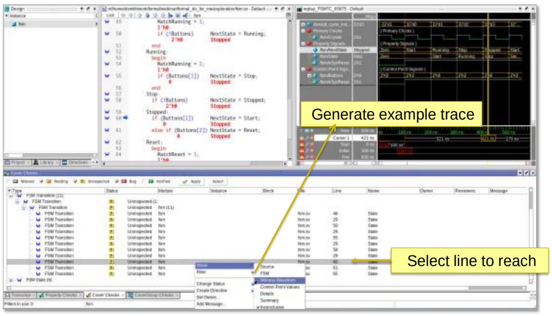


Design visualization with …

No testbench

No testcase

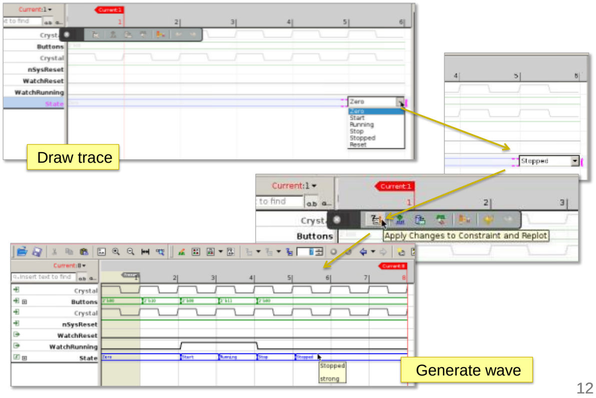
Auto Trace from Coverage App



Generate example trace

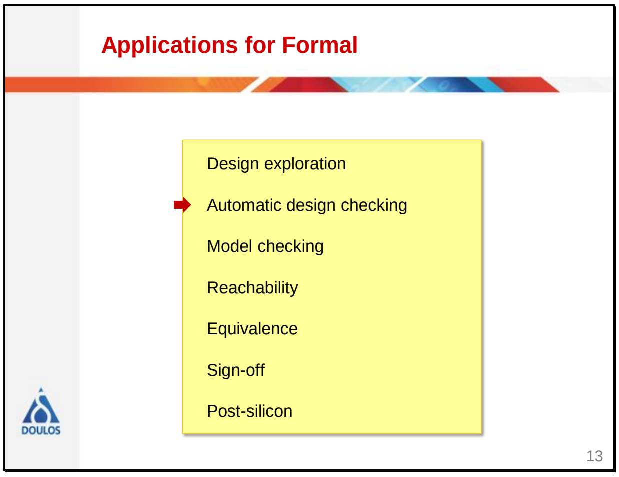
Select line to reach

Draw a Scenario



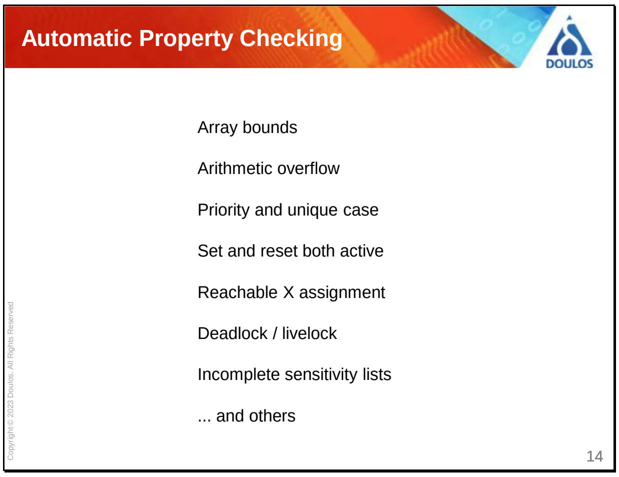
Draw trace

Generate wave



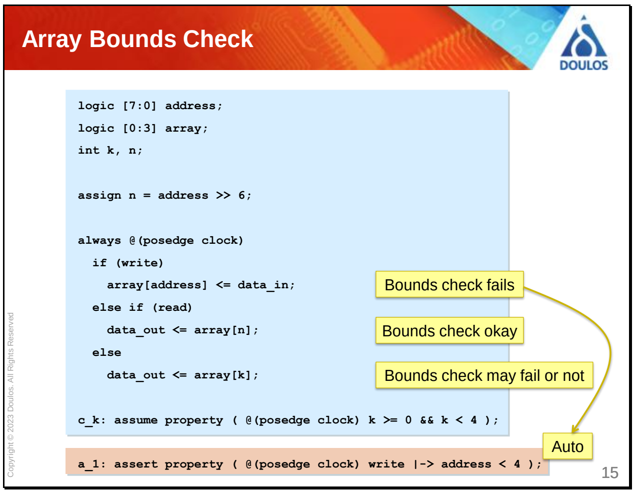
Applications for Formal

- Design exploration
- ➡ Automatic design checking
- Model checking
- Reachability
- Equivalence
- Sign-off
- Post-silicon



Automatic Property Checking

- Array bounds
- Arithmetic overflow
- Priority and unique case
- Set and reset both active
- Reachable X assignment
- Deadlock / livelock
- Incomplete sensitivity lists
- ... and others



Array Bounds Check

```
logic [7:0] address;
logic [0:3] array;
int k, n;

assign n = address >> 6;

always @(posedge clock)
  if (write)
    array[address] <= data_in;
  else if (read)
    data_out <= array[n];
  else
    data_out <= array[k];

c_k: assume property ( @(posedge clock) k >= 0 && k < 4 );
```

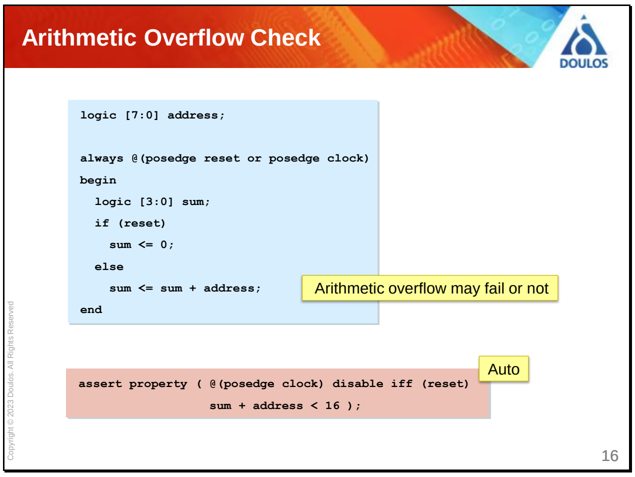
Bounds check fails

Bounds check okay

Bounds check may fail or not

Auto

```
a_1: assert property ( @(posedge clock) write |-> address < 4 );
```



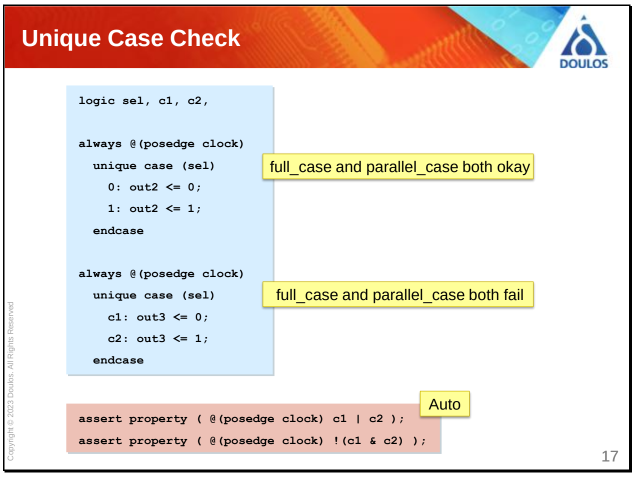
Arithmetic Overflow Check

```
logic [7:0] address;

always @(posedge reset or posedge clock)
begin
  logic [3:0] sum;
  if (reset)
    sum <= 0;
  else
    sum <= sum + address;
end
```

Arithmetic overflow may fail or not

Auto

```
assert property ( @(posedge clock) disable iff (reset)
                  sum + address < 16 );
```



Unique Case Check

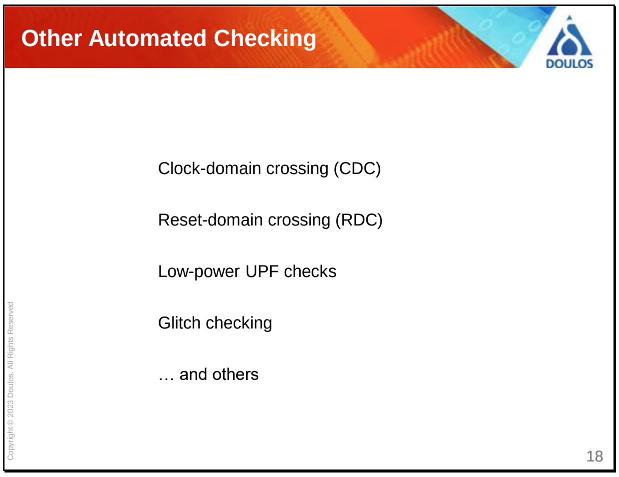
```
logic sel, c1, c2,

always @(posedge clock)
  unique case (sel)
    0: out2 <= 0;
    1: out2 <= 1;
  endcase

always @(posedge clock)
  unique case (sel)
    c1: out3 <= 0;
    c2: out3 <= 1;
  endcase
```

full_case and parallel_case both okay

full_case and parallel_case both fail

Auto

```
assert property ( @(posedge clock) c1 | c2 );
assert property ( @(posedge clock) !(c1 & c2) );
```



Other Automated Checking

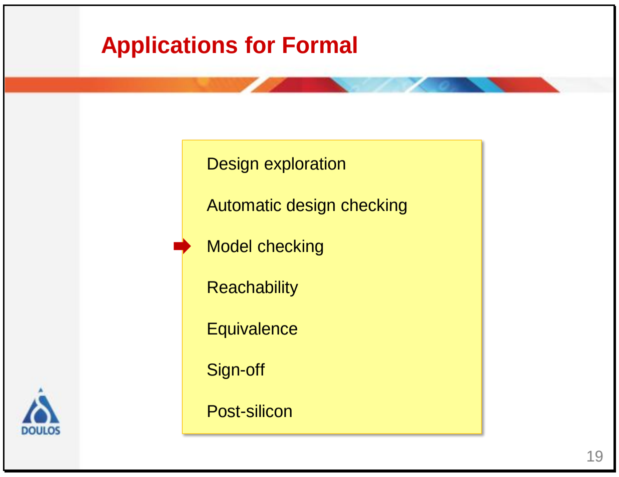
Clock-domain crossing (CDC)

Reset-domain crossing (RDC)

Low-power UPF checks

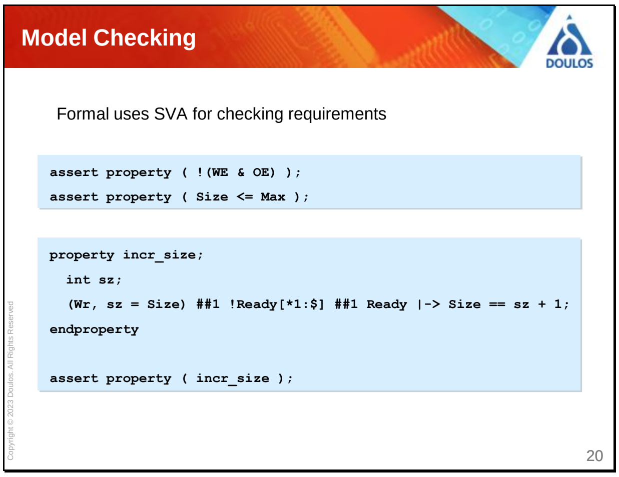
Glitch checking

… and others



Applications for Formal

- Design exploration
- Automatic design checking
- ➡ Model checking
- Reachability
- Equivalence
- Sign-off
- Post-silicon

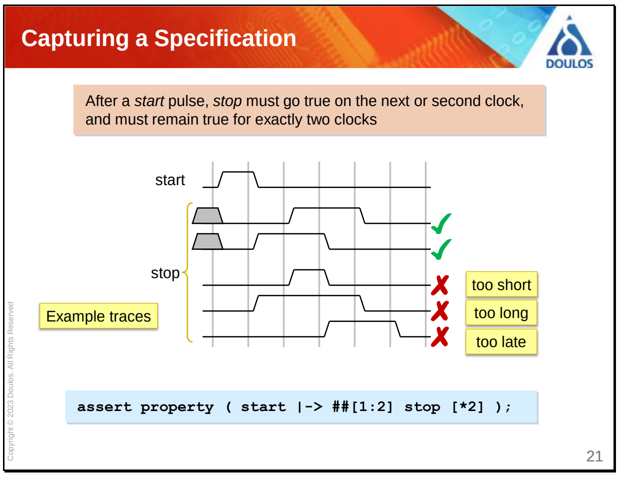


Model Checking

Formal uses SVA for checking requirements

```
assert property ( !(WE & OE) );
assert property ( Size <= Max );
```

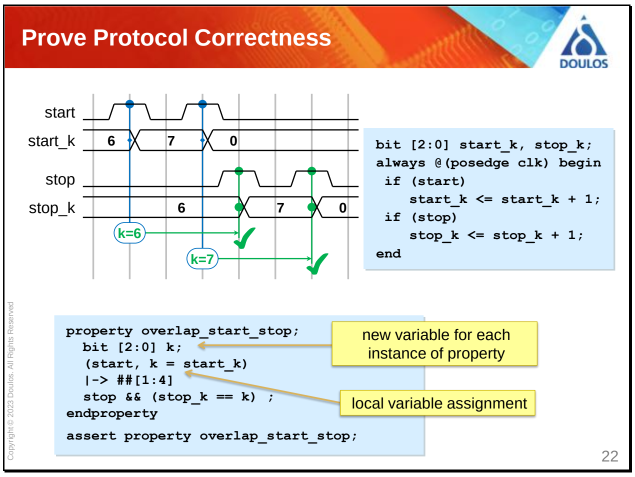
```
property incr_size;
  int sz;
  (Wr, sz = Size) ##1 !Ready[*1:$] ##1 Ready |-> Size == sz + 1;
endproperty

assert property ( incr_size );
```



Capturing a Specification

After a *start* pulse, *stop* must go true on the next or second clock, and must remain true for exactly two clocks

```
assert property ( start |-> ##[1:2] stop [*2] );
```



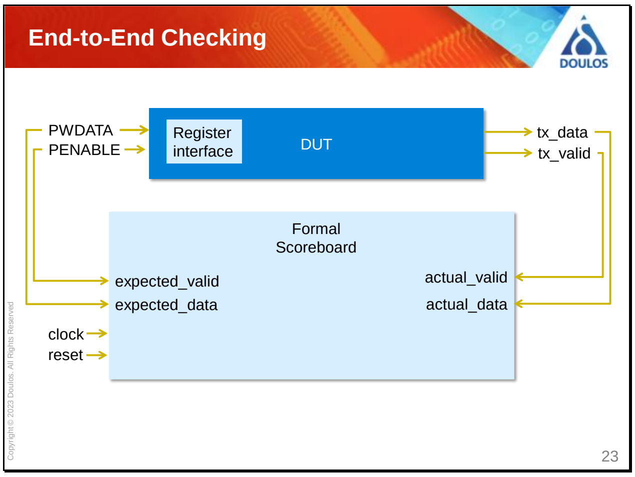
Prove Protocol Correctness

```
bit [2:0] start_k, stop_k;
always @(posedge clk) begin
 if (start)
    start_k <= start_k + 1;
 if (stop)
    stop_k <= stop_k + 1;
end
```

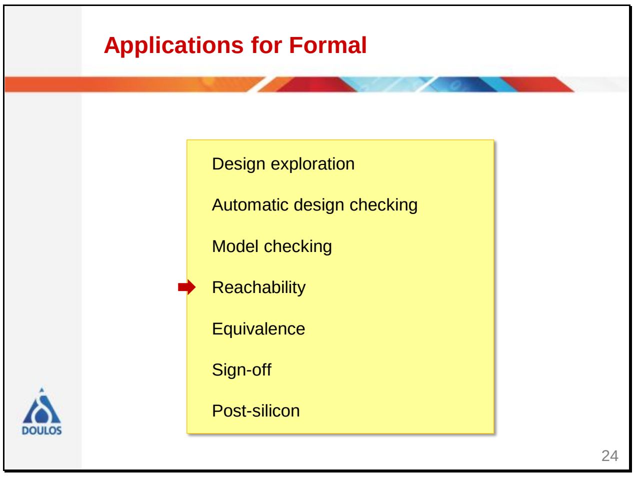
```
property overlap_start_stop;
  bit [2:0] k;
  (start, k = start_k)
  |-> ##[1:4]
  stop && (stop_k == k) ;
endproperty
assert property overlap_start_stop;
```

new variable for each instance of property

local variable assignment



End-to-End Checking

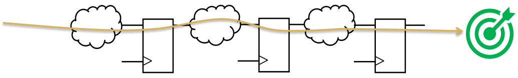


Applications for Formal

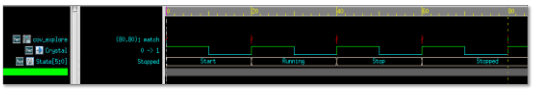
- Design exploration
- Automatic design checking
- Model checking
- ➡ Reachability
- Equivalence
- Sign-off
- Post-silicon

What Is Reachability?

Reachability – given any legal stimulus, is it possible to reach a scenario or line of code?



```
cover property ( State == Stopped );
```



Many Applications

Deadlock

Livelock

Vacuous assertions

Liveness

X-propagation

Connectivity

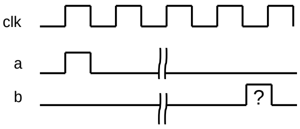
Registers

Security

Liveness

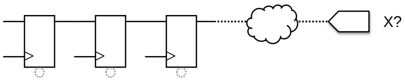
Does something eventually happen?

```
assert property ( a |-> s_eventually ( b ) );
```



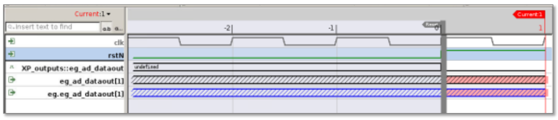
Hard (impossible) to prove in simulation

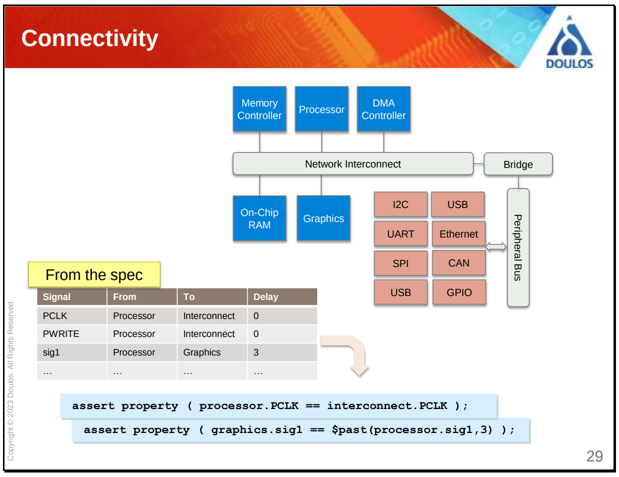
X Propagation



Non-resettable flops

```
cover property ( $isunknown( dataout ) );
```





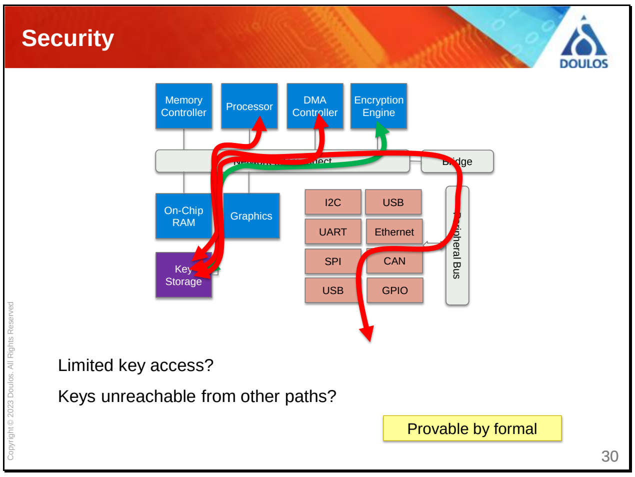
Connectivity

From the spec

Signal	From	To	Delay
PCLK	Processor	Interconnect	0
PWRITE	Processor	Interconnect	0
sig1	Processor	Graphics	3
…	…	…	…

```
assert property ( processor.PCLK == interconnect.PCLK );
```

```
assert property ( graphics.sig1 == $past(processor.sig1,3) );
```

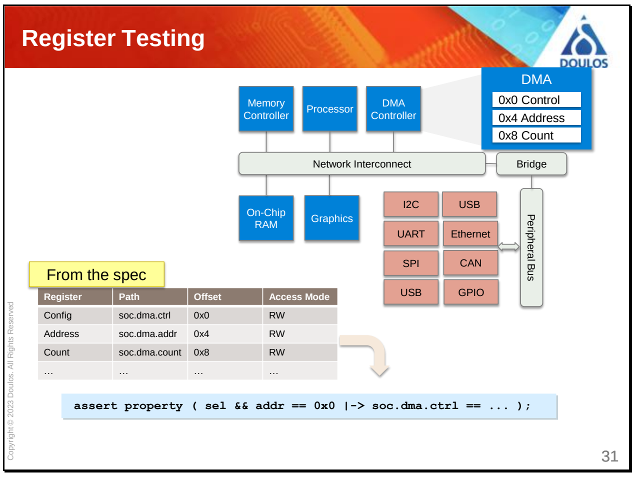


Security

Limited key access?

Keys unreachable from other paths?

Provable by formal

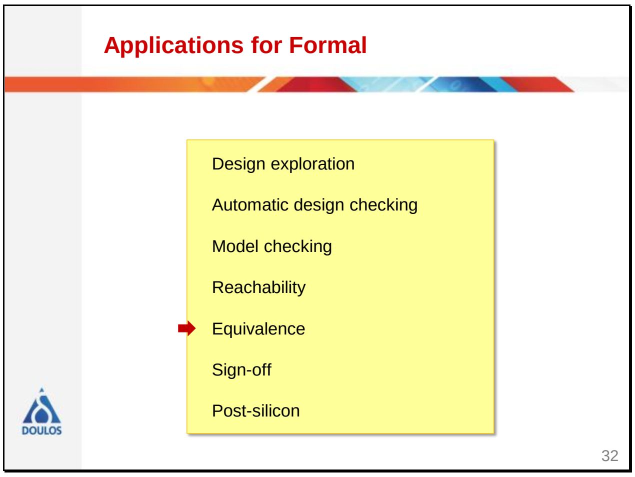


Register Testing

From the spec

Register	Path	Offset	Access Mode
Config	soc.dma.ctrl	0x0	RW
Address	soc.dma.addr	0x4	RW
Count	soc.dma.count	0x8	RW
…	…	…	…

```
assert property ( sel && addr == 0x0 |-> soc.dma.ctrl == ... );
```



Applications for Formal

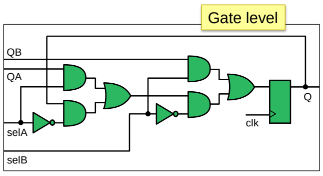
- Design exploration
- Automatic design checking
- Model checking
- Reachability
- ➡ Equivalence
- Sign-off
- Post-silicon

Logic Equivalency Checking

RTL

```
module selAB (
  input  logic clk,
  input  logic QA, selA, QB, selB,
  output logic Q
);

  always @(posedge clk)
    begin
      if (selA) Q <= QA;
      if (selB) Q <= QB;
    end

endmodule
```

Gate level



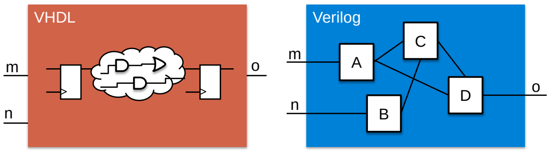
Are they functionally the same?

RTL versus gate-level netlist

Netlist versus netlist

Only works with recognizable equivalency points (signal names)

Sequential Equivalency Checking



Dynamic, not static like LEC – advances the clock

Shows equivalency between different implementations

Equivalency at the port-level

RTL <-> RTL, RTL <-> HLS (SystemC/C/C++)

Many Applications

VHDL <-> Verilog translation

Incremental feature updates (chicken bits)

ECO fixes

Data path verification

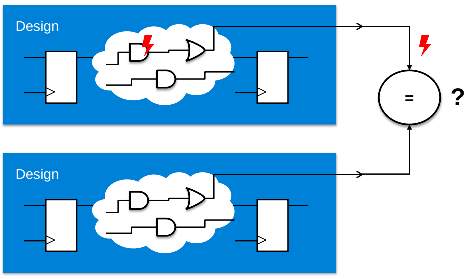
C to RTL equivalence

Functional safety

Fault injection

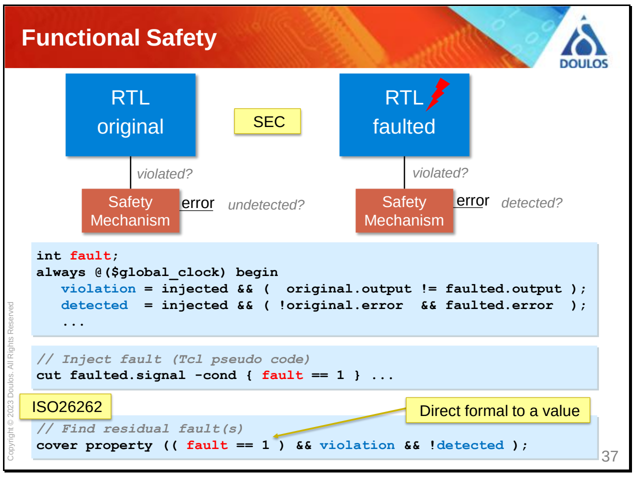
Safety mechanism insertion

Fault Injection



SEC can traverse through state better than model checking

Simply check if outputs are affected by the injected fault



Functional Safety

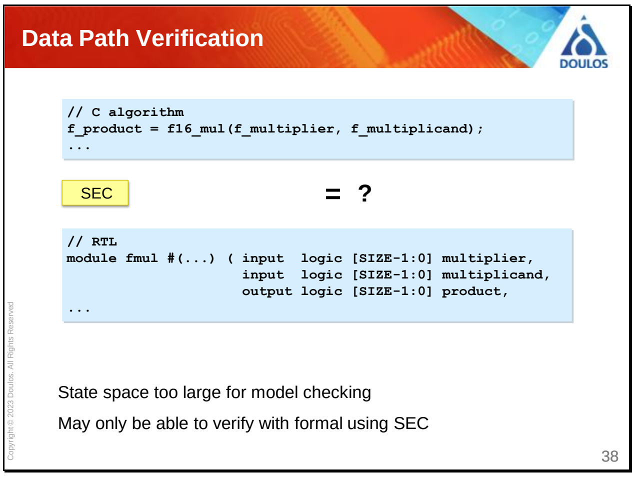
```
int fault;
always @($global_clock) begin
   violation = injected && (  original.output != faulted.output );
   detected  = injected && ( !original.error  && faulted.error  );
   ...
```

```
// Inject fault (Tcl pseudo code)
cut faulted.signal -cond { fault == 1 } ...
```

ISO26262

Direct formal to a value

```
// Find residual fault(s)
cover property (( fault == 1 ) && violation && !detected );
```



Data Path Verification

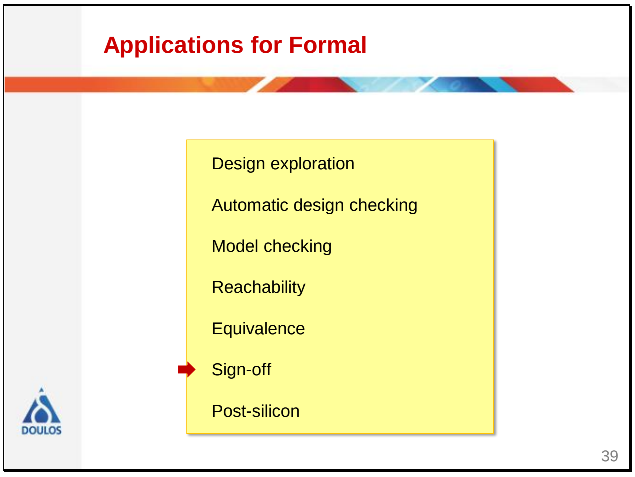
```
// C algorithm
f_product = f16_mul(f_multiplier, f_multiplicand);
...
```

SEC = ?

```
// RTL
module fmul #(...) ( input  logic [SIZE-1:0] multiplier,
                     input  logic [SIZE-1:0] multiplicand,
                     output logic [SIZE-1:0] product,
...
```

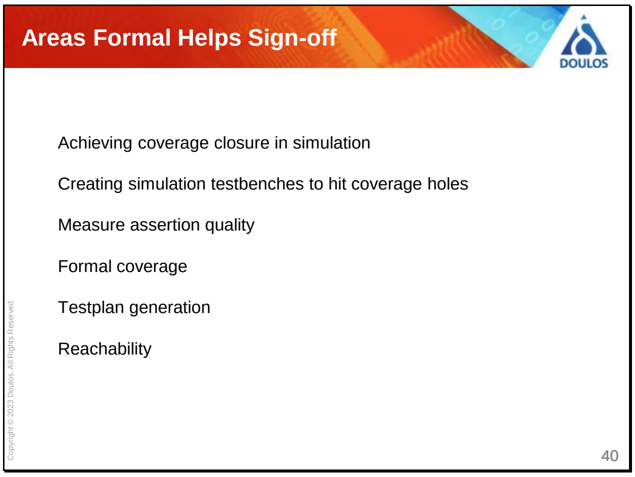
State space too large for model checking

May only be able to verify with formal using SEC



Applications for Formal

- Design exploration
- Automatic design checking
- Model checking
- Reachability
- Equivalence
- ➡ Sign-off
- Post-silicon



Areas Formal Helps Sign-off

Achieving coverage closure in simulation

Creating simulation testbenches to hit coverage holes

Measure assertion quality

Formal coverage

Testplan generation

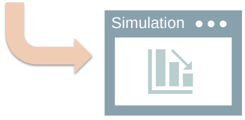
Reachability

Coverage Exclusions for Simulation

Formal finds unreachables and generates exclusions

```
<formal_tool> generate exclude exclude_file.tcl
```

```
coverage exclude -scope
{/tb_axi4lite_2_apb4/dut/u_master_interface/u_apb_master_s
c} -srcfile .../src/vlog/apb_master_sc.v -linerange 88 –
item s 1 -reason "EU"
coverage exclude -scope
{/tb_axi4lite_2_apb4/dut/u_master_interface/u_apb_master_s
c} -srcfile .../src/vlog/apb_master_sc.v -linerange 106 –
item s 1 -reason "EU"
```



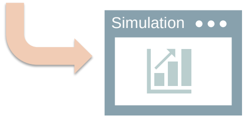
Filter coverage

Testbench Generation

Generate stimulus to target coverage holes

```
<formal_tool> generate testbenches
```

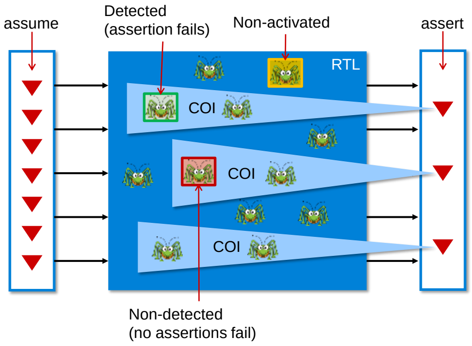
```
module replay_vlog;
initial begin
    #1;
    force axi4lite_to_apb4.use_1clk_i = 1'b0;
    force axi4lite_to_apb4.PRESETn_i = 1'b0;
    force axi4lite_to_apb4.PREADY_i = 1'b0;
    force axi4lite_to_apb4.PSLVERR_i = 1'b0;
    force axi4lite_to_apb4.PSELx_i_csr = 1'b0;
    ...
```



Fill coverage

Measuring Assertion Quality

RTL Mutation Coverage



Formal Coverage

Assertion density – are there enough?

- Cone-of-influence (COI) coverage
- Proof core coverage

Code coverage

- Proof core coverage

Functional coverage

- Cover properties
- Synthesizable covergroups

Assertion quality

- Mutation coverage

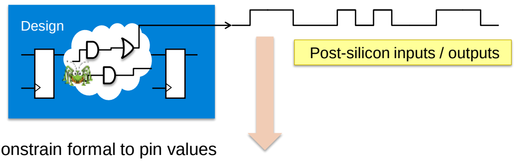
Merges with simulation coverage

Applications for Formal

- Design exploration
- Automatic design checking
- Model checking
- Reachability
- Equivalence
- Sign-off
- ➡ Post-silicon

Post-Silicon Debug

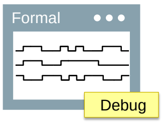
Formal can reproduce post-silicon results for debug



Constrain formal to pin values

```
assume property ( pins == ... );
```

```
cover property ( state == ERROR );
```



What Can Formal Do For Me?

- What is formal?
- Where can formal be used?
- Applications for formal
- ➡ Wrap-up

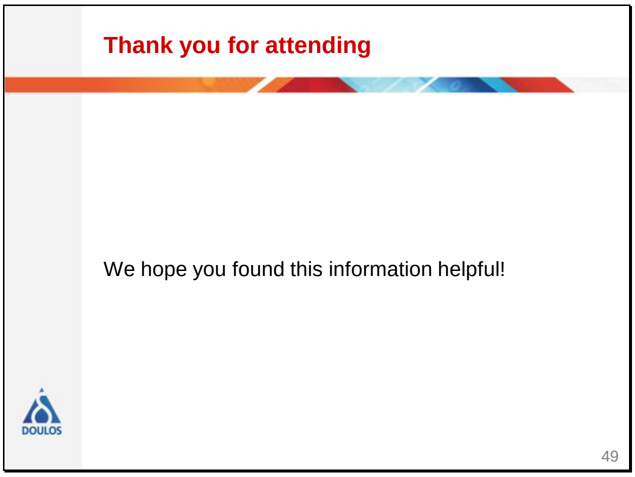
Summary

Formal complements your simulation flow

Formal verifies scenarios hard or tedious in simulation

Formal can be part of any verification planning and effort

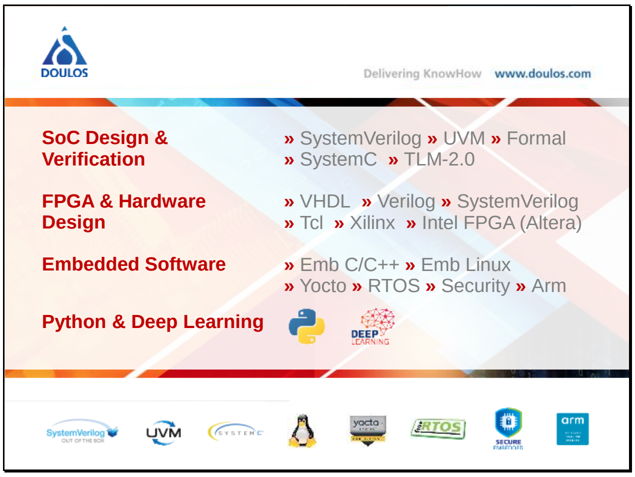
Why would you not take advantage of what formal can do?



Thank you for attending

We hope you found this information helpful!

DOULOS



DOULOS

Delivering KnowHow www.doulos.com

SoC Design & Verification	» SystemVerilog » UVM » Formal » SystemC » TLM-2.0
FPGA & Hardware Design	» VHDL » Verilog » SystemVerilog » Tcl » Xilinx » Intel FPGA (Altera)
Embedded Software	» Emb C/C++ » Emb Linux » Yocto » RTOS » Security » Arm
Python & Deep Learning	

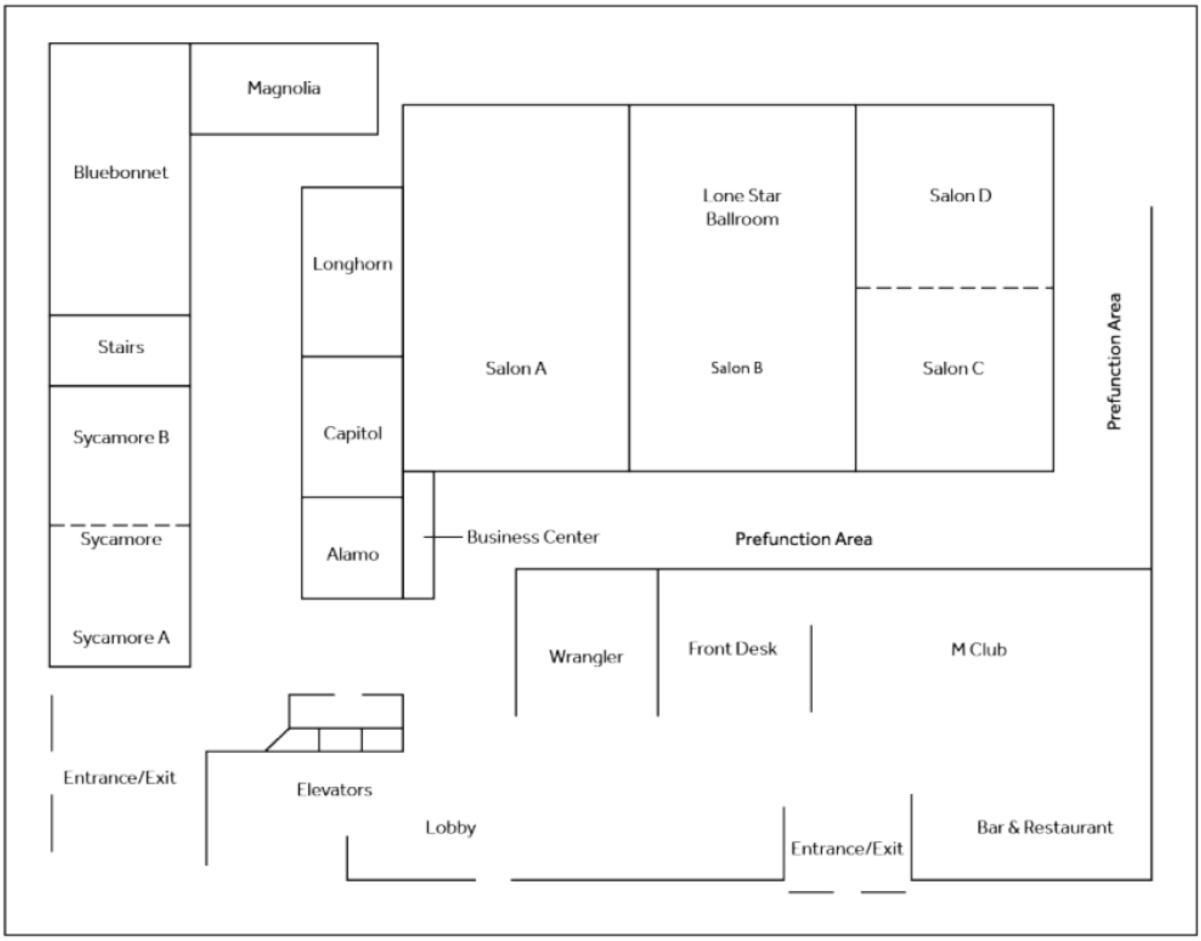
Notes

Track Session

UVM for AMS Verification

Lonestar Ballroom – Salon D

FLOOR PLAN



We would be grateful if you could move to the track session as quickly as possible.

Notes

Peter Grove & Steven Holloway

Renesas
Distinguished Member of Technical Staff &
Member of Technical Staff

Renesas's Submission to the UVM-(A)MS working group

User Paper

Abstract

Explanation on how UVM can be applied to DMS/AMS using a concept of a MS Bridge module. The focus will be on an AMS Device-Under-Test, but the concepts work for DMS. The audience will be guided over subtleties of AMS simulators and a known limitation with the proposal and possible solutions. There will be a walk though of how this was applied to a Mixed signal block. The audience should not take way the current implementation until an official release of UVM-AMS has been made. The current contents of the presentation and example code has been shared to the EDA community to feed into a white paper on the topic. Steven will cover the UVM aspects and Peter will go over mixed signal parts.

DMS – Digital-Mixed-Signal also referred to as Real-Number-Modelling
AMS – Analog-Mixed-Signal
MS – Mixed Signal

Biography

<u>Peter Grove</u>

Peter has worked in the industry starting back in 2001 when he joined a small company called Wolfson MicroElectronics, where he was project lead for more than 15 production devices. Since then Peter has only worked at one other company, Nujira, before joining Dialog (now Renesas) at their Edinburgh office. Peter has been with Dialog since 2014. Peter's background has been main digital design, but has over the years taken charge of many large mixed signal devices that are in volume production and been exposed to enough analogue design work to appreciate the issues they face in verification. Peter has an eye for looking for ways in which techniques can be done to improve chip level coverage, simulation runtime improvement to name a few.

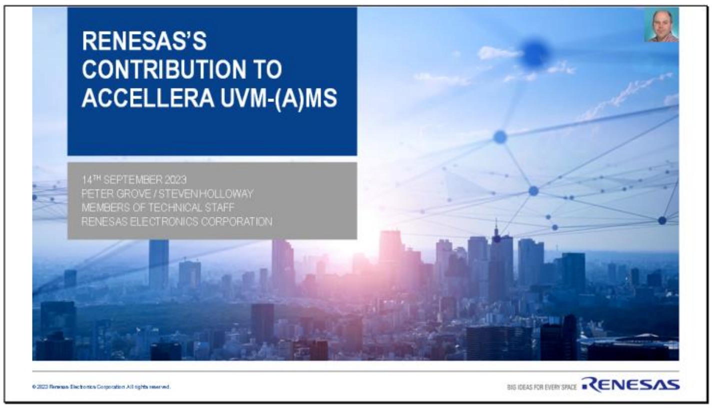
Peter is also in a unique position that during his days at Wolfson he was a key player in defining their schematic/Layout tool set with integrated revision control. This has allowed Peter to gather a large number of skills not just in design work but in all the backend flows and EDA tools, understanding different netlist types and how the tools work.

Peter's technical interests are mixed signal and analogue verification methodologies, design flows. Peter also is an Acellera SystemVerilog-AMS committee chair, UVM-AMS member/key contributor making sure the 'users' feedback on the language is considered and not what the vendors just want to support.

Steven Holloway

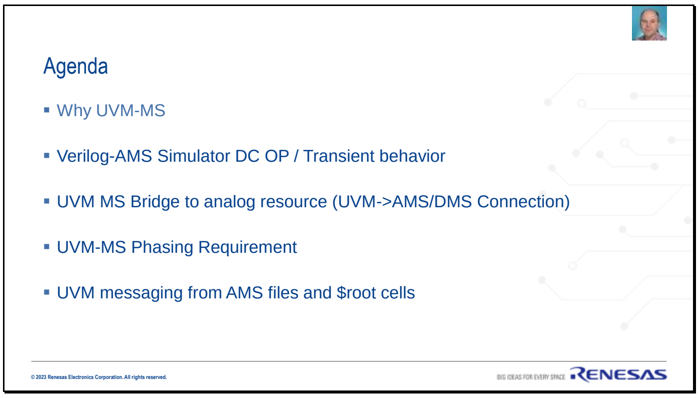
Steve has 20+ years' experience of digital verification including eRM, OVM, UVM and formal property checking. He has led the verification of large-scale consumer SoC projects. He joined Dialog Semiconductor in 2011 and previously worked for Doulos, NXP and Trident Microsystems. He joined the Technical Ladder in 2015.

Steve has presented at multiple external conferences including a panel session at DVCon US. He participates in industry standards bodies and has contributed code to the Accellera UVM-AMS working group



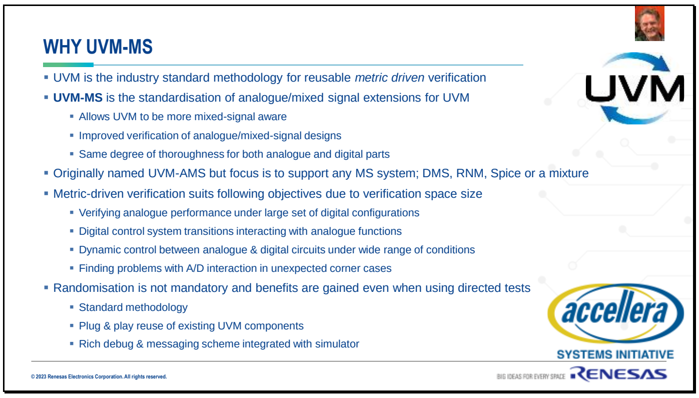
RENESAS'S CONTRIBUTION TO ACCELLERA UVM-(A)MS

14TH SEPTEMBER 2023
PETER GROVE / STEVEN HOLLOWAY
MEMBERS OF TECHNICAL STAFF
RENESAS ELECTRONICS CORPORATION



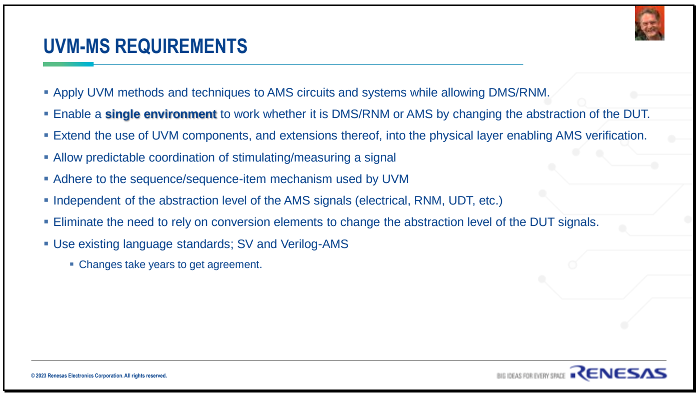
Agenda

- Why UVM-MS
- Verilog-AMS Simulator DC OP / Transient behavior
- UVM MS Bridge to analog resource (UVM->AMS/DMS Connection)
- UVM-MS Phasing Requirement
- UVM messaging from AMS files and $root cells



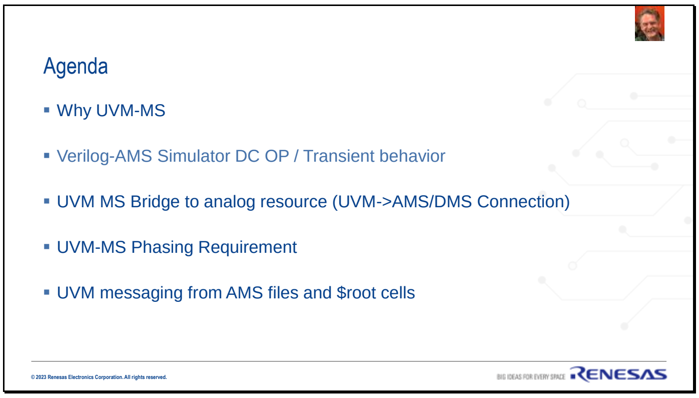
WHY UVM-MS

- UVM is the industry standard methodology for reusable *metric driven* verification
- **UVM-MS** is the standardisation of analogue/mixed signal extensions for UVM
  - Allows UVM to be more mixed-signal aware
  - Improved verification of analogue/mixed-signal designs
  - Same degree of thoroughness for both analogue and digital parts
- Originally named UVM-AMS but focus is to support any MS system; DMS, RNM, Spice or a mixture
- Metric-driven verification suits following objectives due to verification space size
  - Verifying analogue performance under large set of digital configurations
  - Digital control system transitions interacting with analogue functions
  - Dynamic control between analogue & digital circuits under wide range of conditions
  - Finding problems with A/D interaction in unexpected corner cases
- Randomisation is not mandatory and benefits are gained even when using directed tests
  - Standard methodology
  - Plug & play reuse of existing UVM components
  - Rich debug & messaging scheme integrated with simulator



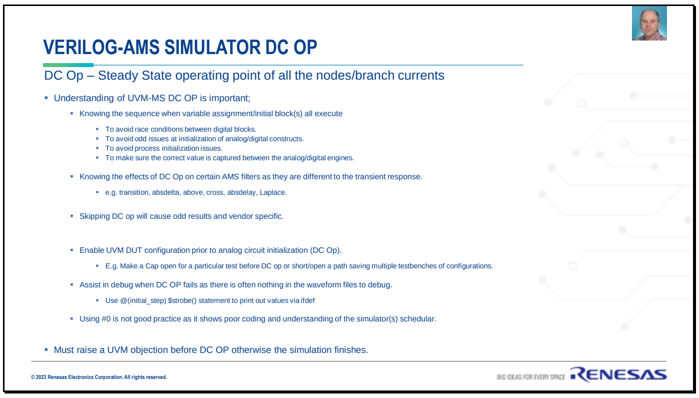
UVM-MS REQUIREMENTS

- Apply UVM methods and techniques to AMS circuits and systems while allowing DMS/RNM.
- Enable a **single environment** to work whether it is DMS/RNM or AMS by changing the abstraction of the DUT.
- Extend the use of UVM components, and extensions thereof, into the physical layer enabling AMS verification.
- Allow predictable coordination of stimulating/measuring a signal
- Adhere to the sequence/sequence-item mechanism used by UVM
- Independent of the abstraction level of the AMS signals (electrical, RNM, UDT, etc.)
- Eliminate the need to rely on conversion elements to change the abstraction level of the DUT signals.
- Use existing language standards; SV and Verilog-AMS
  - Changes take years to get agreement.



Agenda

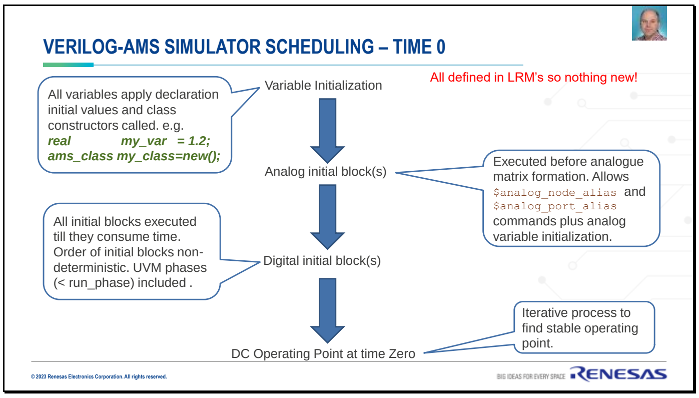
- Why UVM-MS
- Verilog-AMS Simulator DC OP / Transient behavior
- UVM MS Bridge to analog resource (UVM->AMS/DMS Connection)
- UVM-MS Phasing Requirement
- UVM messaging from AMS files and $root cells



VERILOG-AMS SIMULATOR DC OP

DC Op – Steady State operating point of all the nodes/branch currents

- Understanding of UVM-MS DC OP is important;
  - Knowing the sequence when variable assignment/initial block(s) all execute
    - To avoid race conditions between digital blocks.
    - To avoid odd issues at initialization of analog/digital constructs.
    - To avoid process initialization issues.
    - To make sure the correct value is captured between the analog/digital engines.
  - Knowing the effects of DC Op on certain AMS filters as they are different to the transient response.
    - e.g. transition, absdelta, above, cross, absdelay, Laplace.
  - Skipping DC op will cause odd results and vendor specific.
  - Enable UVM DUT configuration prior to analog circuit initialization (DC Op).
    - E.g. Make a Cap open for a particular test before DC op or short/open a path saving multiple testbenches of configurations.
  - Assist in debug when DC OP fails as there is often nothing in the waveform files to debug.
    - Use @(initial_step) $strobe() statement to print out values via ifdef
  - Using #0 is not good practice as it shows poor coding and understanding of the simulator(s) schedular.
- Must raise a UVM objection before DC OP otherwise the simulation finishes.



VERILOG-AMS SIMULATOR SCHEDULING – TIME 0

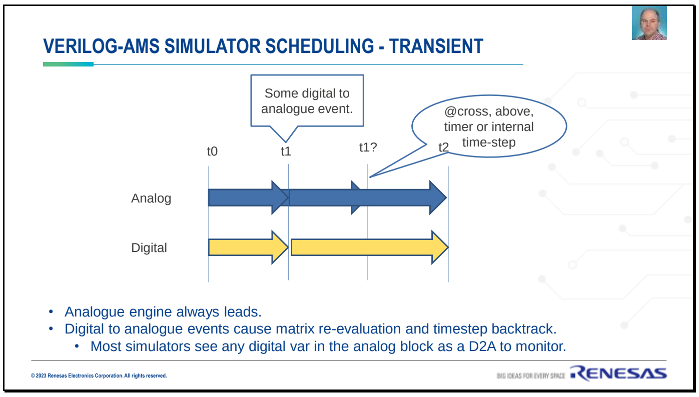
All defined in LRM's so nothing new!

All variables apply declaration initial values and class constructors called. e.g.
real my_var = 1.2;
ams_class my_class=new();

All initial blocks executed till they consume time. Order of initial blocks non-deterministic. UVM phases (< run_phase) included .

Executed before analogue matrix formation. Allows `$analog_node_alias` and `$analog_port_alias` commands plus analog variable initialization.

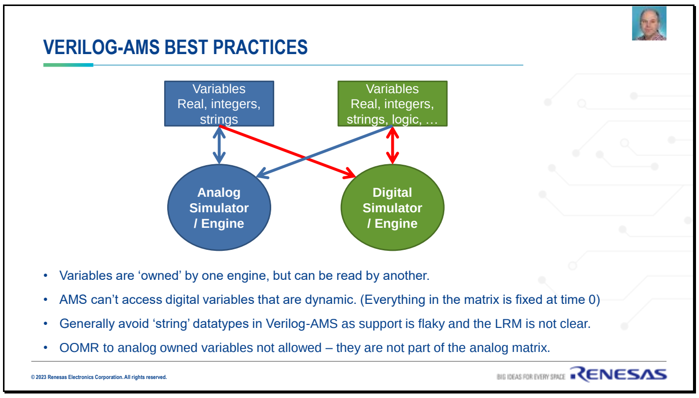
Iterative process to find stable operating point.



VERILOG-AMS SIMULATOR SCHEDULING - TRANSIENT

Some digital to analogue event.
@cross, above, timer or internal time-step
t0
t1
t1?
t2
Analog
Digital

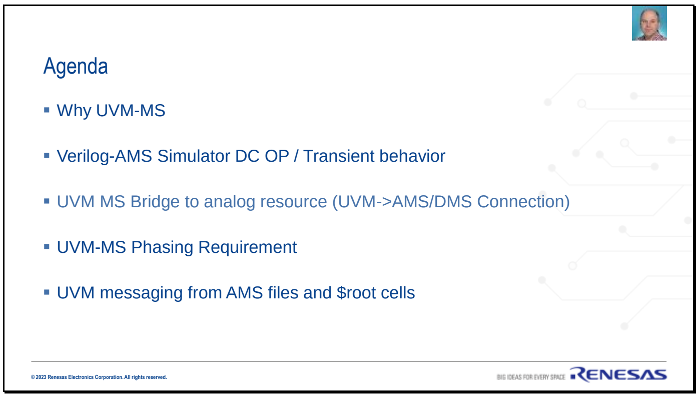
- Analogue engine always leads.
- Digital to analogue events cause matrix re-evaluation and timestep backtrack.
  - Most simulators see any digital var in the analog block as a D2A to monitor.



VERILOG-AMS BEST PRACTICES

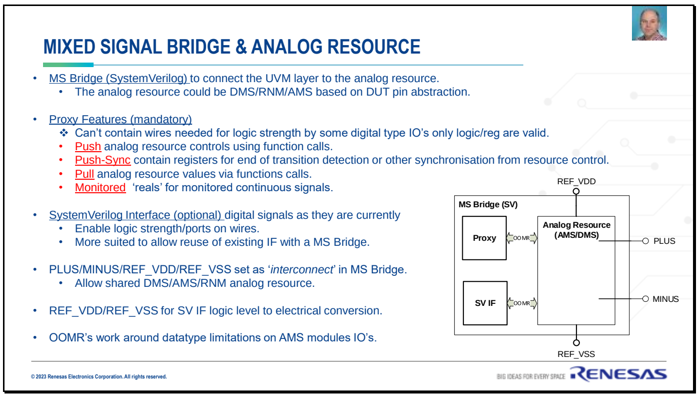
Variables
Real, integers, strings
Variables
Real, integers, strings, logic, …
Analog Simulator / Engine
Digital Simulator / Engine

- Variables are ‘owned’ by one engine, but can be read by another.
- AMS can’t access digital variables that are dynamic. (Everything in the matrix is fixed at time 0)
- Generally avoid ‘string’ datatypes in Verilog-AMS as support is flaky and the LRM is not clear.
- OOMR to analog owned variables not allowed – they are not part of the analog matrix.



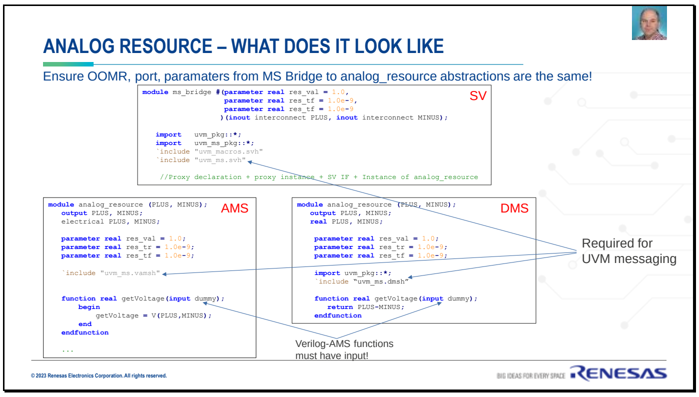
Agenda

- Why UVM-MS
- Verilog-AMS Simulator DC OP / Transient behavior
- UVM MS Bridge to analog resource (UVM->AMS/DMS Connection)
- UVM-MS Phasing Requirement
- UVM messaging from AMS files and $root cells



MIXED SIGNAL BRIDGE & ANALOG RESOURCE

- MS Bridge (SystemVerilog) to connect the UVM layer to the analog resource.
  - The analog resource could be DMS/RNM/AMS based on DUT pin abstraction.
- Proxy Features (mandatory)
  - ❖ Can’t contain wires needed for logic strength by some digital type IO’s only logic/reg are valid.
  - Push analog resource controls using function calls.
  - Push-Sync contain registers for end of transition detection or other synchronisation from resource control.
  - Pull analog resource values via functions calls.
  - Monitored ‘reals’ for monitored continuous signals.
- SystemVerilog Interface (optional) digital signals as they are currently
  - Enable logic strength/ports on wires.
  - More suited to allow reuse of existing IF with a MS Bridge.
- PLUS/MINUS/REF_VDD/REF_VSS set as ‘*interconnect*’ in MS Bridge.
  - Allow shared DMS/AMS/RNM analog resource.
- REF_VDD/REF_VSS for SV IF logic level to electrical conversion.
- OOMR’s work around datatype limitations on AMS modules IO’s.



ANALOG RESOURCE – WHAT DOES IT LOOK LIKE

Ensure OOMR, port, paramaters from MS Bridge to analog_resource abstractions are the same!

SV

```
module ms_bridge #(parameter real res_val = 1.0,
                   parameter real res_tf = 1.0e-9,
                   parameter real res_tf = 1.0e-9
                  )(inout interconnect PLUS, inout interconnect MINUS);

   import   uvm_pkg::*;
   import   uvm_ms_pkg::*;
   `include "uvm_macros.svh"
   `include "uvm_ms.svh"

    //Proxy declaration + proxy instance + SV IF + Instance of analog_resource
```

AMS

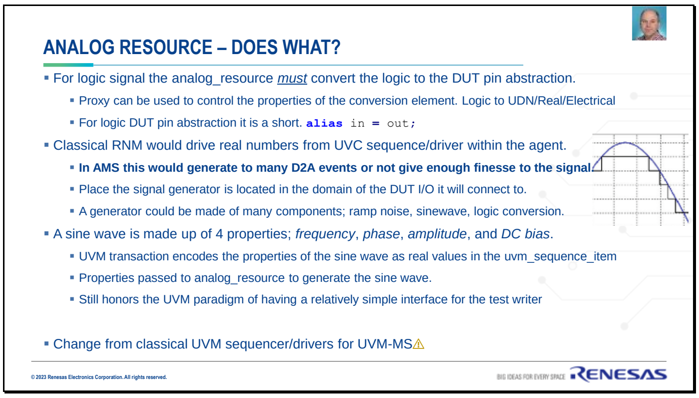
```
module analog_resource (PLUS, MINUS);
   output PLUS, MINUS;
   electrical PLUS, MINUS;

   parameter real res_val = 1.0;
   parameter real res_tr = 1.0e-9;
   parameter real res_tf = 1.0e-9;

   `include "uvm_ms.vamsh"


   function real getVoltage(input dummy);
       begin
           getVoltage = V(PLUS,MINUS);
       end
   endfunction

   ...
```

DMS

```
module analog_resource (PLUS, MINUS);
   output PLUS, MINUS;
   real PLUS, MINUS;

    parameter real res_val = 1.0;
    parameter real res_tr = 1.0e-9;
    parameter real res_tf = 1.0e-9;

    import uvm_pkg::*;
    `include “uvm_ms.dmsh”

    function real getVoltage(input dummy);
       return PLUS-MINUS;
    endfunction
```

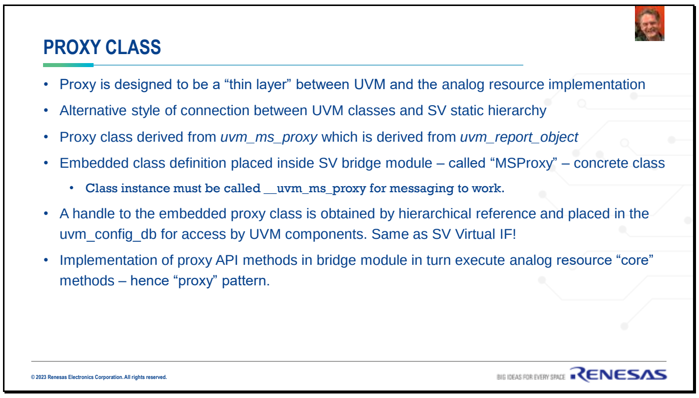
Required for UVM messaging

Verilog-AMS functions must have input!



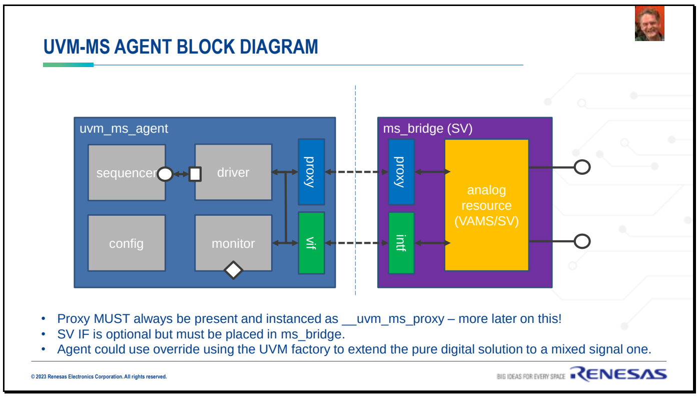
ANALOG RESOURCE – DOES WHAT?

- For logic signal the analog_resource *must* convert the logic to the DUT pin abstraction.
  - Proxy can be used to control the properties of the conversion element. Logic to UDN/Real/Electrical
  - For logic DUT pin abstraction it is a short. `alias in = out;`
- Classical RNM would drive real numbers from UVC sequence/driver within the agent.
  - **In AMS this would generate to many D2A events or not give enough finesse to the signal.**
  - Place the signal generator is located in the domain of the DUT I/O it will connect to.
  - A generator could be made of many components; ramp noise, sinewave, logic conversion.
- A sine wave is made up of 4 properties; *frequency*, *phase*, *amplitude*, and *DC bias*.
  - UVM transaction encodes the properties of the sine wave as real values in the uvm_sequence_item
  - Properties passed to analog_resource to generate the sine wave.
  - Still honors the UVM paradigm of having a relatively simple interface for the test writer
- Change from classical UVM sequencer/drivers for UVM-MS⚠



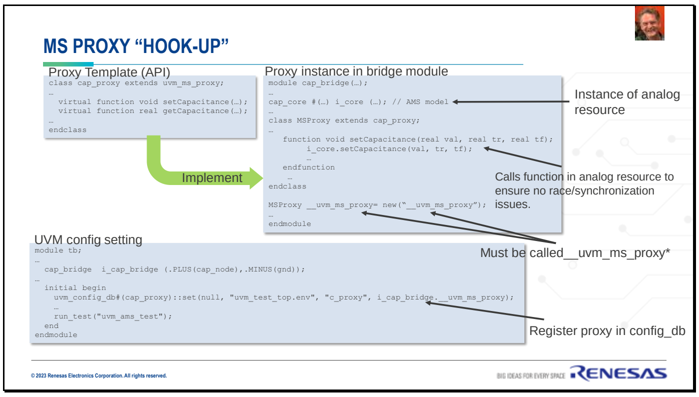
PROXY CLASS

- Proxy is designed to be a “thin layer” between UVM and the analog resource implementation
- Alternative style of connection between UVM classes and SV static hierarchy
- Proxy class derived from *uvm_ms_proxy* which is derived from *uvm_report_object*
- Embedded class definition placed inside SV bridge module – called “MSProxy” – concrete class
  - Class instance must be called __uvm_ms_proxy for messaging to work.
- A handle to the embedded proxy class is obtained by hierarchical reference and placed in the uvm_config_db for access by UVM components. Same as SV Virtual IF!
- Implementation of proxy API methods in bridge module in turn execute analog resource “core” methods – hence “proxy” pattern.



UVM-MS AGENT BLOCK DIAGRAM

- Proxy MUST always be present and instanced as __uvm_ms_proxy – more later on this!
- SV IF is optional but must be placed in ms_bridge.
- Agent could use override using the UVM factory to extend the pure digital solution to a mixed signal one.



MS PROXY “HOOK-UP”

Proxy Template (API)

```
class cap_proxy extends uvm_ms_proxy;
…
  virtual function void setCapacitance(…);
  virtual function real getCapacitance(…);
…
endclass
```

Implement

Proxy instance in bridge module

```
module cap_bridge(…);
…
cap_core #(…) i_core (…); // AMS model
…
class MSProxy extends cap_proxy;
…
   function void setCapacitance(real val, real tr, real tf);
        i_core.setCapacitance(val, tr, tf);
        …
   endfunction
    …
endclass

MSProxy __uvm_ms_proxy= new("__uvm_ms_proxy");
…
endmodule
```

Instance of analog resource

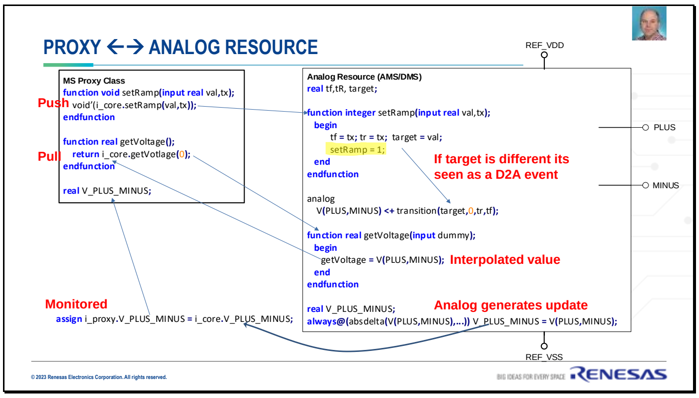
Calls function in analog resource to ensure no race/synchronization issues.

Must be called__uvm_ms_proxy*

UVM config setting

```
module tb;
…
  cap_bridge  i_cap_bridge (.PLUS(cap_node),.MINUS(gnd));
…
  initial begin
    uvm_config_db#(cap_proxy)::set(null, "uvm_test_top.env", "c_proxy", i_cap_bridge.__uvm_ms_proxy);
    …
    run_test("uvm_ams_test");
  end
endmodule
```

Register proxy in config_db



PROXY ←→ ANALOG RESOURCE

MS Proxy Class

```
function void setRamp(input real val,tx);
  void'(i_core.setRamp(val,tx));
endfunction

function real getVoltage();
   return i_core.getVotlage(0);
endfunction

real V_PLUS_MINUS;
```

Push

Pull

Monitored

```
assign i_proxy.V_PLUS_MINUS = i_core.V_PLUS_MINUS;
```

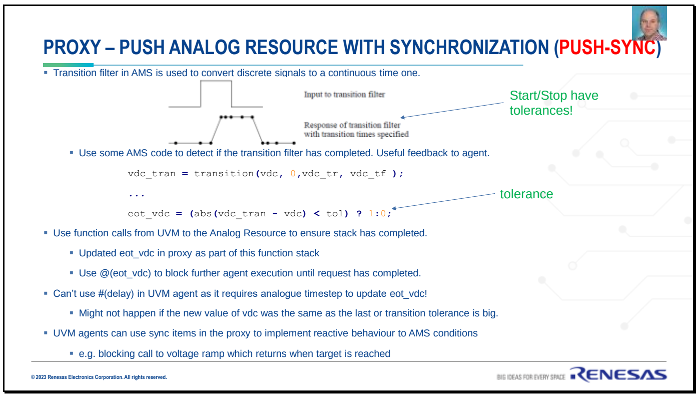
Analog Resource (AMS/DMS)

```
real tf,tR, target;

function integer setRamp(input real val,tx);
  begin
      tf = tx; tr = tx;  target = val;
      setRamp = 1;
  end
endfunction

analog
  V(PLUS,MINUS) <+ transition(target,0,tr,tf);

function real getVoltage(input dummy);
  begin
    getVoltage = V(PLUS,MINUS);
  end
endfunction

real V_PLUS_MINUS;
always@(absdelta(V(PLUS,MINUS),...)) V_PLUS_MINUS = V(PLUS,MINUS);
```

If target is different its seen as a D2A event

Interpolated value

Analog generates update



PROXY – PUSH ANALOG RESOURCE WITH SYNCHRONIZATION (PUSH-SYNC)

- Transition filter in AMS is used to convert discrete signals to a continuous time one.

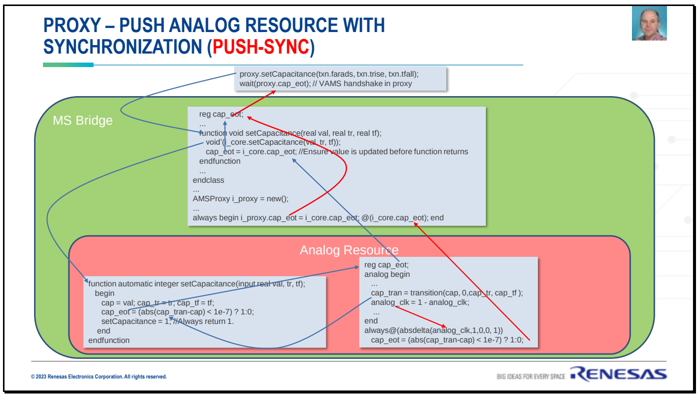
Input to transition filter
Response of transition filter with transition times specified
Start/Stop have tolerances!

  - Use some AMS code to detect if the transition filter has completed. Useful feedback to agent.

```
vdc_tran = transition(vdc, 0,vdc_tr, vdc_tf );
...
eot_vdc = (abs(vdc_tran - vdc) < tol) ? 1:0;
```

tolerance

- Use function calls from UVM to the Analog Resource to ensure stack has completed.
  - Updated eot_vdc in proxy as part of this function stack
  - Use @(eot_vdc) to block further agent execution until request has completed.
- Can't use #(delay) in UVM agent as it requires analogue timestep to update eot_vdc!
  - Might not happen if the new value of vdc was the same as the last or transition tolerance is big.
- UVM agents can use sync items in the proxy to implement reactive behaviour to AMS conditions
  - e.g. blocking call to voltage ramp which returns when target is reached



PROXY – PUSH ANALOG RESOURCE WITH SYNCHRONIZATION (PUSH-SYNC)

```
proxy.setCapacitance(txn.farads, txn.trise, txn.tfall);
wait(proxy.cap_eot); // VAMS handshake in proxy
```

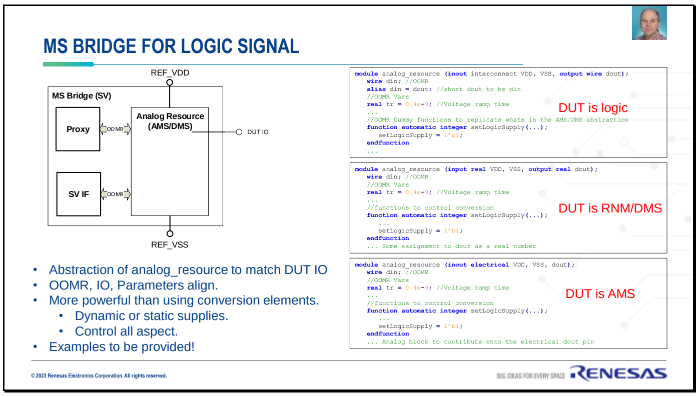
MS Bridge

```
  reg cap_eot;
  ...
  function void setCapacitance(real val, real tr, real tf);
    void'(i_core.setCapacitance(val, tr, tf));
    cap_eot = i_core.cap_eot; //Ensure value is updated before function returns
  endfunction
  ...
endclass
...
AMSProxy i_proxy = new();
...
always begin i_proxy.cap_eot = i_core.cap_eot; @(i_core.cap_eot); end
```

Analog Resource

```
function automatic integer setCapacitance(input real val, tr, tf);
  begin
    cap = val; cap_tr = tr; cap_tf = tf;
    cap_eot = (abs(cap_tran-cap) < 1e-7) ? 1:0;
    setCapacitance = 1; //Always return 1.
   end
endfunction
```

```
reg cap_eot;
analog begin
  ...
  cap_tran = transition(cap, 0,cap_tr, cap_tf );
  analog_clk = 1 - analog_clk;
   ...
end
always@(absdelta(analog_clk,1,0,0, 1))
  cap_eot = (abs(cap_tran-cap) < 1e-7) ? 1:0;
```



MS BRIDGE FOR LOGIC SIGNAL

```
module analog_resource (inout interconnect VDD, VSS, output wire dout);
   wire din; //OOMR
   alias din = dout; //short dout to be din
   //OOMR Vars
   real tr = 0.4e-9; //Voltage ramp time
   ...
   //OOMR Dummy functions to replicate whats in the AMS/DMS abstraction
   function automatic integer setLogicSupply(...);
      setLogicSupply = 1'bl;
   endfunction
   ...
```

DUT is logic

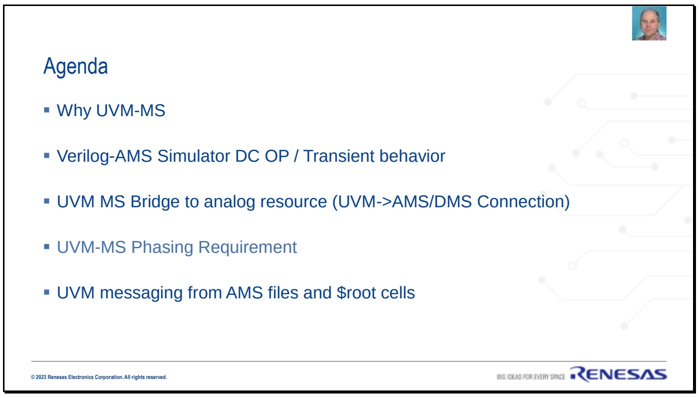
```
module analog_resource (input real VDD, VSS, output real dout);
   wire din; //OOMR
   //OOMR Vars
   real tr = 0.4e-9; //Voltage ramp time
   ...
   //functions to control conversion
   function automatic integer setLogicSupply(...);
      ...
      setLogicSupply = 1'bl;
   endfunction
   ... Some assignment to dout as a real number
```

DUT is RNM/DMS

```
module analog_resource (inout electrical VDD, VSS, dout);
   wire din; //OOMR
   //OOMR Vars
   real tr = 0.4e-9; //Voltage ramp time
   ...
   //functions to control conversion
   function automatic integer setLogicSupply(...);
      ...
      setLogicSupply = 1'bl;
   endfunction
   ... Analog block to contribute onto the electrical dout pin
```

DUT is AMS

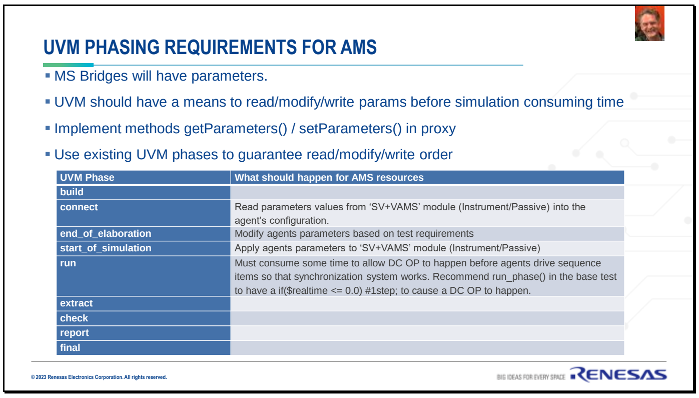
- Abstraction of analog_resource to match DUT IO
- OOMR, IO, Parameters align.
- More powerful than using conversion elements.
  - Dynamic or static supplies.
  - Control all aspect.
- Examples to be provided!



Agenda

- Why UVM-MS
- Verilog-AMS Simulator DC OP / Transient behavior
- UVM MS Bridge to analog resource (UVM->AMS/DMS Connection)
- UVM-MS Phasing Requirement
- UVM messaging from AMS files and $root cells

BIG IDEAS FOR EVERY SPACE RENESAS

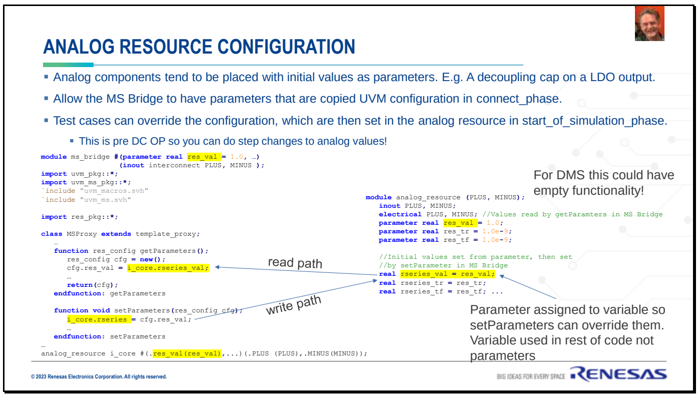


UVM PHASING REQUIREMENTS FOR AMS

- MS Bridges will have parameters.
- UVM should have a means to read/modify/write params before simulation consuming time
- Implement methods getParameters() / setParameters() in proxy
- Use existing UVM phases to guarantee read/modify/write order

UVM Phase	What should happen for AMS resources
build	
connect	Read parameters values from 'SV+VAMS' module (Instrument/Passive) into the agent's configuration.
end_of_elaboration	Modify agents parameters based on test requirements
start_of_simulation	Apply agents parameters to 'SV+VAMS' module (Instrument/Passive)
run	Must consume some time to allow DC OP to happen before agents drive sequence items so that synchronization system works. Recommend run_phase() in the base test to have a if($realtime <= 0.0) #1step; to cause a DC OP to happen.
extract	
check	
report	
final	

BIG IDEAS FOR EVERY SPACE RENESAS



ANALOG RESOURCE CONFIGURATION

- Analog components tend to be placed with initial values as parameters. E.g. A decoupling cap on a LDO output.
- Allow the MS Bridge to have parameters that are copied UVM configuration in connect_phase.
- Test cases can override the configuration, which are then set in the analog resource in start_of_simulation_phase.
  - This is pre DC OP so you can do step changes to analog values!

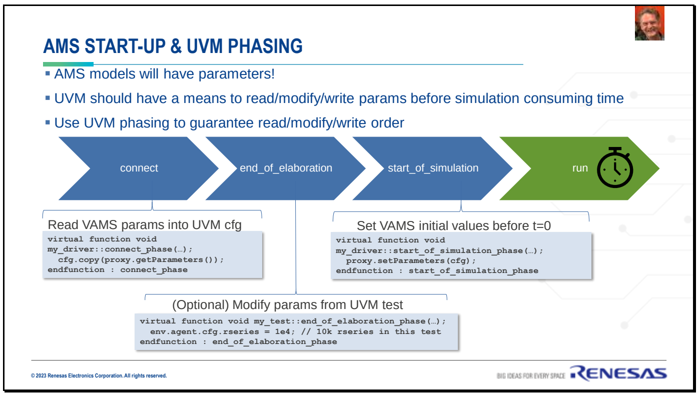
```
module ms_bridge #(parameter real res_val = 1.0, …)
                  (inout interconnect PLUS, MINUS );
import uvm_pkg::*;
import uvm_ms_pkg::*;
`include "uvm_macros.svh"
`include "uvm_ms.svh"

import res_pkg::*;

class MSProxy extends template_proxy;
   …
   function res_config getParameters();
      res_config cfg = new();
      cfg.res_val = i_core.rseries_val;
      …
      return(cfg);
   endfunction: getParameters

   function void setParameters(res_config cfg);
      i_core.rseries = cfg.res_val;
      …
   endfunction: setParameters
…
analog_resource i_core #(.res_val(res_val),...)(.PLUS (PLUS),.MINUS(MINUS));
```

For DMS this could have empty functionality!

```
module analog_resource (PLUS, MINUS);
   inout PLUS, MINUS;
   electrical PLUS, MINUS; //Values read by getParamters in MS Bridge
   parameter real res_val = 1.0;
   parameter real res_tr = 1.0e-9;
   parameter real res_tf = 1.0e-9;

   //Initial values set from parameter, then set
   //by setParameter in MS Bridge
   real rseries_val = res_val;
   real rseries_tr = res_tr;
   real rseries_tf = res_tf; ...
```

Parameter assigned to variable so setParameters can override them. Variable used in rest of code not parameters

BIG IDEAS FOR EVERY SPACE RENESAS



AMS START-UP & UVM PHASING

- AMS models will have parameters!
- UVM should have a means to read/modify/write params before simulation consuming time
- Use UVM phasing to guarantee read/modify/write order

Read VAMS params into UVM cfg

```
virtual function void
my_driver::connect_phase(…);
  cfg.copy(proxy.getParameters());
endfunction : connect_phase
```

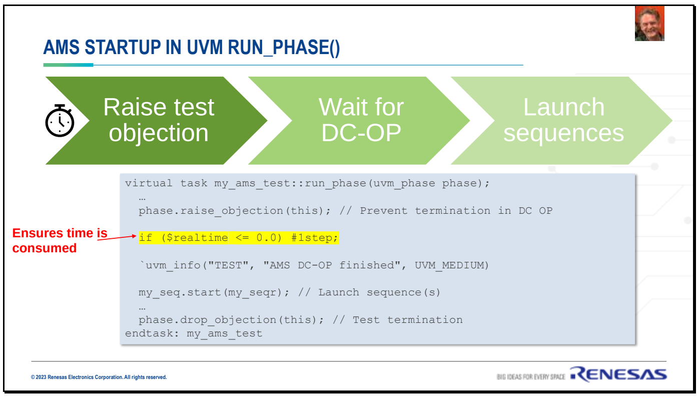
Set VAMS initial values before t=0

```
virtual function void
my_driver::start_of_simulation_phase(…);
  proxy.setParameters(cfg);
endfunction : start_of_simulation_phase
```

(Optional) Modify params from UVM test

```
virtual function void my_test::end_of_elaboration_phase(…);
  env.agent.cfg.rseries = 1e4; // 10k rseries in this test
endfunction : end_of_elaboration_phase
```

BIG IDEAS FOR EVERY SPACE RENESAS

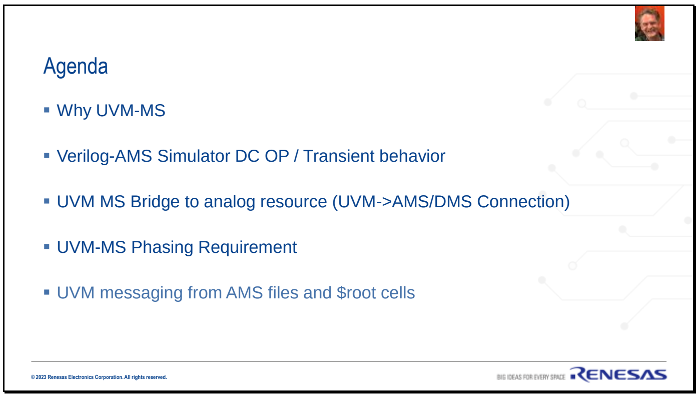


AMS STARTUP IN UVM RUN_PHASE()

Raise test objection
Wait for DC-OP
Launch sequences

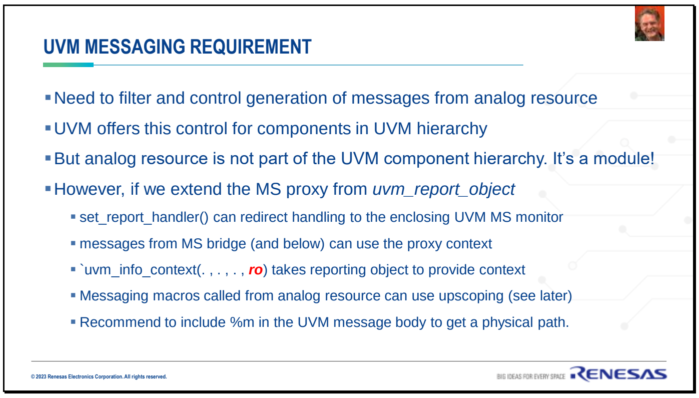
```
virtual task my_ams_test::run_phase(uvm_phase phase);
  …
  phase.raise_objection(this); // Prevent termination in DC OP

  if ($realtime <= 0.0) #1step;

  `uvm_info("TEST", "AMS DC-OP finished", UVM_MEDIUM)

  my_seq.start(my_seqr); // Launch sequence(s)
  …
  phase.drop_objection(this); // Test termination
endtask: my_ams_test
```

Ensures time is consumed



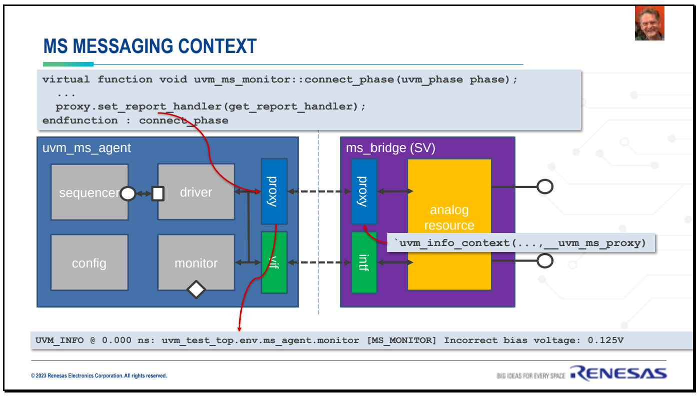
Agenda

- Why UVM-MS
- Verilog-AMS Simulator DC OP / Transient behavior
- UVM MS Bridge to analog resource (UVM->AMS/DMS Connection)
- UVM-MS Phasing Requirement
- UVM messaging from AMS files and $root cells



UVM MESSAGING REQUIREMENT

- Need to filter and control generation of messages from analog resource
- UVM offers this control for components in UVM hierarchy
- But analog resource is not part of the UVM component hierarchy. It’s a module!
- However, if we extend the MS proxy from *uvm_report_object*
  - set_report_handler() can redirect handling to the enclosing UVM MS monitor
  - messages from MS bridge (and below) can use the proxy context
  - `uvm_info_context(. , . , . , , ***ro***) takes reporting object to provide context
  - Messaging macros called from analog resource can use upscoping (see later)
  - Recommend to include %m in the UVM message body to get a physical path.

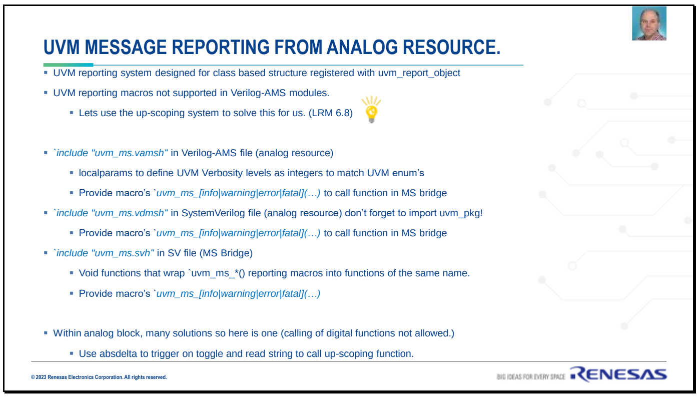


MS MESSAGING CONTEXT

```
virtual function void uvm_ms_monitor::connect_phase(uvm_phase phase);
  ...
  proxy.set_report_handler(get_report_handler);
endfunction : connect_phase
```

```
`uvm_info_context(...,__uvm_ms_proxy)
```

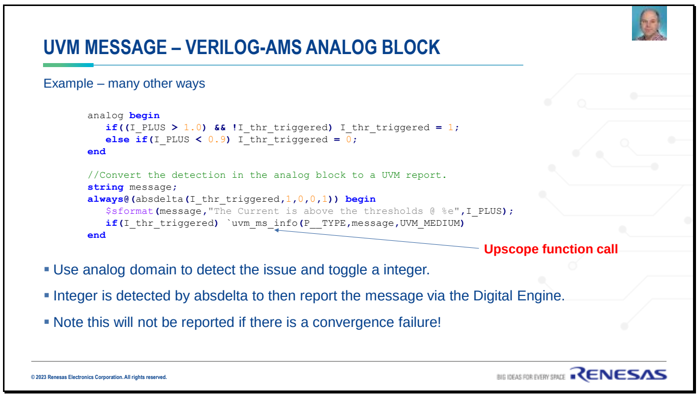
```
UVM_INFO @ 0.000 ns: uvm_test_top.env.ms_agent.monitor [MS_MONITOR] Incorrect bias voltage: 0.125V
```



UVM MESSAGE REPORTING FROM ANALOG RESOURCE.

- UVM reporting system designed for class based structure registered with uvm_report_object
- UVM reporting macros not supported in Verilog-AMS modules.
  - Lets use the up-scoping system to solve this for us. (LRM 6.8)
- *`include "uvm_ms.vamsh"* in Verilog-AMS file (analog resource)
  - localparams to define UVM Verbosity levels as integers to match UVM enum's
  - Provide macro's `*uvm_ms_[info|warning|error|fatal](…)* to call function in MS bridge
- *`include "uvm_ms.vdmsh"* in SystemVerilog file (analog resource) don't forget to import uvm_pkg!
  - Provide macro's `*uvm_ms_[info|warning|error|fatal](…)* to call function in MS bridge
- *`include "uvm_ms.svh"* in SV file (MS Bridge)
  - Void functions that wrap `uvm_ms_*() reporting macros into functions of the same name.
  - Provide macro's `*uvm_ms_[info|warning|error|fatal](…)*
- Within analog block, many solutions so here is one (calling of digital functions not allowed.)
  - Use absdelta to trigger on toggle and read string to call up-scoping function.

BIG IDEAS FOR EVERY SPACE RENESAS



UVM MESSAGE – VERILOG-AMS ANALOG BLOCK

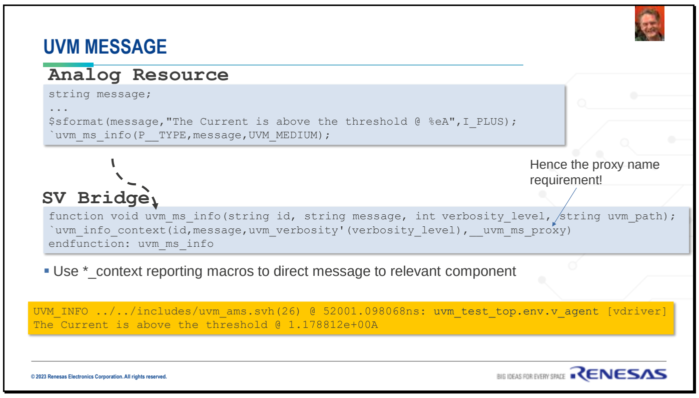
Example – many other ways

```
analog begin
   if((I_PLUS > 1.0) && !I_thr_triggered) I_thr_triggered = 1;
   else if(I_PLUS < 0.9) I_thr_triggered = 0;
end

//Convert the detection in the analog block to a UVM report.
string message;
always@(absdelta(I_thr_triggered,1,0,0,1)) begin
   $sformat(message,"The Current is above the thresholds @ %e",I_PLUS);
   if(I_thr_triggered) `uvm_ms_info(P__TYPE,message,UVM_MEDIUM)
end
```

Upscope function call

- Use analog domain to detect the issue and toggle a integer.
- Integer is detected by absdelta to then report the message via the Digital Engine.
- Note this will not be reported if there is a convergence failure!

BIG IDEAS FOR EVERY SPACE RENESAS



UVM MESSAGE

Analog Resource

```
string message;
...
$sformat(message,"The Current is above the threshold @ %eA",I_PLUS);
`uvm_ms_info(P__TYPE,message,UVM_MEDIUM);
```

SV Bridge

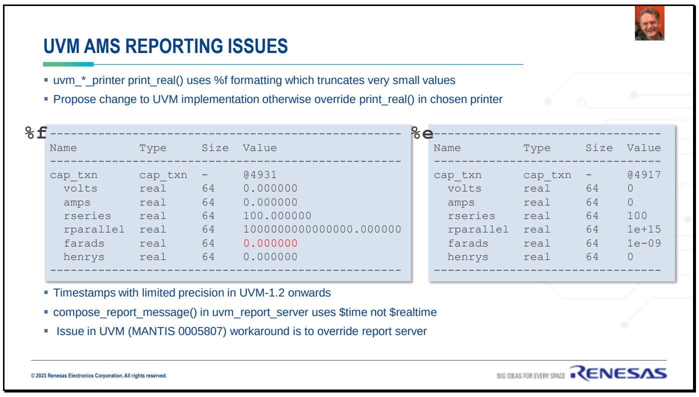
Hence the proxy name requirement!

```
function void uvm_ms_info(string id, string message, int verbosity_level, string uvm_path);
`uvm_info_context(id,message,uvm_verbosity'(verbosity_level),__uvm_ms_proxy)
endfunction: uvm_ms_info
```

- Use *_context reporting macros to direct message to relevant component

```
UVM_INFO ../../includes/uvm_ams.svh(26) @ 52001.098068ns: uvm_test_top.env.v_agent [vdriver]
The Current is above the threshold @ 1.178812e+00A
```

BIG IDEAS FOR EVERY SPACE RENESAS



UVM AMS REPORTING ISSUES

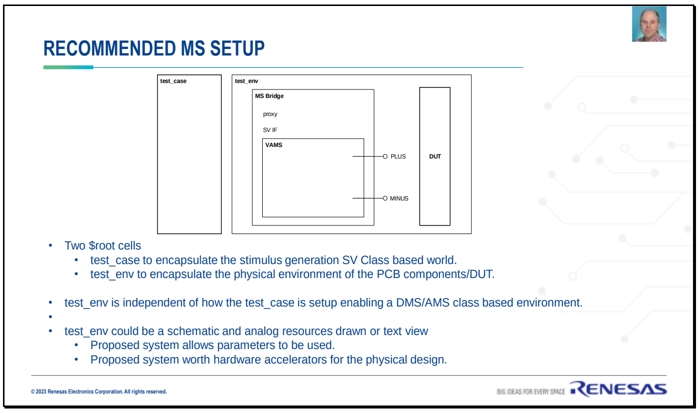
- uvm_*_printer print_real() uses %f formatting which truncates very small values
- Propose change to UVM implementation otherwise override print_real() in chosen printer

```
%f---------------------------------------------------
Name          Type      Size  Value
-----------------------------------------------------
cap_txn       cap_txn   -     @4931
  volts       real      64    0.000000
  amps        real      64    0.000000
  rseries     real      64    100.000000
  rparallel   real      64    1000000000000000.000000
  farads      real      64    0.000000
  henrys      real      64    0.000000
-----------------------------------------------------
```

```
%e---------------------------------
Name          Type      Size  Value
-----------------------------------
cap_txn       cap_txn   -     @4917
  volts       real      64    0
  amps        real      64    0
  rseries     real      64    100
  rparallel   real      64    1e+15
  farads      real      64    1e-09
  henrys      real      64    0
-----------------------------------
```

- Timestamps with limited precision in UVM-1.2 onwards
- compose_report_message() in uvm_report_server uses $time not $realtime
- Issue in UVM (MANTIS 0005807) workaround is to override report server

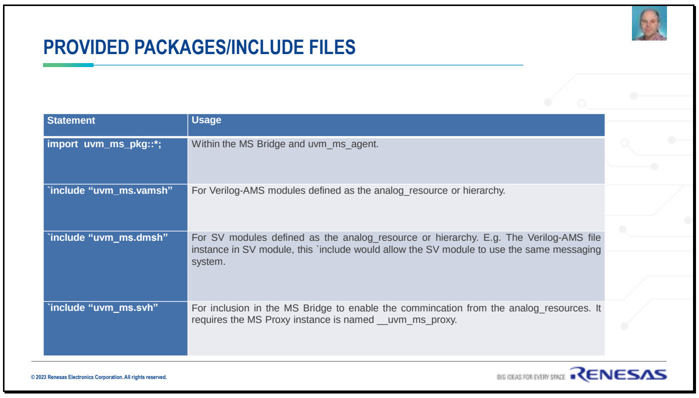
BIG IDEAS FOR EVERY SPACE RENESAS



RECOMMENDED MS SETUP

test_case
test_env
MS Bridge
proxy
SV IF
VAMS
PLUS
MINUS
DUT

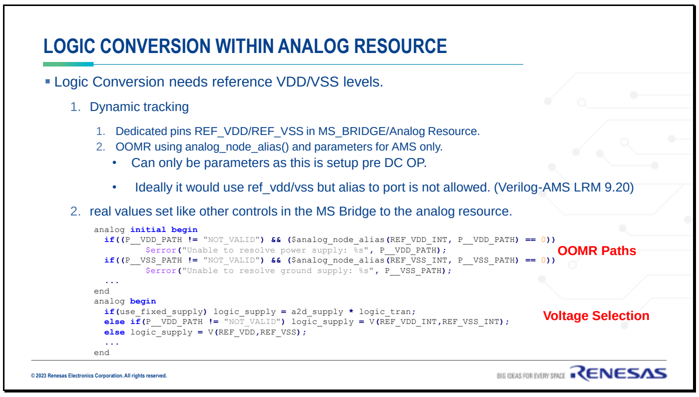
- Two $root cells
  - test_case to encapsulate the stimulus generation SV Class based world.
  - test_env to encapsulate the physical environment of the PCB components/DUT.
- test_env is independent of how the test_case is setup enabling a DMS/AMS class based environment.
-
- test_env could be a schematic and analog resources drawn or text view
  - Proposed system allows parameters to be used.
  - Proposed system worth hardware accelerators for the physical design.



PROVIDED PACKAGES/INCLUDE FILES

Statement	Usage
import uvm_ms_pkg::*;	Within the MS Bridge and uvm_ms_agent.
`include “uvm_ms.vamsh”	For Verilog-AMS modules defined as the analog_resource or hierarchy.
`include “uvm_ms.dmsh”	For SV modules defined as the analog_resource or hierarchy. E.g. The Verilog-AMS file instance in SV module, this `include would allow the SV module to use the same messaging system.
`include “uvm_ms.svh”	For inclusion in the MS Bridge to enable the commincation from the analog_resources. It requires the MS Proxy instance is named __uvm_ms_proxy.

Renesas.com



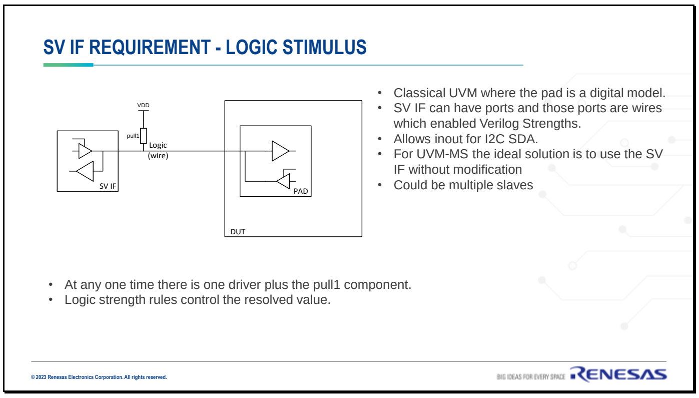
LOGIC CONVERSION WITHIN ANALOG RESOURCE

- Logic Conversion needs reference VDD/VSS levels.
  1. Dynamic tracking
     1. Dedicated pins REF_VDD/REF_VSS in MS_BRIDGE/Analog Resource.
     2. OOMR using analog_node_alias() and parameters for AMS only.
        - Can only be parameters as this is setup pre DC OP.
        - Ideally it would use ref_vdd/vss but alias to port is not allowed. (Verilog-AMS LRM 9.20)
  2. real values set like other controls in the MS Bridge to the analog resource.

```
analog initial begin
  if((P__VDD_PATH != "NOT_VALID") && ($analog_node_alias(REF_VDD_INT, P__VDD_PATH) == 0))
        $error("Unable to resolve power supply: %s", P__VDD_PATH);
  if((P__VSS_PATH != "NOT_VALID") && ($analog_node_alias(REF_VSS_INT, P__VSS_PATH) == 0))
        $error("Unable to resolve ground supply: %s", P__VSS_PATH);
  ...
end
analog begin
  if(use_fixed_supply) logic_supply = a2d_supply * logic_tran;
  else if(P__VDD_PATH != "NOT_VALID") logic_supply = V(REF_VDD_INT,REF_VSS_INT);
  else logic_supply = V(REF_VDD,REF_VSS);
  ...
end
```

OOMR Paths

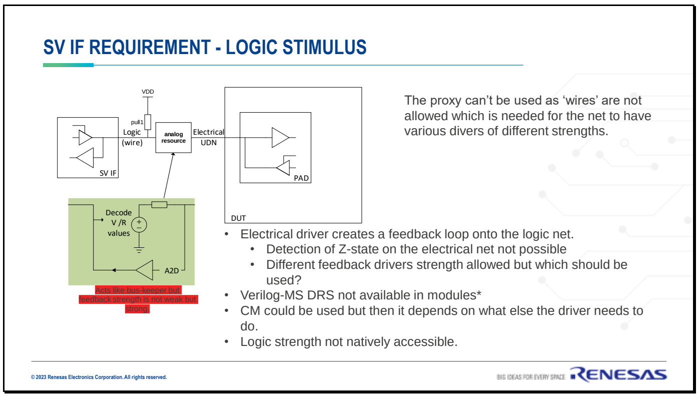
Voltage Selection



SV IF REQUIREMENT - LOGIC STIMULUS

- Classical UVM where the pad is a digital model.
- SV IF can have ports and those ports are wires which enabled Verilog Strengths.
- Allows inout for I2C SDA.
- For UVM-MS the ideal solution is to use the SV IF without modification
- Could be multiple slaves

- At any one time there is one driver plus the pull1 component.
- Logic strength rules control the resolved value.

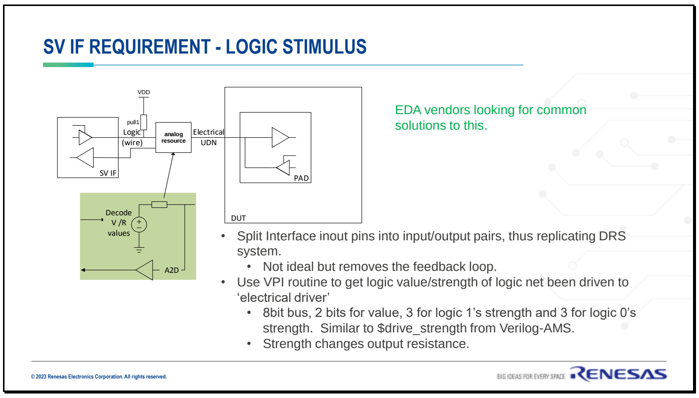


SV IF REQUIREMENT - LOGIC STIMULUS

Acts like bus-keeper but feedback strength is not weak but strong.

The proxy can't be used as 'wires' are not allowed which is needed for the net to have various divers of different strengths.

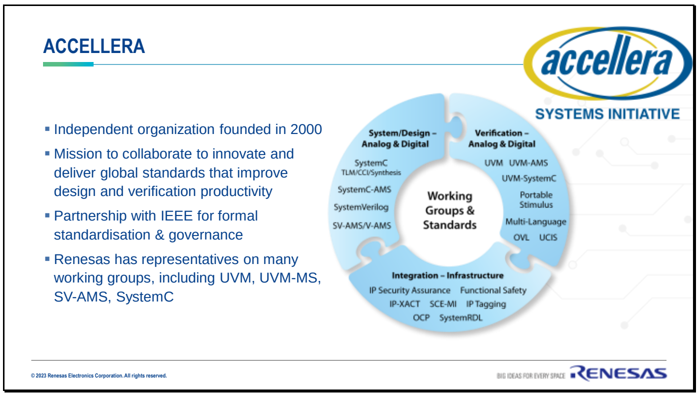
- Electrical driver creates a feedback loop onto the logic net.
  - Detection of Z-state on the electrical net not possible
  - Different feedback drivers strength allowed but which should be used?
- Verilog-MS DRS not available in modules*
- CM could be used but then it depends on what else the driver needs to do.
- Logic strength not natively accessible.



SV IF REQUIREMENT - LOGIC STIMULUS

EDA vendors looking for common solutions to this.

- Split Interface inout pins into input/output pairs, thus replicating DRS system.
  - Not ideal but removes the feedback loop.
- Use VPI routine to get logic value/strength of logic net been driven to 'electrical driver'
  - 8bit bus, 2 bits for value, 3 for logic 1's strength and 3 for logic 0's strength. Similar to $drive_strength from Verilog-AMS.
  - Strength changes output resistance.



ACCELLERA

- Independent organization founded in 2000
- Mission to collaborate to innovate and deliver global standards that improve design and verification productivity
- Partnership with IEEE for formal standardisation & governance
- Renesas has representatives on many working groups, including UVM, UVM-MS, SV-AMS, SystemC

Larry Lapides

Imperas Software Ltd
VP Worldwide Sales

A Modern Fable: The Lost Art of Processor Verification

Platinum Sponsor

Abstract

The open standard Instruction Set Architecture (ISA) of RISC-V offers new design flexibilities and opportunities, and is having a significant impact on the design side of many SoC projects. An optimized processor enables developers to unlock hidden value in performance, power savings, security, differentiated features, and an enduring market advantage.

While every SoC design team now has a free architecture license to build a custom RISC-V processor or extend an existing core with custom instructions, this also represents a surge in verification work and a step-change in verification complexity. With other ISAs, verification methodologies have largely been kept proprietary. Now within the RISC-V community, the art and science of processor verification is resurfacing. This represents a massive migration in verification responsibility, and the creation of a new verification ecosystem.

This talk outlines the various methodologies for RISC-V processor verification, which leverage established SoC verification technologies with UVM and SystemVerilog. The individual components of a step-compare methodology will be discussed, including reference model, verification IP, functional coverage and test generation. Detailed examples of successful, complex processor verification projects will be presented, including flows to support verification of complex events and architectures such as interrupts, Debug and privilege modes, multi-hart processors and multi-issue and out-of-order pipelines.

Biography

Larry is currently VP Worldwide Sales at Imperas Software Ltd., and previously ran worldwide sales at EDA companies including Verisity Design (the top performing IPO of 2001 in the U.S.). Larry has about 30 years in software tools and EDA, plus time spent in infrared sensors and systems engineering. Larry holds a BA in Physics from the University of California Berkeley, a MS in Applied and Engineering Physics from Cornell University and a MBA from Clark University where he was an Entrepreneur-in-Residence during Fall 2006, when he developed and taught the course on Entrepreneurial Communication and Influence

Notes

ImperasDV™ RISC-V Processor Verification Solutions

RISC-V is an open standard ISA (Instruction Set Architecture) that allows any developer to design and extend a custom processor, while remaining compatible with the growing ecosystem of supporting tools and software. The innovation and impact of RISC-V on the design side is driving new developments across all market segments and applications.

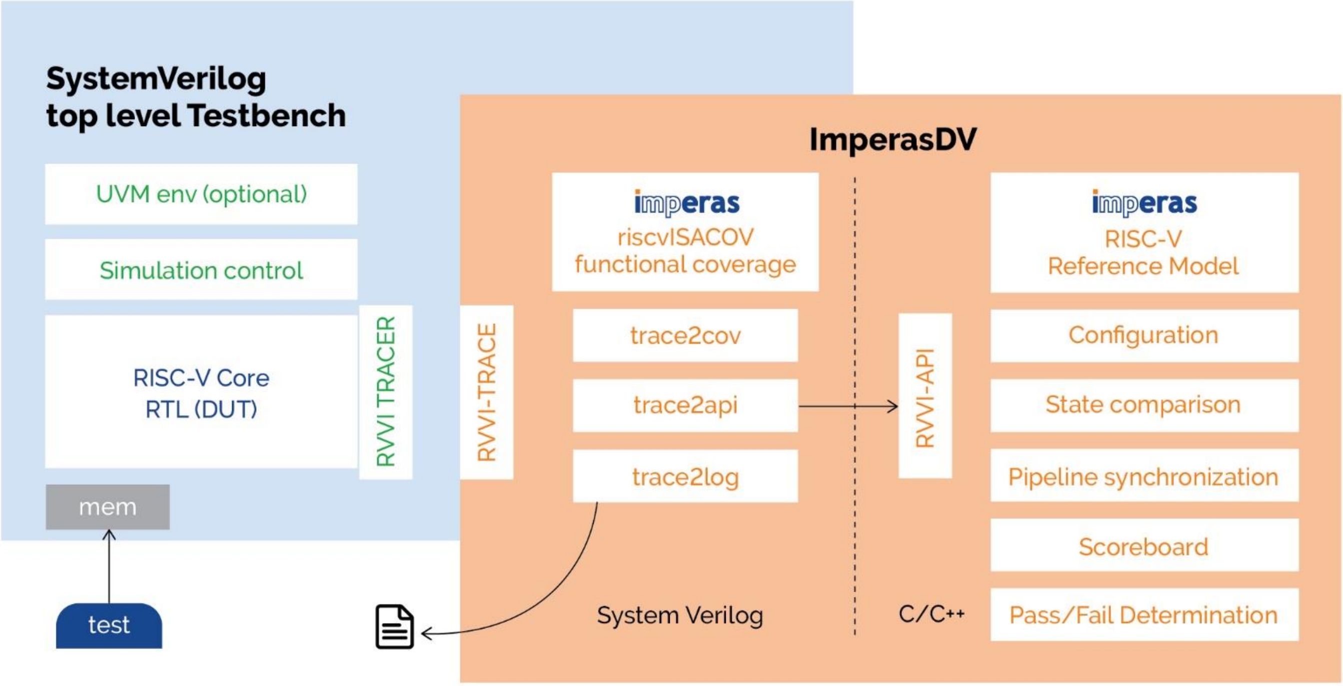
Now, with **ImperasDV**, developers have a dependable, reference model-based solution for verification that is compatible with the current UVM SystemVerilog methods for SoC verification.

www.imperas.com/imperasDV

ImperasDV and Imperas RISC-V processor verification technology is already in active use with many leading customers, some of which have working silicon prototypes and are now working on 2nd generation designs. These customers, partners, and users span the breadth of RISC-V adopters from open source to commercial; research to industrial; microcontrollers to high-performance computing.

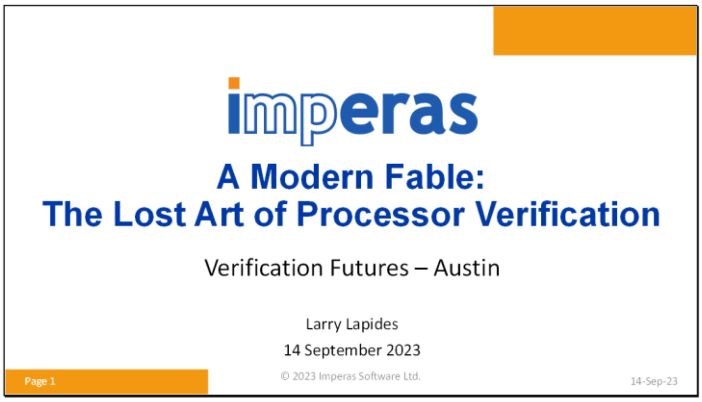
A select sample of these include **Codasip, Dolphin Design, EM Microelectronics (Swatch), Frontgrade Gaisler, Intrinsix, NSITEXE (Denso), Nvidia Networking (Mellanox), NXP, OpenHW Group, MIPS, Seagate Technology, Silicon Labs, Valtrix Systems**, and **Ventana Micro Systems**, plus many others yet to be made public.

RISC-V Processor Functional Verification with RVVI & ImperasDV



RISC-V Processor Design Verification →

- **Verification IP**
- **Test & Coverage suites**
- **RISC-V Reference model**



A Modern Fable: The Lost Art of Processor Verification

Verification Futures – Austin

Larry Lapides

14 September 2023

Agenda

- RISC-V and processor verification
- RISC-V processor models
- RISC-V processor verification methodology
- Processor verification success
- Summary

Agenda

- **RISC-V and processor verification**
- RISC-V processor models
- RISC-V processor verification methodology
- Processor verification success
- Summary

RISC-V Is Why We Are All Worried About Processor Verification

- RISC-V is taking over the processor world, except for x86
  - Yes, that includes Arm
- RISC-V processor customization means that every RISC-V developer needs to verify the RISC-V processor
- Lost art? Processor IP vendors guard their verification methodology and details more than the IP itself
  - With the verification flow, someone could reverse engineer a high quality processor
  - There are few public details about x86, Arm or Apple processor verification

RISC-V Freedom Enables Domain Specific Processing

- **Who**: RISC-V users include traditional semiconductor companies, and embedded systems companies now practicing vertical integration by developing their own SoCs
- **What**: RISC-V is an open instruction set architecture (ISA), it is not a processor implementation
- **Where**: RISC-V is growing in market segments where x86 (PCs, data centers) and Arm (mobile) architectures are not dominant
  - Small microcontrollers for SoC management, replacing proprietary cores
  - Verticals such as IoT and automotive
  - Horizontal markets such as security and AI/ML
  - Deep embedded applications
- **When**: RISC-V processors are now used in over 30% of SoCs
- **Why**: The freedom of the open ISA enables users to develop *differentiated* domain specific processors and processing systems

Keys to RISC-V SoC Success

1) Processor IP
   - Processor IP vendor
   - Open source IP
   - Build it yourself
2) Processor verification
3) Software porting, development, bring up, test

- All 3 areas need to account for the addition of custom features to the processor (*because everyone adds custom features to the processor*)

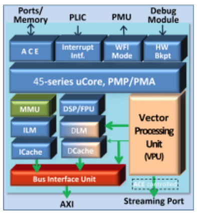
Keys to RISC-V SoC Success

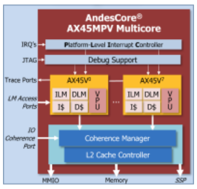
1) Processor IP
   - Processor IP vendor
   - Open source IP
   - Build it yourself
2) **Processor verification**
3) Software porting, development, bring up, test

- Users of all 3 types of processor IP need to account for the addition of custom features to the processor (*because everyone adds custom features to the processor*)
- Success in processor verification requires a high-quality model of the processor
- Success in processor verification requires innovative technologies and methodologies – ***the lost art of processor verification***

RISC-V Processor Complexity

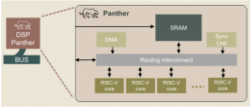
- RISC-V is a modular instruction set architecture
- Any extension (functional group of instructions, e.g. atomics, compressed, floating point, vector) can be added to the base processor
- Then add in interrupts, privilege modes, Debug mode, multi-hart (multi-core), etc. and it gets complex
- Then processor DV, tool chain development and other software development is needed



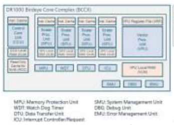


RISC-V Processing Subsystems

- Multi-processor subsystems are commonly being developed using RISC-V cores
- Application areas include DSP, AI/ML and packet processing
- This adds complexity to both the DV and software development tasks



Dolphin Design “Panther” DSP


DR1000 Birdseye Core Complex (BCCX)
Control Core Unit
Scalar Proc Unit
Vector Proc Unit
MPU
WDT
DTU
ICU
SMU
DBG
EMU
MPU: Memory Protection Unit
WDT: Watch Dog Timer
DTU: Data Transfer Unit
ICU: Interrupt Controller/Request
SMU: System Management Unit
DBG: Debug Unit
EMU: Error Management Unit

NSITEXE Data Flow Processor

Agenda

- RISC-V and processor verification
- **RISC-V processor models**
- RISC-V processor verification methodology
- Processor verification success
- Summary

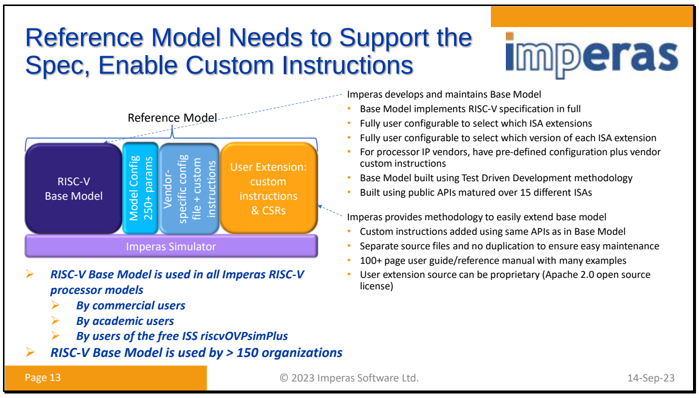
RISC-V Model Requirements

- Model the ISA, including all versions of the ratified spec, and stable unratified extensions
- Model other behavioral components, e.g. interrupt controllers
- Easily update and configure the model(s) for the next project
- User-extendable for custom instructions, registers, …
- Model actual processor IP, e.g. Andes, MIPS, NSITEXE, OpenHW, SiFive, SweRV, …
- Well-defined test process – for the model! – including coverage metrics
- Interface to other simulators, e.g. SystemVerilog (Xcelium), SystemC (Helium), Imperas virtual platform simulators
- Interface to software debug tools, e.g. GDB/Eclipse, Imperas MPD
- Interface to software analysis tools including access to processor internal state, etc.

- Most RISC-V ISSs can meet one or two of these requirements
- Imperas models and simulators were built to satisfy these requirements, and matured through usage on non-RISC-V ISAs over the last 15+ years

Imperas OVP RISC-V Fast Processor Models

- Use cases
  - Architecture analysis, including (especially) custom instructions
  - Software development, debug and test
  - Processor and SoC verification
- Existing Imperas Open Virtual Platforms (OVP) Fast Processor Models of …
  - Generic or envelope models of RV32/64 IMAFDCEVBHKPZ* M/S/U privilege modes
  - Models of processor IP vendors: Andes, MIPS, NSITEXE, OpenHW, SiFive, SweRV, …
  - Models for developers building their own processor
- Custom instructions easily added by user or by Imperas
  - New instructions are added in a side file so as not to perturb the verified model
  - Custom instructions are analyzed for effectiveness
- Models are built using Test Driven Development (TDD) methodology
  - Tests are built at the same time as features are added
  - Continuous Integration (CI) test flow used
  - > 15,000 tests for models + simulator
  - Additional testing by processor IP vendors to validate models



Reference Model Needs to Support the Spec, Enable Custom Instructions

Reference Model
RISC-V Base Model
Model Config 250+ params
Vendor-specific config file + custom instructions
User Extension: custom instructions & CSRs
Imperas Simulator

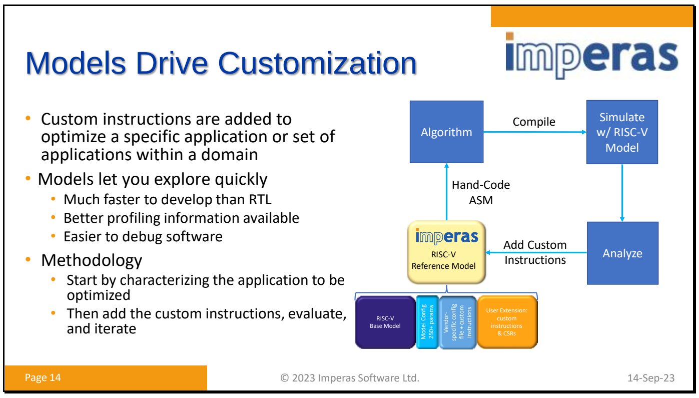
Imperas develops and maintains Base Model

- Base Model implements RISC-V specification in full
- Fully user configurable to select which ISA extensions
- Fully user configurable to select which version of each ISA extension
- For processor IP vendors, have pre-defined configuration plus vendor custom instructions
- Base Model built using Test Driven Development methodology
- Built using public APIs matured over 15 different ISAs

Imperas provides methodology to easily extend base model

- Custom instructions added using same APIs as in Base Model
- Separate source files and no duplication to ensure easy maintenance
- 100+ page user guide/reference manual with many examples
- User extension source can be proprietary (Apache 2.0 open source license)

- ***RISC-V Base Model is used in all Imperas RISC-V processor models***
  - ***By commercial users***
  - ***By academic users***
  - ***By users of the free ISS riscvOVPsimPlus***
- ***RISC-V Base Model is used by > 150 organizations***

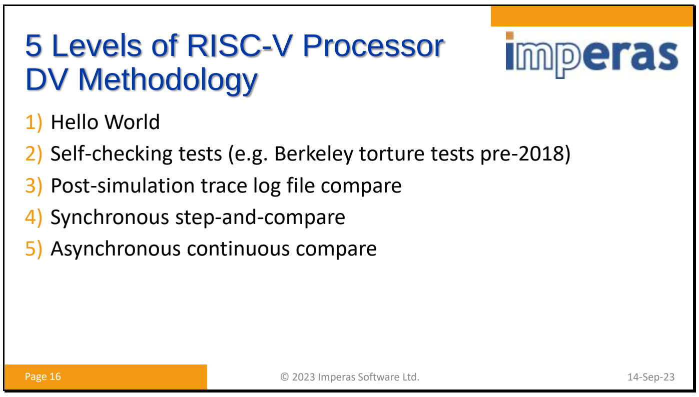


Models Drive Customization

- Custom instructions are added to optimize a specific application or set of applications within a domain
- Models let you explore quickly
  - Much faster to develop than RTL
  - Better profiling information available
  - Easier to debug software
- Methodology
  - Start by characterizing the application to be optimized
  - Then add the custom instructions, evaluate, and iterate

Agenda

- RISC-V and processor verification
- RISC-V processor models
- **RISC-V processor verification methodology**
- Processor verification success
- Summary



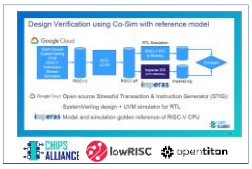
5 Levels of RISC-V Processor DV Methodology

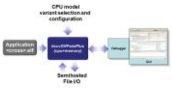
1) Hello World
2) Self-checking tests (e.g. Berkeley torture tests pre-2018)
3) Post-simulation trace log file compare
4) Synchronous step-and-compare
5) Asynchronous continuous compare

5 Levels of RISC-V Processor DV Methodology

1) Hello World
2) Self-checking tests (e.g. Berkeley torture tests pre-2018)
3) Post-simulation trace log file compare
4) Synchronous step-and-compare
5) Asynchronous continuous compare

3) Post-Simulation Trace Log File Compare (Entry Level DV)

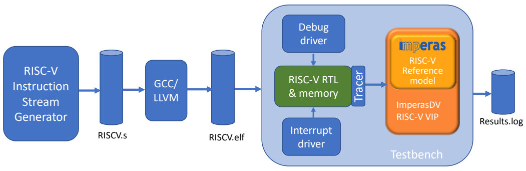

Design Verification using Co-Sim with reference model
Google Cloud
RTL Simulation
Compare
imperas
Open source Stressful Transaction & Instruction Generator (STIG)
SystemVerilog design + UVM simulator for RTL
imperas Model and simulation golden reference of RISC-V CPU
CHIPS ALLIANCE
lowRISC
opentitan



Imperas riscvOVPsimPlus Reference Simulator

- Process
  - use random generator (ISG) to create tests
  - during simulation of ISS write trace log file
  - during simulation of RTL write trace log file
  - at the end of both runs, run logs through compare program to see differences / failures
- ISS: riscvOVPsimPlus includes Trace and GDB interface
  - Free ISS: https://www.ovpworld.org/riscvOVPsimPlus
- ISG: riscv-dv from Google Cloud / Chips Alliance
  - Free ISG: https://github.com/google/riscv-dv

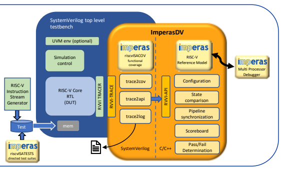
5) Async Continuous Compare (Highest Quality DV Methodology)



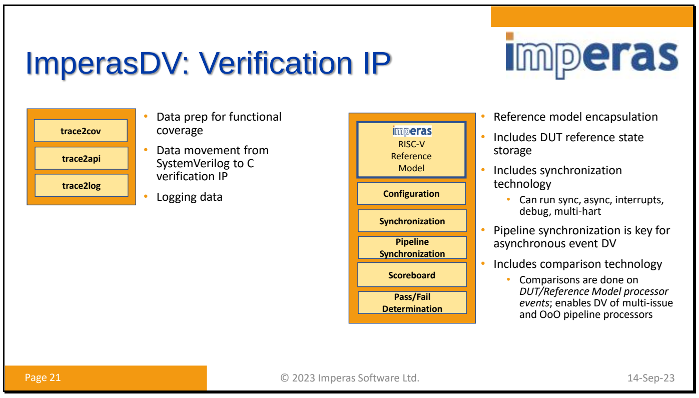
- Asynchronous events are driven into the DUT
- Tracer informs reference model about async events
- Verification IP handles async events, scoreboarding, comparison, pass/fail
- ***Asynchronous continuous compare methodology is needed to support features such as interrupts, privilege modes, Debug mode, multi-hart, multi-issue and OoO pipeline, …***

ImperasDV Components

- Reference model needed for comparison of correct behavior
- Verification IP provides ease of use, saves time and resources
- RVVI standard provides communication between test bench and reference model subsystem
- riscvISACOV: functional coverage modules
- Test suites: riscvISATESTS, directed test suites for difficult extensions
- MultiProcessor Debugger (MPD) enables RTL-reference model co-debug



- Feature selection and design choices require serious consideration due to implications of every decision
  - Every addition dramatically compounds verification complexity
  - Adds schedule, resources, quality costs == big risks
- Before 2021, no off-the-shelf toolkit/products available for DV of processors … then came ImperasDV
- ***ImperasDV, with async continuous compare methodology, is needed to support features such as interrupts, privilege modes, Debug mode, multi-hart, multi-issue and OoO pipeline, …***



ImperasDV: Verification IP

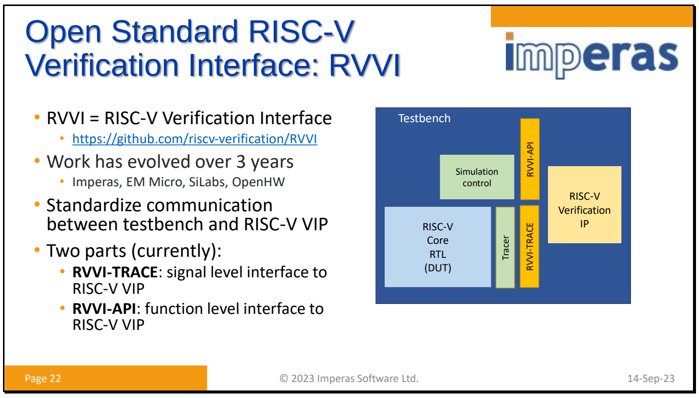
imperas

trace2cov
trace2api
trace2log

- Data prep for functional coverage
- Data movement from SystemVerilog to C verification IP
- Logging data

imperas
RISC-V Reference Model
Configuration
Synchronization
Pipeline Synchronization
Scoreboard
Pass/Fail Determination

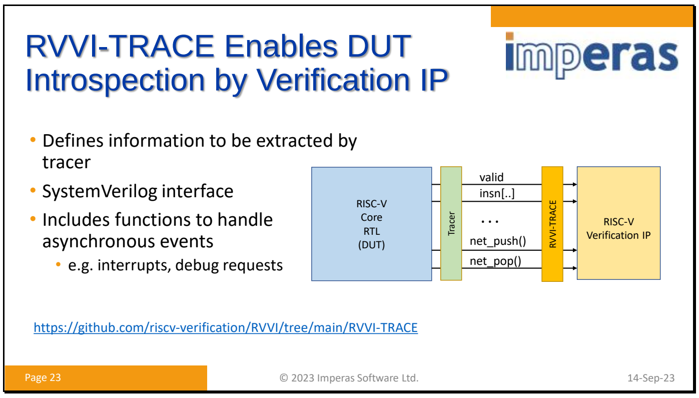
- Reference model encapsulation
- Includes DUT reference state storage
- Includes synchronization technology
  - Can run sync, async, interrupts, debug, multi-hart
- Pipeline synchronization is key for asynchronous event DV
- Includes comparison technology
  - Comparisons are done on *DUT/Reference Model processor events*; enables DV of multi-issue and OoO pipeline processors



Open Standard RISC-V Verification Interface: RVVI

imperas

- RVVI = RISC-V Verification Interface
  - https://github.com/riscv-verification/RVVI
- Work has evolved over 3 years
  - Imperas, EM Micro, SiLabs, OpenHW
- Standardize communication between testbench and RISC-V VIP
- Two parts (currently):
  - **RVVI-TRACE**: signal level interface to RISC-V VIP
  - **RVVI-API**: function level interface to RISC-V VIP



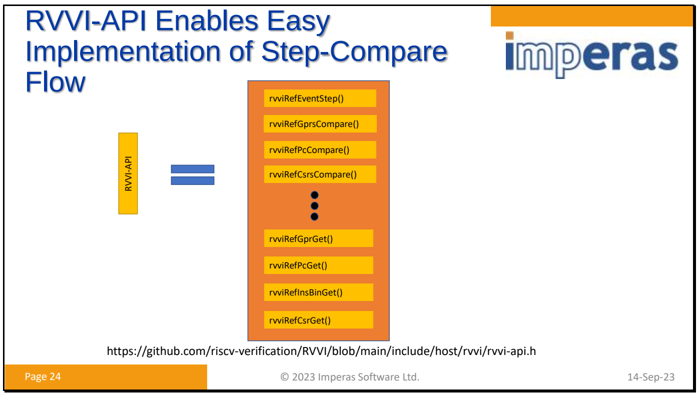
RVVI-TRACE Enables DUT Introspection by Verification IP

imperas

- Defines information to be extracted by tracer
- SystemVerilog interface
- Includes functions to handle asynchronous events
  - e.g. interrupts, debug requests

RISC-V Core RTL (DUT)
Tracer
valid
insn[..]
. . .
net_push()
net_pop()
RVVI-TRACE
RISC-V Verification IP

https://github.com/riscv-verification/RVVI/tree/main/RVVI-TRACE



RVVI-API Enables Easy Implementation of Step-Compare Flow

imperas

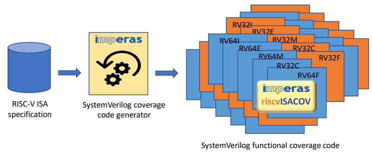
RVVI-API
rvviRefEventStep()
rvviRefGprsCompare()
rvviRefPcCompare()
rvviRefCsrsCompare()
rvviRefGprGet()
rvviRefPcGet()
rvviRefInsBinGet()
rvviRefCsrGet()

https://github.com/riscv-verification/RVVI/blob/main/include/host/rvvi/rvvi-api.h

Functional Coverage of RISC-V Instructions: Scope

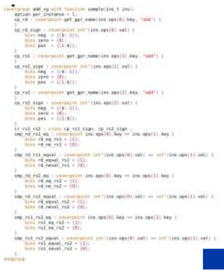
- There are many different instructions in the RV64 extensions:
  - Integer: 56, Maths: 13, Compressed: 30, FP-Single: 30, FP-Double: 32
  - Vector: 356, Bitmanip: 47 Krypto-scalar: 85
  - P-DSP: 318
  - For RV64 that is 967 instructions…
- Each instruction needs SystemVerilog covergroups and coverpoints
  - 10-40 lines of SystemVerilog for each instruction
- 10,000-40,000++ lines of code to be written
  - Not design or core specific

riscvISACOV Is Automatically Generated SystemVerilog Functional Coverage

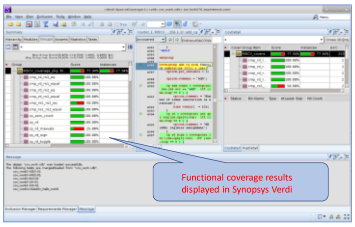


- riscvISACOV provides functional coverage of Instructions and operands
- Roadmap includes CSRs and data hazards
- Imperas tools can automatically generate functional coverage code for custom instructions

Functional Coverage is the Key Verification Metric



```
covergroup add_cg with function sample(ins_t ins);
    option.per_instance = 1;
    cp_rd : coverpoint get_gpr_name(ins.ops[0].key, "add") {
    }
    cp_rd_sign : coverpoint int'(ins.ops[0].val) {
        bins neg  = {[$:-1]};
        bins zero = {0};
        bins pos  = {[1:$]};
    }
    cp_rs1 : coverpoint get_gpr_name(ins.ops[1].key, "add") {
    }
    cp_rs1_sign : coverpoint int'(ins.ops[1].val) {
        bins neg  = {[$:-1]};
        bins zero = {0};
        bins pos  = {[1:$]};
    }
    cp_rs2 : coverpoint get_gpr_name(ins.ops[2].key, "add") {
    }
    cp_rs2_sign : coverpoint int'(ins.ops[2].val) {
        bins neg  = {[$:-1]};
        bins zero = {0};
        bins pos  = {[1:$]};
    }
    cr_rs1_rs2 : cross cp_rs1_sign, cp_rs2_sign ;
    cmp_rd_rs1_eq : coverpoint ins.ops[0].key == ins.ops[1].key {
        bins rd_eq_rs1 = {1};
        bins rd_ne_rs1 = {0};
    }
    cmp_rd_rs1_eqval : coverpoint int'(ins.ops[0].val) == int'(ins.ops[1].val) {
        bins rd_eqval_rs1 = {1};
        bins rd_neval_rs1 = {0};
    }
    cmp_rd_rs2_eq : coverpoint ins.ops[0].key == ins.ops[1].key {
        bins rd_eq_rs2 = {1};
        bins rd_ne_rs2 = {0};
    }
    cmp_rd_rs2_eqval : coverpoint int'(ins.ops[0].val) == int'(ins.ops[1].val) {
        bins rd_eqval_rs2 = {1};
        bins rd_neval_rs2 = {0};
    }
    cmp_rs1_rs2_eq : coverpoint ins.ops[0].key == ins.ops[1].key {
        bins rs1_eq_rs2 = {1};
        bins rs1_ne_rs2 = {0};
    }
    cmp_rs1_rs2_eqval : coverpoint int'(ins.ops[0].val) == int'(ins.ops[1].val) {
        bins rs1_eqval_rs2 = {1};
        bins rs1_neval_rs2 = {0};
    }
endgroup
```



Functional coverage results displayed in Synopsys Verdi

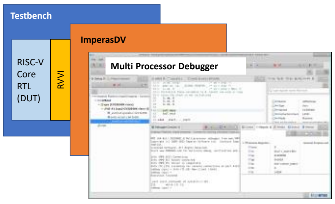
riscvISACOV (and ImperasDV) works with any SystemVerilog simulator

Test Stimuli

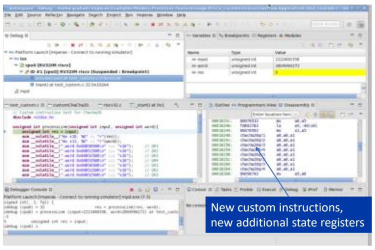
- Instruction Stream Generator (ISG) and/or directed tests
- ISG generates test programs using constrained random approach
  - Most often obtain the ISG:
    - Commercial such as Valtrix STING
    - Open source such as Google riscv-dv
  - Require toolchains like GCC, LLVM for assemblers, linkers
  - Require functional coverage so that you know what you've tested!
- Directed tests
  - Imperas have developed a directed RISC-V test generator, instruction coverage verification IP and a mutating fault simulator (for test qualification) to provide high quality test suites
    - The generated tests suites are targeting architectural compatibility as defined in the RVIA architectural test working group coverage requirements
  - Free Imperas architectural validation test suites (50+), including RV32/64 I, M, C, F, D, B, K, V, P
    - https://github.com/riscv-ovpsim/imperas-riscv-tests
  - Imperas commercial directed test suites for vector extension, protected memory components
    - Can support any RISC-V vector or PMP configuration; the user selects the configuration and Imperas generates the test suite

ImperasDV: Debug with MPD

- Imperas MPD is an Eclipse based debug tool
- Can debug using source line or instruction level
- See new custom instructions and any new additional state registers
- Break at the first mismatch, debug SW and RTL concurrently



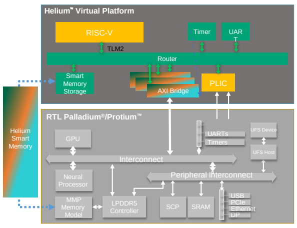
Software Debug and Analysis Tools Automatically Work With the Custom Instructions


New custom instructions, new additional state registers

New custom instructions in trace disassembly

Hybrid Simulation-Emulation for HW-SW Co-Verification

- It takes more than just the RTL simulator for comprehensive processor verification
- An additional tool the the hardware emulator, e.g. Cadence Palladium
- The interface to Palladium is via the Cadence Helium SystemC simulator

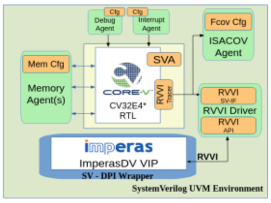


Agenda

- RISC-V and processor verification
- RISC-V processor models
- RISC-V processor verification methodology
- **Processor verification success**
- Summary

RISC-V Processor DV Results

- OpenHW Core-V-Verif
  - OpenHW users now on 3rd generation of Core-V-Verif DV flow
  - CV32E40P successfully taped out
  - CV32E40Pv2, CV32E40S, CV32E40X, CV32E20 using this flow today; all expected to complete DV this year
- Other successful DV projects using Imperas include Codasip, MIPS, Nagra, NSITEXE, Nvidia Networking, …



Case Study: Wally RISC-V Core

- Configurable core:
  - RV32I, RV32E, RV64I, RV64E
  - A, C, F, D, M extensions, privileged modes, CSRs
  - MMU/TLB virtual memory, caches
- Developed at Harvey Mudd College / Oklahoma State University
  - Focus: high quality core for processor architecture education
  - Now in OpenHW as CORE-V Wally (https://github.com/openhwgroup/cvw)
- Status in January 2023 – before starting to use RVVI + ImperasDV for verification:
  - Passing all RISC-V International compliance tests, Imperas compatibility tests
    - Using Compliance Level post-simulation signature file compare
  - Boots Linux

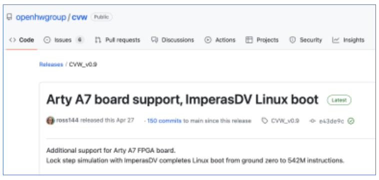
Wally + RVVI – Status (July 2023)

- RVVI Tracer + testbench integration: 3 days of effort
- Results:
  - 20+ bugs found in simulation almost immediately using ImperasDV and riscv-dv
  - Reached Linux prompt with continuous checking: 2 days of simulation
  - One bug found just *after* the Linux command prompt (!)
  - Functional coverage achieved by booting Linux: covergroups 37% (bins 3%)

JT0

- Future work:
  - Boot Linux with co-sim using hardware assisted verification
  - Achieve 100% functional coverage using constrained-random tests

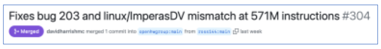
JT1



Arty A7 board support, ImperasDV Linux boot

ross144 released this Apr 27 · 150 commits to main since this release CVW_v0.9 e43de9c

Additional support for Arty A7 FPGA board.
Lock step simulation with ImperasDV completes Linux boot from ground zero to 542M instructions.



Fixes bug 203 and linux/ImperasDV mismatch at 571M instructions #304

davidharrishmc merged 1 commit into openhwgroup:main from ross144:main last week

Agenda

- RISC-V and processor verification
- RISC-V processor models
- RISC-V processor verification methodology
- Processor verification success
- **Summary**

Processor Verification: The Key to the RISC-V Castle

- Asynchronous step-compare methodology
- High quality reference models
- Verification IP
- Verification standards
- Verification metrics

- ImperasDV

Thank you

Larry Lapides
LarryL@imperas.com

Adnan Hamid

Breker Verification Systems

Founder and CTO

Advanced RISC-V Verification Technique Learnings for SoC Validation

Gold Sponsor

Abstract

The verification of application-level RISC-V cores require specialized techniques and approaches previously the purview of Arm, Intel and other processor companies. The open and customizable RISC-V cores have led to many new processor development teams with unique microarchitectural approaches that require extensive verification.

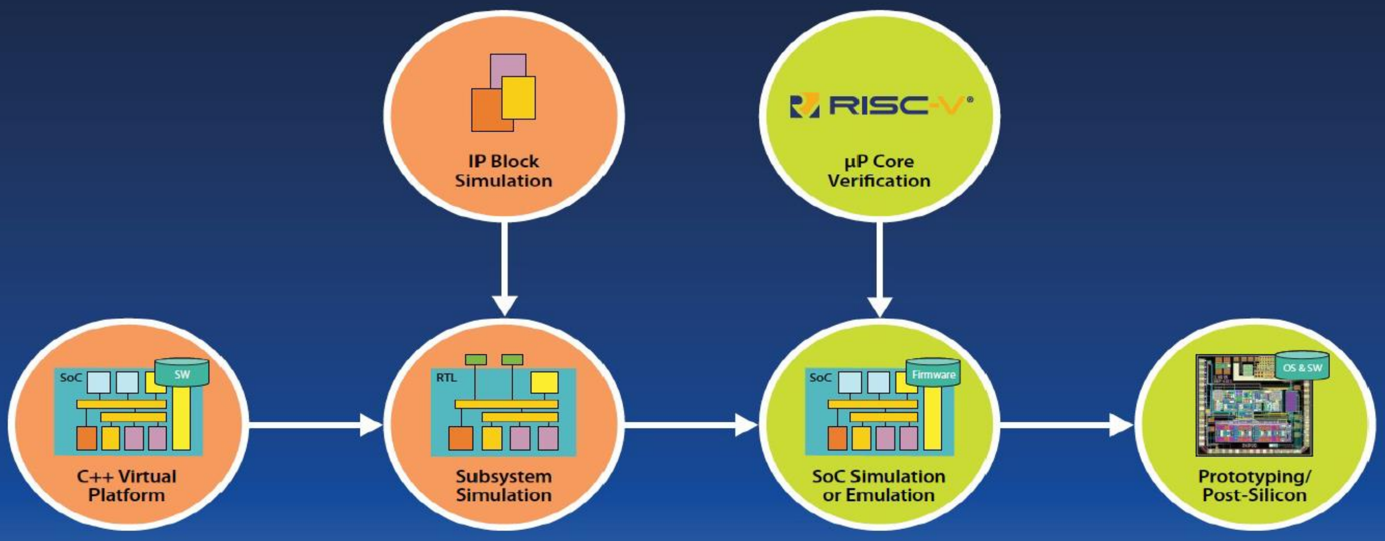
Breker has found that a key aspect of RISC-V core verification involves its smooth operation within the larger system. For example, load-store anomalies, asynchronous interrupt mechanisms, and security protocols are just a few of many issues that must be fully analysed. In developing new test approaches for these and other scenarios, their application in more general System-on-Chips has become apparent, and indeed these methods can track complex system corner cases that will never be detected simply by running real workloads or benchmarks.This presentation will describe many techniques useful for RISC-V core verification, and also how they may be applied to the broader SoC at large for high coverage verification.

Biography

Adnan is the founder and CTO of Breker and the inventor of its core technology. Noted as the father of Portable Stimulus, he has over 20 years of experience in functional verification automation, much of it spent working in this domain.

Prior to Breker, he managed AMD’s System Logic Division, and also led their verification team to create the first test case generator providing 100% coverage for an x86-class microprocessor. In addition, Adnan spent several years at Cadence Design Systems and served as the subject matter expert in system-level verification, developing solutions for Texas Instruments, Siemens/Infineon, Motorola/Freescale, and General Motors.

Adnan holds twelve patents in test case generation and synthesis. He received BS degrees in Electrical Engineering and Computer Science from Princeton University, and an
MBA from the University of Texas at Austin.



SoC Integrity
SystemVIP
High Coverage,
Automated Apps

IP, End-to-End
Test Content
SystemUVM Scalable,
Portable Test Content

RISC-V Core
& SoC Integrity
Comprehensive Toolkits
for Stringent Testing

System & SW
Validation
Firmware Test,
Performance Profiling

RISC-V Core SystemVIP

Random Instructions	Do instructions yield correct results
Register/Register Hazards	Pipeline perturbations dues to register conflicts
Load/Store Integrity	Memory conflict patterns
Conditionals and Branches	Pipeline perturbations from synchronous PC change
Exceptions	Jumping to and returning from ISR
Asynchronous Interrupts	Pipeline perturbations from asynchronous PC change
Privilege Level Switching	Context switching
Core Security	Register and Memory protection by privilege level
Core Paging/MMU	Memory virtualization and TLB operation
Sleep/Wakeup	State retention across WFI
Voltage/Freq Scaling	Operation at different clock ratios
Core Coherency	Caches, evictions and snoops

SoC SystemVIP

Random Memory Tests	Test Cores/Fabrics/Memory controllers
Random Register Tests	Read/write test to all uncore registers
System Interrupts	Randomized interrupts through CLINT
Multi-core Execution	Concurrent operations on fabric and memory
Memory Ordering	For weakly order memory protocols
Atomic Operation	Across all memory types
System Coherency	Cover all cache transitions, evictions, snoops
System Paging/IOMMU	System memory virtualization
System Security	Register and Memory protection across system
Power Management	System wide sleep/wakeup and voltage/freq scaling
Packet Generation	Generating networking packets for I/O testing
Interface Testing	Analyzing coherent interfaces including CXL & UCIe
SoC Profiling	Layering concurrent tests to check operation under stress
Firmware-First	Executing SW on block or sub-system without processor

BREKER™

Automated Test Synthesis

Fast test availability

Quality SystemVIP / easy composition

Ultrahigh coverage

Auto bug tracking in complex scenarios

Portable & Reusable

Same tests across platforms and projects

www.brekersystems.com | info@brekersystems.com

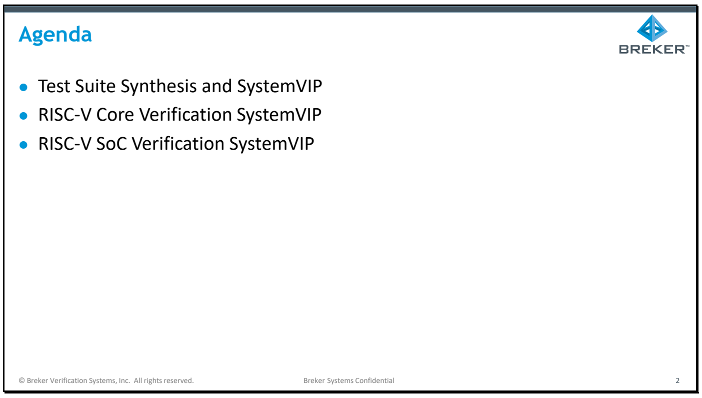
BREKER™ THE LEADER IN PORTABLE STIMULUS

Advanced RISC-V Verification Technique Learnings for SoC Validation

Using Breker SystemVIP for RISC-V System Ready

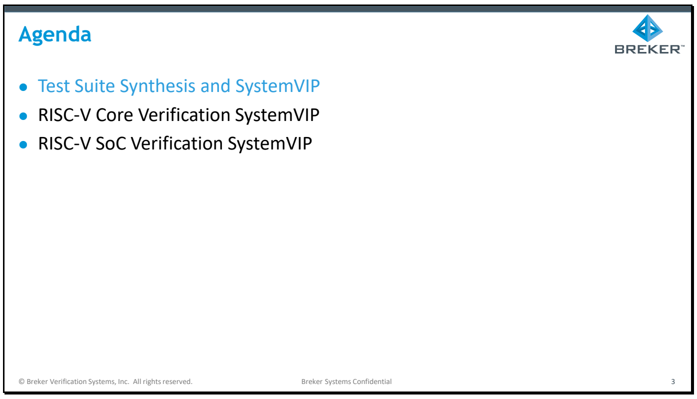
Adnan Hamid, CTO, Breker Verification Systems

Verification Futures Austin 2023



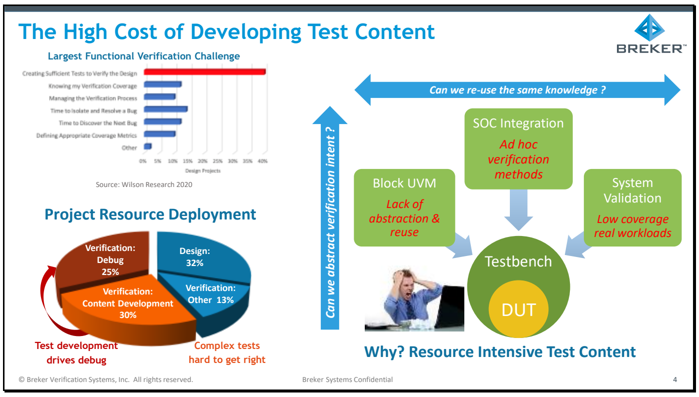
Agenda

- Test Suite Synthesis and SystemVIP
- RISC-V Core Verification SystemVIP
- RISC-V SoC Verification SystemVIP



Agenda

- Test Suite Synthesis and SystemVIP
- RISC-V Core Verification SystemVIP
- RISC-V SoC Verification SystemVIP



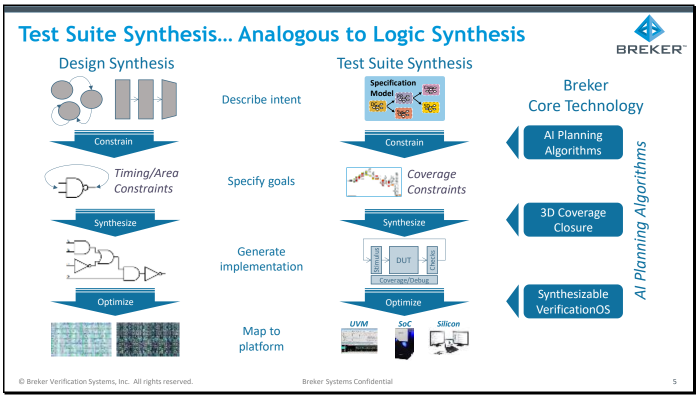
The High Cost of Developing Test Content

Source: Wilson Research 2020

Test development drives debug

Complex tests hard to get right

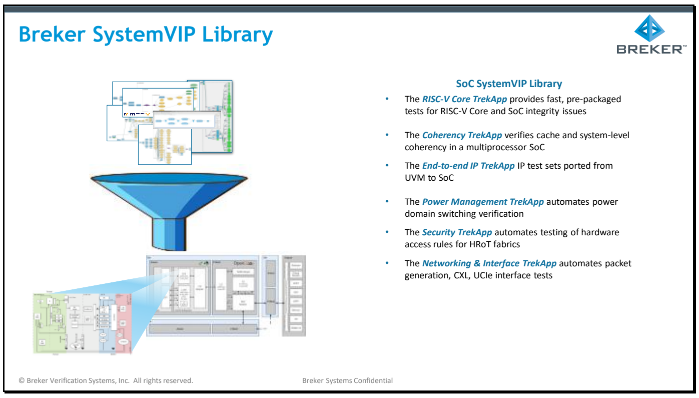
Why? Resource Intensive Test Content



Test Suite Synthesis… Analogous to Logic Synthesis

Design Synthesis

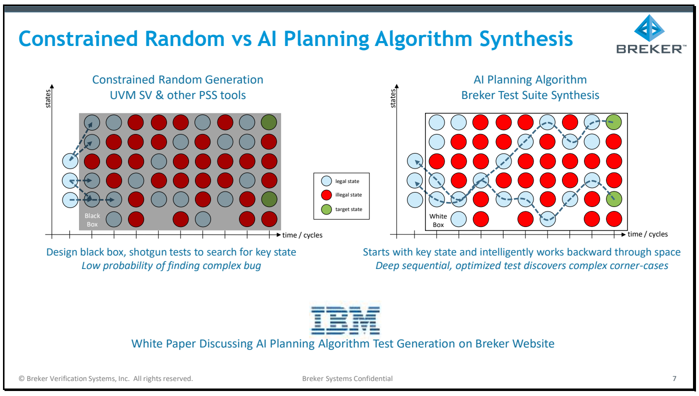
Test Suite Synthesis



Breker SystemVIP Library

SoC SystemVIP Library

- The ***RISC-V Core TrekApp*** provides fast, pre-packaged tests for RISC-V Core and SoC integrity issues
- The ***Coherency TrekApp*** verifies cache and system-level coherency in a multiprocessor SoC
- The ***End-to-end IP TrekApp*** IP test sets ported from UVM to SoC
- The ***Power Management TrekApp*** automates power domain switching verification
- The ***Security TrekApp*** automates testing of hardware access rules for HRoT fabrics
- The ***Networking & Interface TrekApp*** automates packet generation, CXL, UCIe interface tests



Constrained Random vs AI Planning Algorithm Synthesis

Constrained Random Generation
UVM SV & other PSS tools

Design black box, shotgun tests to search for key state
Low probability of finding complex bug

AI Planning Algorithm
Breker Test Suite Synthesis

Starts with key state and intelligently works backward through space
Deep sequential, optimized test discovers complex corner-cases

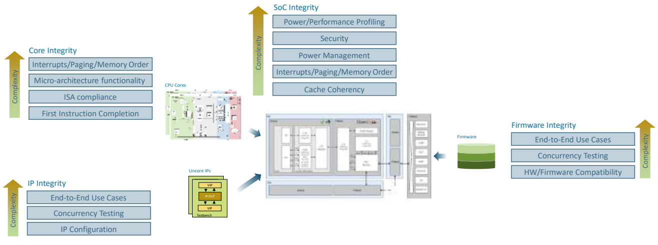
White Paper Discussing AI Planning Algorithm Test Generation on Breker Website

A Look At RISC-V

- Open Instruction Set Architecture (ISA) creating a discontinuity in the market
- Appears to be gaining significant traction in multiple applications
- Significant verification challenges
  - ○ Arm spends $150M per year on 10^{15} verification cycles per core
  - ○ Hard for RISC-V development group to achieve this same quality
  - ○ Lots of applications expands verification requirements
  - ○ Requires automation, reuse and other new thinking

RISC-V®

RISC-V Verification & Validation Tasks

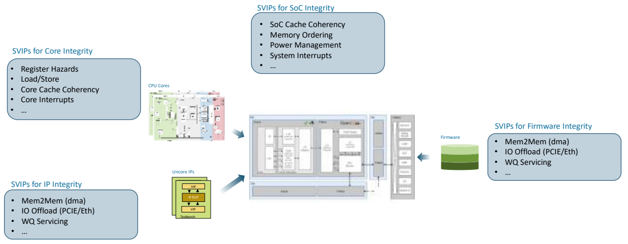

SoC Integrity
- Power/Performance Profiling
- Security
- Power Management
- Interrupts/Paging/Memory Order
- Cache Coherency

Core Integrity
- Interrupts/Paging/Memory Order
- Micro-architecture functionality
- ISA compliance
- First Instruction Completion

Firmware Integrity
- End-to-End Use Cases
- Concurrency Testing
- HW/Firmware Compatibility

IP Integrity
- End-to-End Use Cases
- Concurrency Testing
- IP Configuration

Breker RISC-V SystemVIP Portfolio

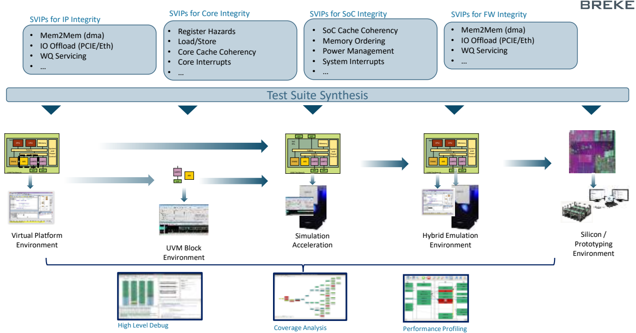

SVIPs for SoC Integrity
- SoC Cache Coherency
- Memory Ordering
- Power Management
- System Interrupts
- …

SVIPs for Core Integrity
- Register Hazards
- Load/Store
- Core Cache Coherency
- Core Interrupts
- …

SVIPs for Firmware Integrity
- Mem2Mem (dma)
- IO Offload (PCIE/Eth)
- WQ Servicing
- …

SVIPs for IP Integrity
- Mem2Mem (dma)
- IO Offload (PCIE/Eth)
- WQ Servicing
- …

Single Source of Truth for all stages of Verification & Validation


SVIPs for IP Integrity
- Mem2Mem (dma)
- IO Offload (PCIE/Eth)
- WQ Servicing
- …

SVIPs for Core Integrity
- Register Hazards
- Load/Store
- Core Cache Coherency
- Core Interrupts
- …

SVIPs for SoC Integrity
- SoC Cache Coherency
- Memory Ordering
- Power Management
- System Interrupts
- …

SVIPs for FW Integrity
- Mem2Mem (dma)
- IO Offload (PCIE/Eth)
- WQ Servicing
- …

Test Suite Synthesis

Virtual Platform Environment

UVM Block Environment

Simulation Acceleration

Hybrid Emulation Environment

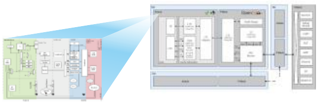
Silicon / Prototyping Environment

High Level Debug

Coverage Analysis

Performance Profiling

Different Challenges for Core vs SoC Verification



RISC-V Core Verification Challenges

Random Instructions	Do instructions yield correct results
Register/Register Hazards	Pipeline perturbations dues to register conflicts
Load/Store Integrity	Memory conflict patterns
Conditionals and Branches	Pipeline perturbations from synchronous PC change
Exceptions	Jumping to and returning from ISR
Asynchronous Interrupts	Pipeline perturbations from asynchronous PC change
Privilege Level Switching	Context switching
Core Security	Register and Memory protection by privilege level
Core Paging/MMU	Memory virtualization and TLB operation
Sleep/Wakeup	State retention across WFI
Voltage/Freq Scaling	Operation at different clock ratios
Core Coherency	Caches, evictions and snoops

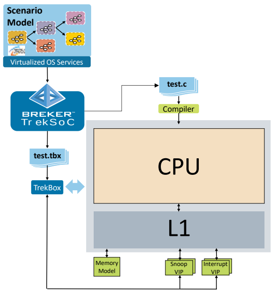
RISC-V SoC Verification Challenges

System Coherency	Cover all cache transitions, evictions, snoops
System Paging/IOMMU	System memory virtualization
System Security	Register and Memory protection across system
Power Management	System wide sleep/wakeup and voltage/freq scaling
Packet Generation	Generating networking packets for I/O testing
Interface Testing	Analyzing coherent interfaces including CXL & UCIe
Random Memory Tests	Test Cores/Fabrics/Memory controllers across DDR, OCRAM, FLASH etc
Random Register Tests	Read/write test to all uncore registers
System Interrupts	Randomized interrupts through CLINT
Multi-core Execution	Concurrent operations on fabric and memory
Memory Ordering	For weakly order memory protocols
Atomic Operation	Across all memory types

Agenda

- Test Suite Synthesis and SystemVIP
- RISC-V Core Verification SystemVIP
- RISC-V SoC Verification SystemVIP

RISC-V Core Testbench Integration



Scenario Model

Virtualized OS Services

test.c

Compiler

test.tbx

TrekBox

CPU

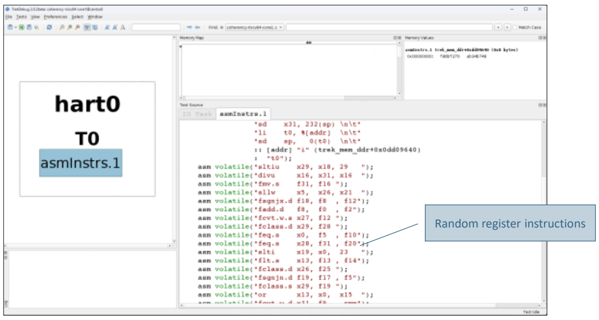
L1

Memory Model

Snoop VIP

Interrupt VIP

RV64 Core Instruction Generation



hart0

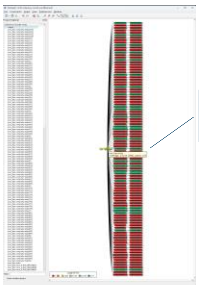
T0

asmInstrs.1

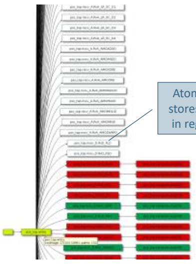
```
                    "sd    x31, 232(sp) \n\t"
                    "li    t0, %[addr]  \n\t"
                    "sd    sp,   0(t0)  \n\t"
                    :: [addr] "i" (trek_mem_ddr+0x0dd09640)
                    :  "t0");
    asm volatile("sltiu    x29, x18, 29   ");
    asm volatile("divu     x16, x31, x16  ");
    asm volatile("fmv.s    f31, f16 ");
    asm volatile("sllw     x5,  x26, x21  ");
    asm volatile("fsgnjx.d f18, f8  , f12");
    asm volatile("fadd.d   f8,  f0  , f2");
    asm volatile("fcvt.w.s x27, f12 ");
    asm volatile("fclass.d x29, f28 ");
    asm volatile("feq.s    x0,  f5  , f10");
    asm volatile("feq.s    x28, f31 , f20");
    asm volatile("slti     x19, x0,  23   ");
    asm volatile("flt.s    x13, f13 , f14");
    asm volatile("fclass.d x26, f25 ");
    asm volatile("fsgnjn.d f19, f17 , f5");
    asm volatile("fclass.s x29, f19 ");
    asm volatile("or       x13, x0,  x15  ");
```

Random register instructions

Instruction Coverage Analysis

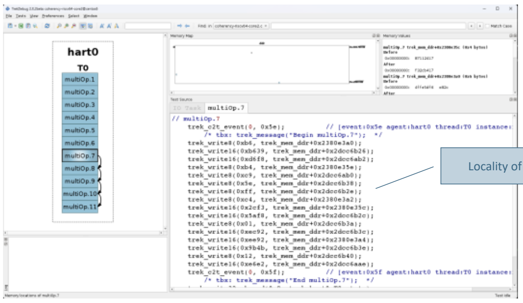


27/103 reachable opcode have been exercised



Atomics, loads and stores not reachable in register only test

RV64 Core Load/Store



```
multiOp.7 trek_mem_ddr+0x2380e35c (0x4 bytes)
Before
0x00000000: 87112417
After
0x0000000c: f32cb417
multiOp.7 trek_mem_ddr+0x2380e3a0 (0x6 bytes)
Before
0x00000000: dffefdf6 e82c
After
```

```
// multiOp.7
    trek_c2t_event(0, 0x5e);            // [event:0x5e agent:hart0 thread:T0 instance:
        /* tbx: trek_message("Begin multiOp.7");  */
    trek_write8(0xb6, trek_mem_ddr+0x2380e3a0);
    trek_write16(0xb639, trek_mem_ddr+0x2dcc6b26);
    trek_write16(0xd6f8, trek_mem_ddr+0x2dcc6ab2);
    trek_write8(0xb4, trek_mem_ddr+0x2380e35e);
    trek_write8(0xc9, trek_mem_ddr+0x2dcc6ab0);
    trek_write8(0x5e, trek_mem_ddr+0x2dcc6b38);
    trek_write8(0xff, trek_mem_ddr+0x2dcc6b2e);
    trek_write8(0xc4, trek_mem_ddr+0x2380e3a2);
    trek_write16(0x2cf3, trek_mem_ddr+0x2380e35c);
    trek_write16(0x5af8, trek_mem_ddr+0x2dcc6b2c);
    trek_write8(0x01, trek_mem_ddr+0x2dcc6b3a);
    trek_write16(0xec92, trek_mem_ddr+0x2dcc6b3c);
    trek_write16(0xee92, trek_mem_ddr+0x2380e3a4);
    trek_write16(0x9b4b, trek_mem_ddr+0x2dcc6b3e);
    trek_write8(0x12, trek_mem_ddr+0x2dcc6b40);
    trek_write16(0xe6e2, trek_mem_ddr+0x2dcc6aae);
    trek_c2t_event(0, 0x5f);            // [event:0x5f agent:hart0 thread:T0 instance:
        /* tbx: trek_message("End multiOp.7");  */
```

Locality of write addrs

Example Address Allocation Patterns

- Random Clusters with locality of reference

```
// memAllocAddrSlice allocated setId:0x1 of 0x4 blocks
// memAllocAddrRand size:0x8 addr: trek_mem_ddr+0x08b810c8
// memAllocAddrRand size:0x8 addr: trek_mem_ddr+0x2380e378
// memAllocAddrRand size:0x8 addr: trek_mem_ddr+0x2380e380
// memAllocAddrRand size:0x8 addr: trek_mem_ddr+0x2380e370
```

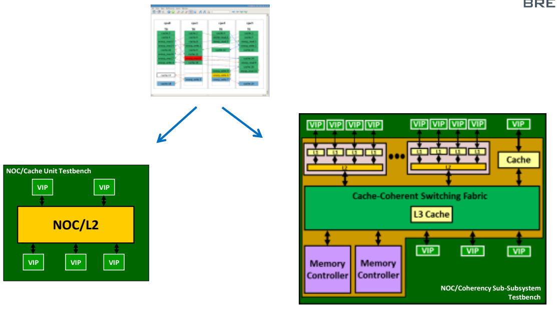
- Stride Patterns across fixed address distances

```
// memAllocAddrSlice allocated setId:0x1 of 0x4 blocks
// memAllocAddrStride stride_len:0x2000 size:0x8 addr: trek_mem_ddr+0x08b830c8
// memAllocAddrStride stride_len:0x2000 size:0x8 addr: trek_mem_ddr+0x08b850c8
// memAllocAddrStride stride_len:0x2000 size:0x8 addr: trek_mem_ddr+0x08b870c8
// memAllocAddrStride stride_len:0x2000 size:0x8 addr: trek_mem_ddr+0x08b890c8
```

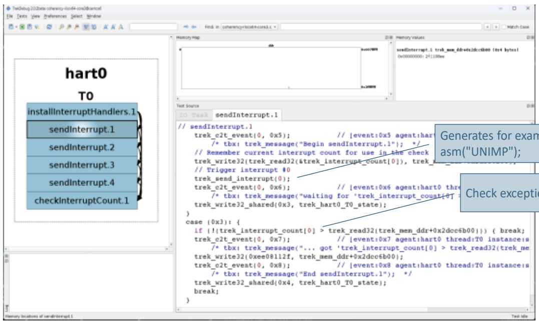
- Sequential Addresses matching a specific Hash

```
// memAllocAddrSlice allocated setId:0x1 of 0x4 blocks
// memAllocAddrHash hash:0x44 size:0x100 addr: trek_mem_ddr+0x08bc1100
// memAllocAddrHash hash:0x44 size:0x100 addr: trek_mem_ddr+0x08c01100
// memAllocAddrHash hash:0x44 size:0x100 addr: trek_mem_ddr+0x08c41100
// memAllocAddrHash hash:0x44 size:0x100 addr: trek_mem_ddr+0x08c81100
```

Application to Unit Bench and Sub-System Bench



RV64 Core Exception Testing



```
sendInterrupt.1 trek_mem_ddr+0x2dcc6b00 (0x4 bytes)
0x00000000: 2f1108ee
```

```
// sendInterrupt.1
    trek_c2t_event(0, 0x5);             // [event:0x5 agent:hart
        /* tbx: trek_message("Begin sendInterrupt.1");  */
    // Remember current interrupt count for use in the check
    trek_write32(trek_read32(&trek_interrupt_count[0]), trek_
    // Trigger interrupt #0
    trek_send_interrupt(0);
    trek_c2t_event(0, 0x6);             // [event:0x6 agent:hart0 thre
        /* tbx: trek_message("waiting for 'trek_interrupt_count[0]
    trek_write32_shared(0x3, trek_hart0_T0_state);
  }
  case (0x3): {
    if (!(trek_interrupt_count[0] > trek_read32(trek_mem_ddr+0x2dcc6b00))) { break;
    trek_c2t_event(0, 0x7);             // [event:0x7 agent:hart0 thread:T0 instance:s
        /* tbx: trek_message("... got 'trek_interrupt_count[0] > trek_read32(trek_me
    trek_write32(0xee08112f, trek_mem_ddr+0x2dcc6b00);
    trek_c2t_event(0, 0x8);             // [event:0x8 agent:hart0 thread:T0 instance:s
        /* tbx: trek_message("End sendInterrupt.1");  */
    trek_write32_shared(0x4, trek_hart0_T0_state);
    break;
  }
```

Generates for example, asm("UNIMP");

Check exception counts

Page Based Virtual Memory Tests

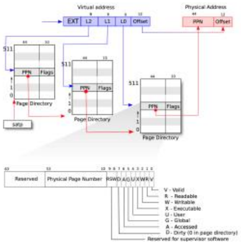
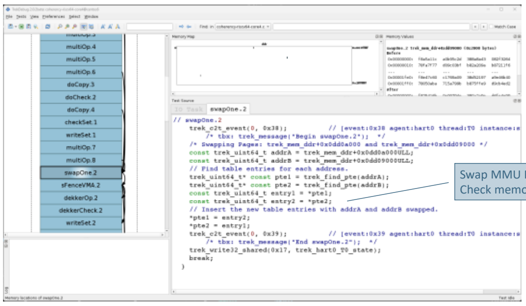


Figure 3.2: RISC-V address translation details.

RV64 Core Page Based MMU Tests



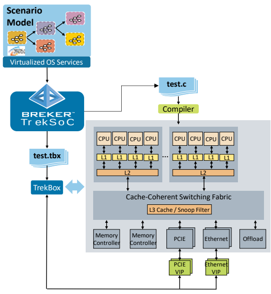
```
// swapOne.2
    trek_c2t_event(0, 0x38);          // [event:0x38 agent:hart0 thread:T0 instance:s
        /* tbx: trek_message("Begin swapOne.2");  */
    /* Swapping Pages: trek_mem_ddr+0x0dd0a000 and trek_mem_ddr+0x0dd09000 */
    const trek_uint64_t addrA = trek_mem_ddr+0x0dd0a000ULL;
    const trek_uint64_t addrB = trek_mem_ddr+0x0dd09000ULL;
    // Find table entries for each address.
    trek_uint64_t* const pte1 = trek_find_pte(addrA);
    trek_uint64_t* const pte2 = trek_find_pte(addrB);
    const trek_uint64_t entry1 = *pte1;
    const trek_uint64_t entry2 = *pte2;
    // Insert the new table entries with addrA and addrB swapped.
    *pte1 = entry2;
    *pte2 = entry1;
    trek_c2t_event(0, 0x39);          // [event:0x39 agent:hart0 thread:T0 instance:s
        /* tbx: trek_message("End swapOne.2");  */
    trek_write32_shared(0x17, trek_hart0_T0_state);
    break;
  }
```

Swap MMU PTE's and Check memory access

Agenda

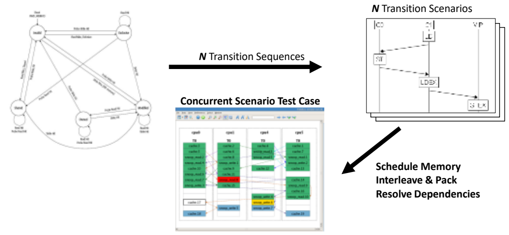
- Test Suite Synthesis and SystemVIP
- RISC-V Core Verification SystemVIP
- RISC-V SoC Verification SystemVIP

RISC-V SoC Testbench Integration



Multi-Agent Scheduling Plans: Overview

- True Sharing within scenario
- False Sharing across scenarios



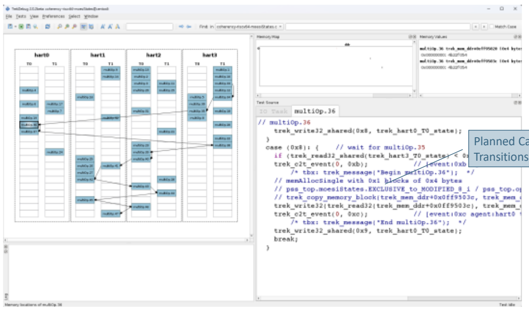
N Transition Sequences

N Transition Scenarios

Concurrent Scenario Test Case

Schedule Memory
Interleave & Pack
Resolve Dependencies

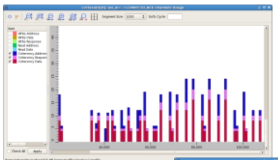
RV64 MultiCore MoesiStates



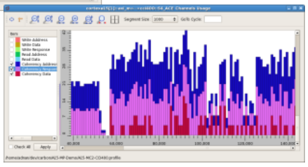
Planned Cache State Transitions

```
// multiOp.36
    trek_write32_shared(0x8, trek_hart0_T0_state);
  }
  case (0x8): {     // wait for multiOp.35
    if (trek_read32_shared(trek_hart3_T0_state) < 0x
    trek_c2t_event(0, 0xb);              // [event:0xb
        /* tbx: trek_message("Begin multiOp.36"); */
    // memAllocSingle with 0x1 blocks of 0x4 bytes
    // pss_top.moesiStates.EXCLUSIVE_to_MODIFIED_8_i / pss_top.o
    // trek_copy_memory_block(trek_mem_ddr+0x0ff9503c, trek_mem_
    trek_write32(trek_read32(trek_mem_ddr+0x0ff9503c), trek_mem_
    trek_c2t_event(0, 0xc);              // [event:0xc agent:hart0
        /* tbx: trek_message("End multiOp.36"); */
    trek_write32_shared(0x9, trek_hart0_T0_state);
    break;
  }
```

Efficacy of System-Integrity Testing using the RISC-V TrekApp

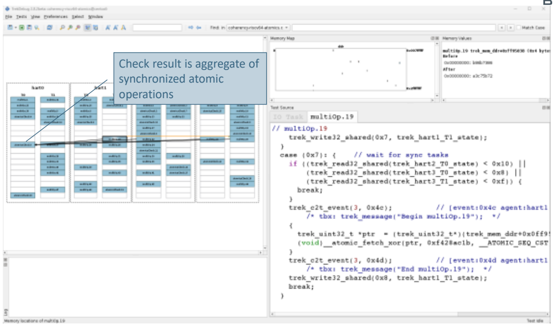


Typical directed coherency test …



… vs. RISC-V TrekApp automated Sys-Integrity tests

Atomics Testing



Check result is aggregate of synchronized atomic operations

```
// multiOp.19
    trek_write32_shared(0x7, trek_hart1_T1_state);
  }
  case (0x7): {     // wait for sync tasks
    if ((trek_read32_shared(trek_hart2_T0_state) < 0x10) ||
        (trek_read32_shared(trek_hart3_T0_state) < 0x8) ||
        (trek_read32_shared(trek_hart3_T1_state) < 0xf)) {
      break;
    }
    trek_c2t_event(3, 0x4c);             // [event:0x4c agent:hart1
        /* tbx: trek_message("Begin multiOp.19"); */
    {
      trek_uint32_t *ptr  = (trek_uint32_t*)(trek_mem_ddr+0x0ff9
      (void)__atomic_fetch_xor(ptr, 0xf428ac1b, __ATOMIC_SEQ_CST
    }
    trek_c2t_event(3, 0x4d);             // [event:0x4d agent:hart1
        /* tbx: trek_message("End multiOp.19"); */
    trek_write32_shared(0x8, trek_hart1_T1_state);
    break;
  }
```

RISC-V SoC Memory Ordering: Dekker Algorithm

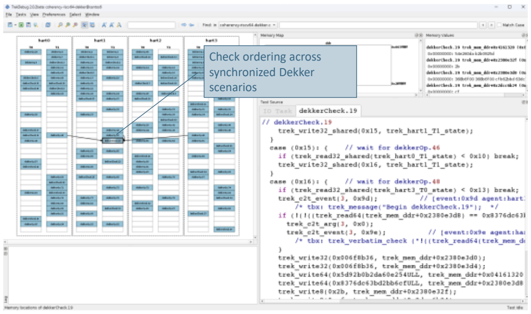
- Assume initial state A=0 , B=0

- The Dekker Algorithm States

```
core 0: ST A, 1; MEM_BARRIER; LD B
core 1: ST B, 1; MEM_BARRIER; LD A
error iff ( A == 0 && B == 0 )
```

- This is a test for a weakly ordered memory system
  - Such a system must preserve the property that a LD may not reorder ahead of a previous ST from the same agent

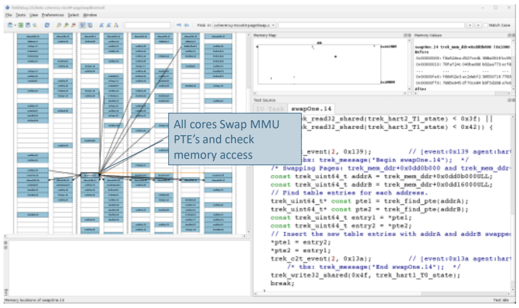
Dekker Memory Ordering



Check ordering across synchronized Dekker scenarios

```
// dekkerCheck.19
    trek_write32_shared(0x15, trek_hart1_T1_state);
  }
  case (0x15): {     // wait for dekkerOp.46
    if (trek_read32_shared(trek_hart0_T1_state) < 0x10) break;
    trek_write32_shared(0x16, trek_hart1_T1_state);
  }
  case (0x16): {     // wait for dekkerOp.48
    if (trek_read32_shared(trek_hart3_T0_state) < 0x13) break;
    trek_c2t_event(3, 0x9d);             // [event:0x9d agent:hart
        /* tbx: trek_message("Begin dekkerCheck.19"); */
    if (!(!((trek_read64(trek_mem_ddr+0x2380e3d8) == 0x8376dc63
      trek_c2t_arg(3, 0x0);
      trek_c2t_event(3, 0x9e);             // [event:0x9e agent:ha
        /* tbx: trek_verbatim_check ("!((trek_read64(trek_mem_d
    }
    trek_write32(0x006f8b36, trek_mem_ddr+0x2380e3d0);
    trek_write32(0x006f8b36, trek_mem_ddr+0x2380e3d4);
    trek_write64(0x5d92b0b2da60e254ULL, trek_mem_ddr+0x04161320
    trek_write64(0x8376dc63bd2bb6cfULL, trek_mem_ddr+0x2380e3d8
    trek_write8(0x2b, trek_mem_ddr+0x2380e32f);
```

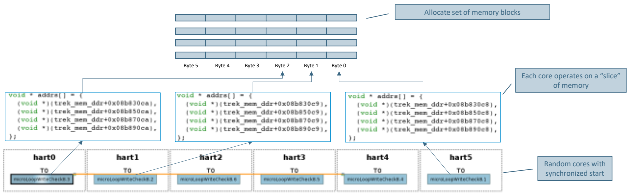
MultiCore MMU Tests



All cores Swap MMU PTE's and check memory access

```
k_read32_shared(trek_hart2_T1_state) < 0x3f) ||
k_read32_shared(trek_hart3_T1_state) < 0x42)) {

_event(2, 0x139);             // [event:0x139 agent:hart
bx: trek_message("Begin swapOne.14"); */
/* Swapping Pages: trek_mem_ddr+0x0dd0b000 and trek_mem_ddr-
const trek_uint64_t addrA = trek_mem_ddr+0x0dd0b000ULL;
const trek_uint64_t addrB = trek_mem_ddr+0x0dd16000ULL;
// Find table entries for each address.
trek_uint64_t* const pte1 = trek_find_pte(addrA);
trek_uint64_t* const pte2 = trek_find_pte(addrB);
const trek_uint64_t entry1 = *pte1;
const trek_uint64_t entry2 = *pte2;
// Insert the new table entries with addrA and addrB swappe
*pte1 = entry2;
*pte2 = entry1;
trek_c2t_event(2, 0x13a);             // [event:0x13a agent:hart
    /* tbx: trek_message("End swapOne.14"); */
trek_write32_shared(0x4f, trek_hart1_T0_state);
break;
```

False-Share Memory Stress Tests



Allocate set of memory blocks

Each core operates on a "slice" of memory

Random cores with synchronized start

```
void * addrs[] = {
  (void *)(trek_mem_ddr+0x08b830ca),
  (void *)(trek_mem_ddr+0x08b850ca),
  (void *)(trek_mem_ddr+0x08b870ca),
  (void *)(trek_mem_ddr+0x08b890ca),
};
```

```
void * addrs[] = {
  (void *)(trek_mem_ddr+0x08b830c9),
  (void *)(trek_mem_ddr+0x08b850c9),
  (void *)(trek_mem_ddr+0x08b870c9),
  (void *)(trek_mem_ddr+0x08b890c9),
};
```

```
void * addrs[] = {
  (void *)(trek_mem_ddr+0x08b830c8),
  (void *)(trek_mem_ddr+0x08b850c8),
  (void *)(trek_mem_ddr+0x08b870c8),
  (void *)(trek_mem_ddr+0x08b890c8),
};
```

```
int trek_microloop_write_check8( void * addrs[], int count, trek_uint8_t pattern){
  int errorCount = 0;
  int ii;
  for ( ii = 0; ii < count; ++ii){
    trek_write8(pattern, addrs[ii]);
  }
  for ( ii = 0; ii < count; ++ii){
    if (trek_read8(addrs[ii]) != pattern) {
      ++errorCount;
      trek_runtime_error("trek_microloop_write_check8", addrs[ii], pattern, trek_read8(addrs[ii]));
    };
  }
  return errorCount;
}
```

```
for ( int ii = 0; ii < 1000; ++ii ){
  errorCount += trek_microloop_write_check8(addrs, 4, 184);
}
```

Each core has free running loop

Thanks for Listening!
Any Questions?

Notes

Balram Naik Meghavath

Broadcom Ltd
Sr Staff Engineer

Improve the Quality of the Testbenches using specialized PySlint solutions

User Paper

Abstract

Simulation is the most common RTL verification technique, involving the execution of testbenches are essential for the verification of the designs. SystemVerilog and UVM have been widely adopted over the last two decades. However, the complex nature of these languages/methodologies can make it difficult for junior-level engineers to create maintainable and reusable code.

Static linting checks have been widely used for RTL design, but they have not been used as widely adopted for Testbenches. In this talk, we share our experience in using a popular open-source framework named PySlint to lint-check SystemVerilog UVM Testbeneches.

We show how PySlint can be used to identify potential problems in SV-UVM Testbenches, such as coding style violations, potential bugs, and potential performance bottlenecks. We also show how PySlint can be used to generate reports that can help engineers to improve the quality of their Testbenches buiodling a robust verification environment process. Some key components of a robust verification environment includes, Testbenches, coverage matrics, Assertions, regression testing.

We believe that PySlint can be a valuable tool for improving the quality of SystemVerilog UVM testbenches. By using PySlint, engineers can identify and fix potential problems in their testbenches early in the development process, which can help to prevent costly delays and errors.

SystemVerilog:
1800-2017 - IEEE Standard for SystemVerilog--Unified Hardware Design, Specification, and Verification Language

UVM Use Guide:
https://www.accellera.org/images/downloads/standards/uvm/uvm_users_guide_1.2.pdf

PySlink:
https://github.com/svenka3/pyslint

Biography

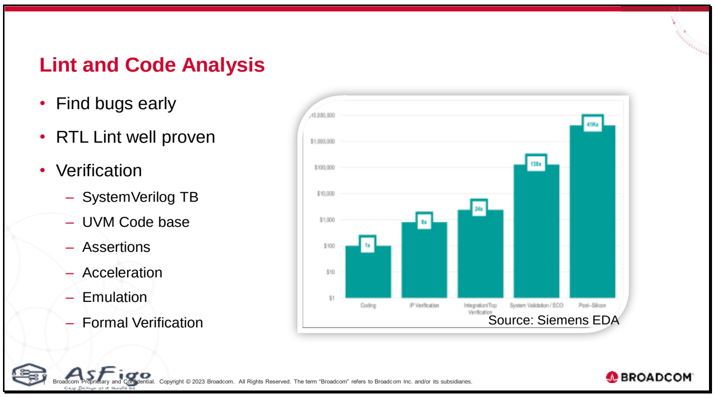
15+ years of Design Verification exp.
Prior to Broadcom, I worked for Microchip , Western Digital.
My Verification expertise from IP level to SOC subsystems
Initially, I worked for VIP development, Net works chips, Mobile chips, and Storage product lines, Last 10 years mainly with Wireless products, **GPS** and **Bluetooth**, **Wifi** SoC's.

VF2023 VERIFICATION FUTURES

Improve the Quality of the Testbenches using specialized PySlint solutions

Balram Meghavath
Srinivasan Venkataramanan

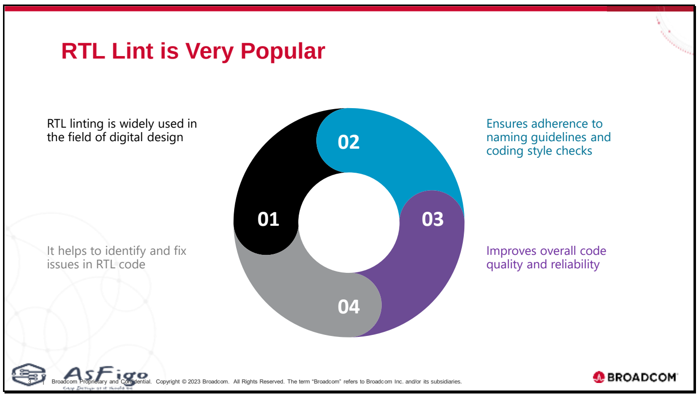
www.asfigo.com



Lint and Code Analysis

- Find bugs early
- RTL Lint well proven
- Verification
  - SystemVerilog TB
  - UVM Code base
  - Assertions
  - Acceleration
  - Emulation
  - Formal Verification

Source: Siemens EDA



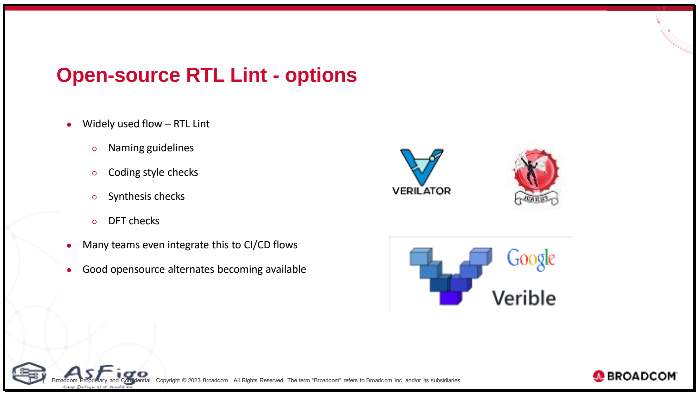
RTL Lint is Very Popular

RTL linting is widely used in the field of digital design

It helps to identify and fix issues in RTL code

Ensures adherence to naming guidelines and coding style checks

Improves overall code quality and reliability



Open-source RTL Lint - options

- Widely used flow – RTL Lint
  - Naming guidelines
  - Coding style checks
  - Synthesis checks
  - DFT checks
- Many teams even integrate this to CI/CD flows
- Good opensource alternates becoming available

TB Lint - Challenges and Alternatives

- New Paradigm: Code analysis approaches changing
- Lack of good parsers/tools/API for newer languages
- Slang Verible, Svlint, and PySlint are good opensource alternatives
- Provide flexibility and customization options

Testbench Linting

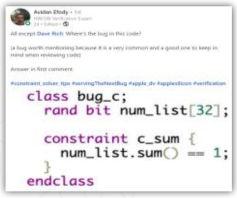
- New paradigm
  - Old/proven concept
  - Lack of good parsers/tools/API
- Verible – C++ based rules
  - https://github.com/chipsalliance/verible
- Svlint – Rust based, custom plugin
  - https://github.com/dalance/svlint
- UVMLint – Python based, custom plugin for UVM TB
  - https://github.com/AsFigo/UVMLint
- PySlint – Python based, works on top of slang/pyslang pkg
  - https://github.com/AsFigo/pyslint

Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live

Testbench Linting – Sample rules

PySlint Rule ID	Description
NAME_CL_PREFIX NAME_CG_PREFIX NAME_ASM_PREFIX NAME_COV_PREFIX	Naming/style checks
SVA_MISSING_LABEL SVA_MISSING_ENDLABEL SVA_NO_PASS_AB SVA_MISSING_FAIL_AB	Assertion specific. Performance, Debug categories
CL_METHOD_NOT_EXTERN CL_MISSING_ENDLABEL FUNC_CNST_MISSING_CAST	Class specific – style, functionality

SV Constraints – case study



Avidan Efody • 1st
HW/SW Verification Expert
2d • Edited •

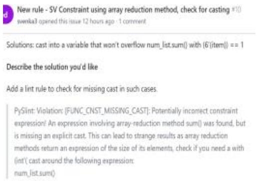
All except Dave Rich: Where's the bug in this code?

(a bug worth mentioning because it is a very common and a good one to keep in mind when reviewing code)

Answer in first comment

#constraint_solver_tips #servingTheNextBug #apple_dv #applesilicon #verification

```
class bug_c;
  rand bit num_list[32];

  constraint c_sum {
    num_list.sum() == 1;
  }
endclass
```



New rule - SV Constraint using array reduction method, check for casting #10
swenka3 opened this issue 12 hours ago · 1 comment

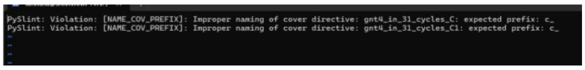
Solutions: cast into a variable that won't overflow num_list.sum() with (6'(item)) == 1

Describe the solution you'd like

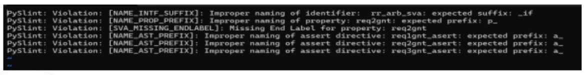
Add a lint rule to check for missing cast in such cases.

> PySlint: Violation: [FUNC_CNST_MISSING_CAST]: Potentially incorrect constraint expression! An expression involving array-reduction method sum() was found, but is missing an explicit cast. This can lead to strange results as array reduction methods return an expression of the size of its elements, check if you need a with (int'(cast around the following expression:
> num_list.sum()

PySlint– results



```
PySlint: Violation: [NAME_COV_PREFIX]: Improper naming of cover directive: gnt4_in_31_cycles_C: expected prefix: c_
PySlint: Violation: [NAME_COV_PREFIX]: Improper naming of cover directive: gnt4_in_31_cycles_C1: expected prefix: c_
~
~
~
~
```



```
PySlint: Violation: [NAME_INTF_SUFFIX]: Improper naming of identifier:  rr_arb_sva: expected suffix: _if
PySlint: Violation: [NAME_PROP_PREFIX]: Improper naming of property: req2gnt: expected prefix: p_
PySlint: Violation: [SVA_MISSING_ENDLABEL]: Missing End Label for property: req2gnt
PySlint: Violation: [NAME_AST_PREFIX]: Improper naming of assert directive: req1gnt_asert: expected prefix: a_
PySlint: Violation: [NAME_AST_PREFIX]: Improper naming of assert directive: req2gnt_asert: expected prefix: a_
PySlint: Violation: [NAME_AST_PREFIX]: Improper naming of assert directive: req3gnt_asert: expected prefix: a_
~
~
```

UVMLint– results

Testbench Linting - summary

- Exciting new frontiers
- Opensource can now lead the way for commercial tools to follow
  - Than just playing a catch-up game
  - Faster turn-around time
  - Potentially vast set of developers!

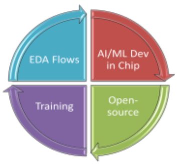
Summary, next steps

- Open-source verification is now real
- Shared our experience in:
  - Assertions
  - Unit Testing
  - RTL Linting
  - Testbench Linting
- PySlint – a case where open-source can leapfrog beyond commercial tools
  - Opportunity for US Universities to take the lead!

Our contributions – GitHub repos/projects

Who we are, what As-Figo do ?

- UK based chip design start-up
  - Bringing decades of Chip Design and EDA expertise
  - Globally reputed training solutions
- RTL Design, Verification
  - SystemVerilog, VHDL, e, PSL etc.
  - Methodologies: UVM, OVM, VMM, eRM etc.
- Bringing AI/ML to chip design flows
  - Generative AI
  - AI Analytics



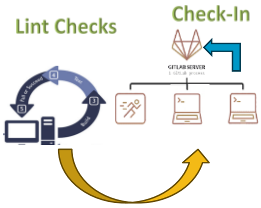
AsFigo
Chip Design as it should be

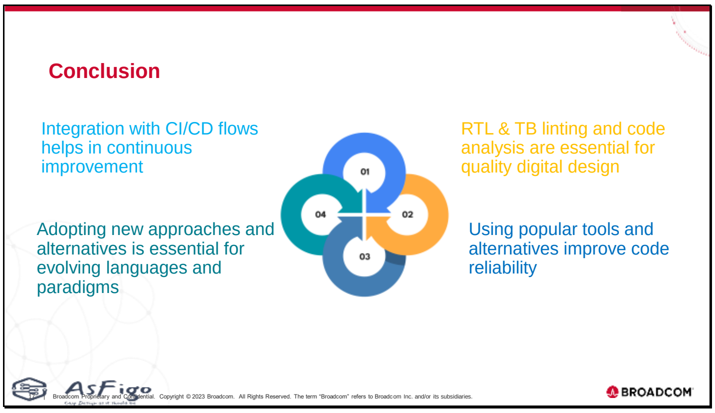
Summary, next steps

- Proved that open-source is now ready for primetime in Design-Verification
- All our code will be available as open-source via GitHub - https://github.com/svenka3/af_cip_verilator
- Looking for volunteers to fix issues on Verilator, improve SVA support
  - svenka3@gmail.com
  - Balram.Meghavath@broadcom.com

Integrate Lint to CI/CD

- Subset of Lint rules
- Customize as per project stage/phase
- Stricter as it gets mature
- Applies to RTL & TB
- Ideally use open-source tools
- Free of license restrictions
- Easy to customize
- Commercial license models available (MIT)





Conclusion

Integration with CI/CD flows helps in continuous improvement

RTL & TB linting and code analysis are essential for quality digital design

Adopting new approaches and alternatives is essential for evolving languages and paradigms

Using popular tools and alternatives improve code reliability

Notes

Hemendra Talesara

Bitstar Technologies
Advisor

Verification by Documentation

User Paper

Abstract

With all the innovations in tools to help with functional closure, one that is not fully appreciated is the need for disciplined documentation at every stage of verification. Documentation often is a victim of tight schedules and yet lack of it remains the root cause of schedule slips, many escapes, delays and churn in projects. We will explore different kinds of documents and shine light on this problem. We claim, without due appreciation of this, we can not close the verification gap.

Biography

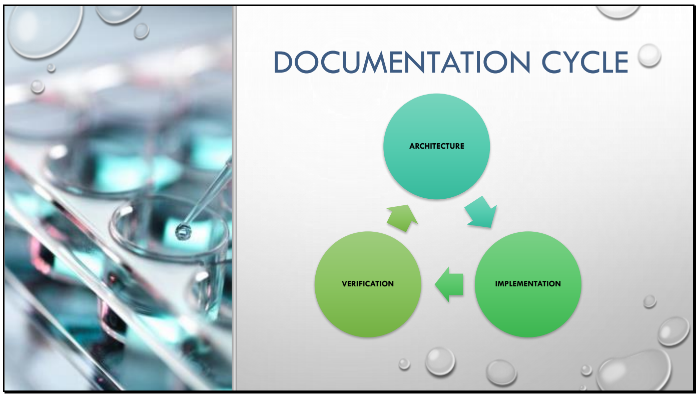
Hemendra Talesara is a distinguished leader in the semiconductor sector in the USA, amassing a rich legacy of over 35 years. He served in senior management roles with IBM, AMD, Synopsys, and many other organizations. Specializing in CPU, GPU, ASIC, and SOC chip design, he boasts profound chip design verification technology expertise. Hemendra's exceptional competencies encompass global team formation, project risk management, and certified corporate governance as a NACD Certified Corporate Board Director. His interests include business Implications of digital disruption, AI, and cyber security. Hemendra's academic journey encompasses an MS in Electrical and Computer Engineering from the University of Texas at Austin and a BE in Electronics and Telecommunication Engineering from IIEST, Shibpur, India. Beyond his professional journey, Hemendra's active involvement in industry conferences, technical journals, teaching, travel, and diverse hobbies showcases his multifaceted persona. With a career defined by inclusive and collaborative leadership, technical prowess, and a commitment to innovation, Hemendra Talesara brings considerable depth to any organization.

Notes

VERIFICATION BY DOCUMENTATION

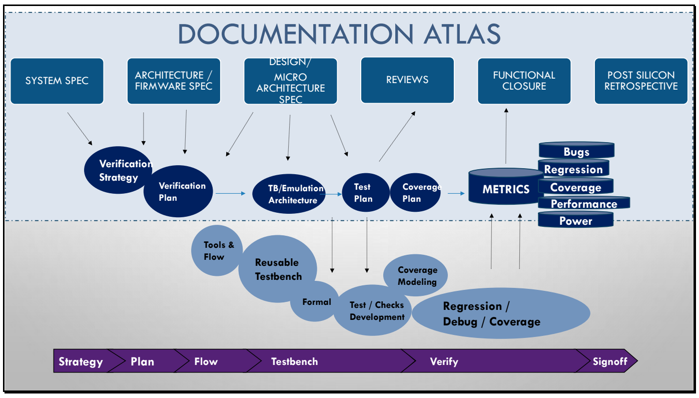
THINK BUG ESCAPE

Hemendra Talesara
Advisor, Mentor, Board Member
Verification Startups



DOCUMENTATION CYCLE

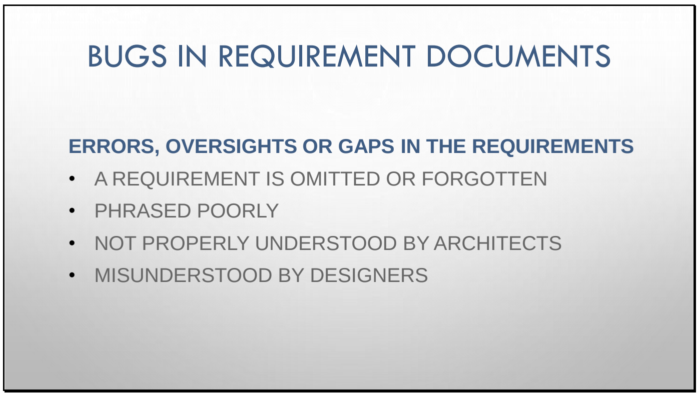
DOCUMENTATION ATLAS



SELECT SOURCES OF

BUG ESCAPES

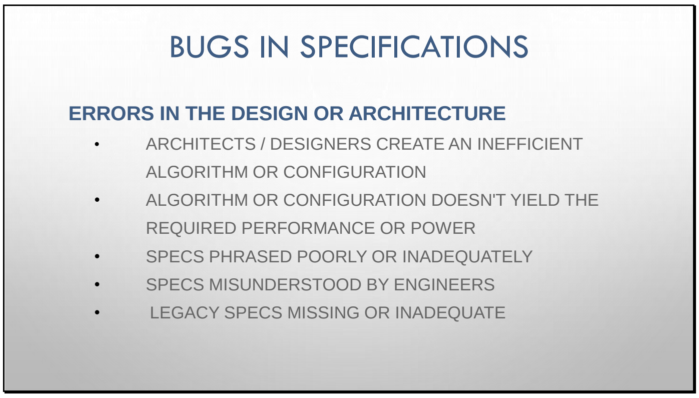
1. **REQUIREMENT DOCUMENTS**
   ERRORS, OVERSIGHTS OR GAPS IN THE REQUIREMENTS.
2. **SPECIFICATIONS**
   ERRORS IN THE DESIGN OR ARCHITECTURE. MISSING ASSUMPTIONS
3. **IMPLEMENTATION**
   ERRORS IN THE CODING OR IMPLEMENTATION.
4. **VERIFICATION PLAN**
   ERRORS IN THE TEST PLANNING OR TEST ACTIVITIES..
5. **COVERAGE PLAN**
   GAPS IN TESTING; VERIFICATION UNIVERSE IS VERY LARGE
6. **REVIEWS (DOCUMENTS AND CODE)**
   ERRORS IN THE PROCESS OR POLICIES



BUGS IN REQUIREMENT DOCUMENTS

ERRORS, OVERSIGHTS OR GAPS IN THE REQUIREMENTS

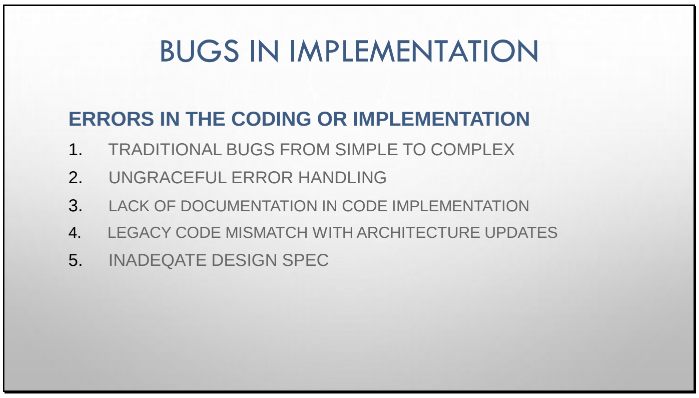
- A REQUIREMENT IS OMITTED OR FORGOTTEN
- PHRASED POORLY
- NOT PROPERLY UNDERSTOOD BY ARCHITECTS
- MISUNDERSTOOD BY DESIGNERS



BUGS IN SPECIFICATIONS

ERRORS IN THE DESIGN OR ARCHITECTURE

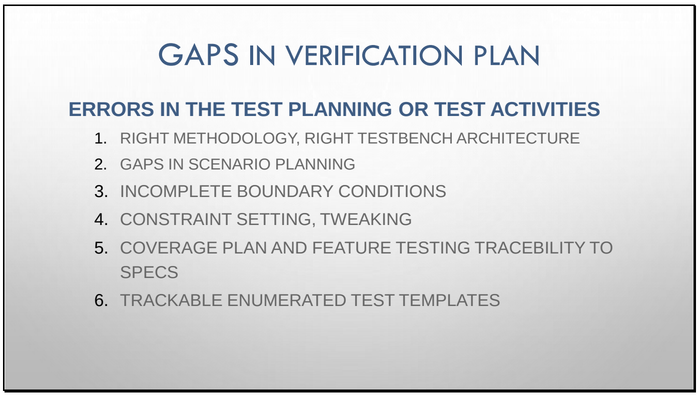
- ARCHITECTS / DESIGNERS CREATE AN INEFFICIENT ALGORITHM OR CONFIGURATION
- ALGORITHM OR CONFIGURATION DOESN'T YIELD THE REQUIRED PERFORMANCE OR POWER
- SPECS PHRASED POORLY OR INADEQUATELY
- SPECS MISUNDERSTOOD BY ENGINEERS
- LEGACY SPECS MISSING OR INADEQUATE



BUGS IN IMPLEMENTATION

ERRORS IN THE CODING OR IMPLEMENTATION

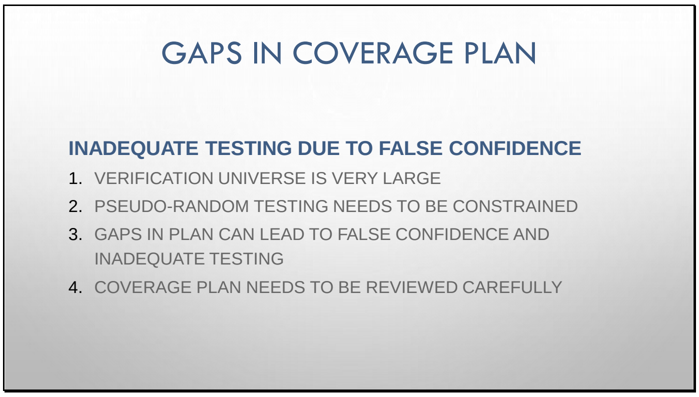
1. TRADITIONAL BUGS FROM SIMPLE TO COMPLEX
2. UNGRACEFUL ERROR HANDLING
3. LACK OF DOCUMENTATION IN CODE IMPLEMENTATION
4. LEGACY CODE MISMATCH WITH ARCHITECTURE UPDATES
5. INADEQATE DESIGN SPEC



GAPS IN VERIFICATION PLAN

ERRORS IN THE TEST PLANNING OR TEST ACTIVITIES

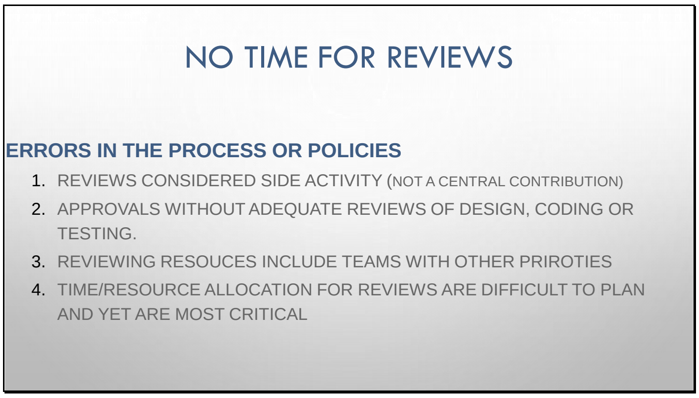
1. RIGHT METHODOLOGY, RIGHT TESTBENCH ARCHITECTURE
2. GAPS IN SCENARIO PLANNING
3. INCOMPLETE BOUNDARY CONDITIONS
4. CONSTRAINT SETTING, TWEAKING
5. COVERAGE PLAN AND FEATURE TESTING TRACEBILITY TO SPECS
6. TRACKABLE ENUMERATED TEST TEMPLATES



GAPS IN COVERAGE PLAN

INADEQUATE TESTING DUE TO FALSE CONFIDENCE

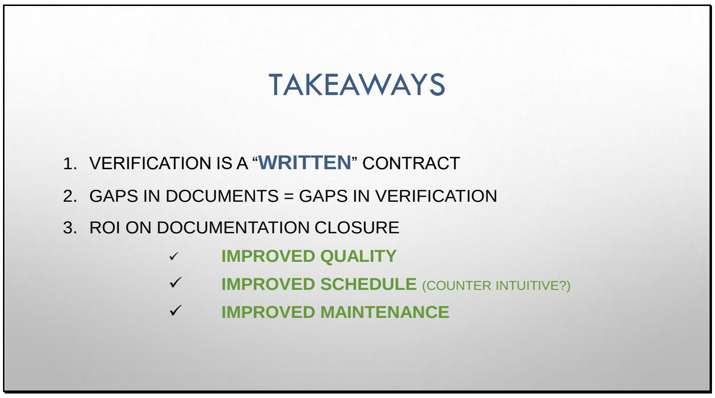
1. VERIFICATION UNIVERSE IS VERY LARGE
2. PSEUDO-RANDOM TESTING NEEDS TO BE CONSTRAINED
3. GAPS IN PLAN CAN LEAD TO FALSE CONFIDENCE AND INADEQUATE TESTING
4. COVERAGE PLAN NEEDS TO BE REVIEWED CAREFULLY



NO TIME FOR REVIEWS

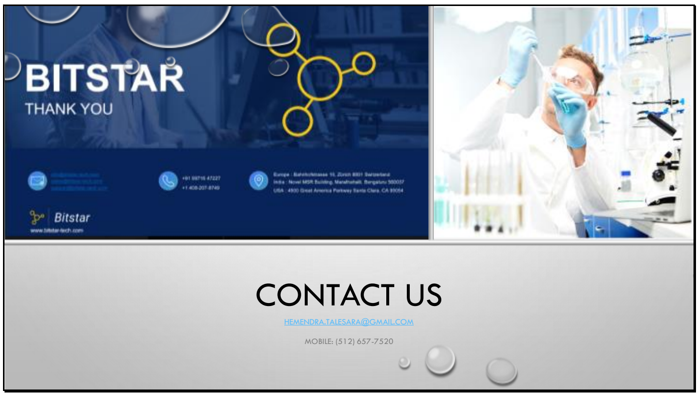
ERRORS IN THE PROCESS OR POLICIES

1. REVIEWS CONSIDERED SIDE ACTIVITY (NOT A CENTRAL CONTRIBUTION)
2. APPROVALS WITHOUT ADEQUATE REVIEWS OF DESIGN, CODING OR TESTING.
3. REVIEWING RESOUCES INCLUDE TEAMS WITH OTHER PRIROTIES
4. TIME/RESOURCE ALLOCATION FOR REVIEWS ARE DIFFICULT TO PLAN AND YET ARE MOST CRITICAL



TAKEAWAYS

1. VERIFICATION IS A “**WRITTEN**” CONTRACT
2. GAPS IN DOCUMENTS = GAPS IN VERIFICATION
3. ROI ON DOCUMENTATION CLOSURE
   - ✓ **IMPROVED QUALITY**
   - ✓ **IMPROVED SCHEDULE** (COUNTER INTUITIVE?)
   - ✓ **IMPROVED MAINTENANCE**


BITSTAR
THANK YOU
+91 99716 47227
+1 408-207-8749
Europe : Bahnhofstrasse 10, Zürich 8001 Switzerland
India : Novel MSR Building, Marathahalli, Bengaluru 560037
USA : 4500 Great America Parkway Santa Clara, CA 95054
Bitstar
www.bitstar-tech.com

CONTACT US

HEMENDRA.TALESARA@GMAIL.COM

MOBILE: (512) 657-7520

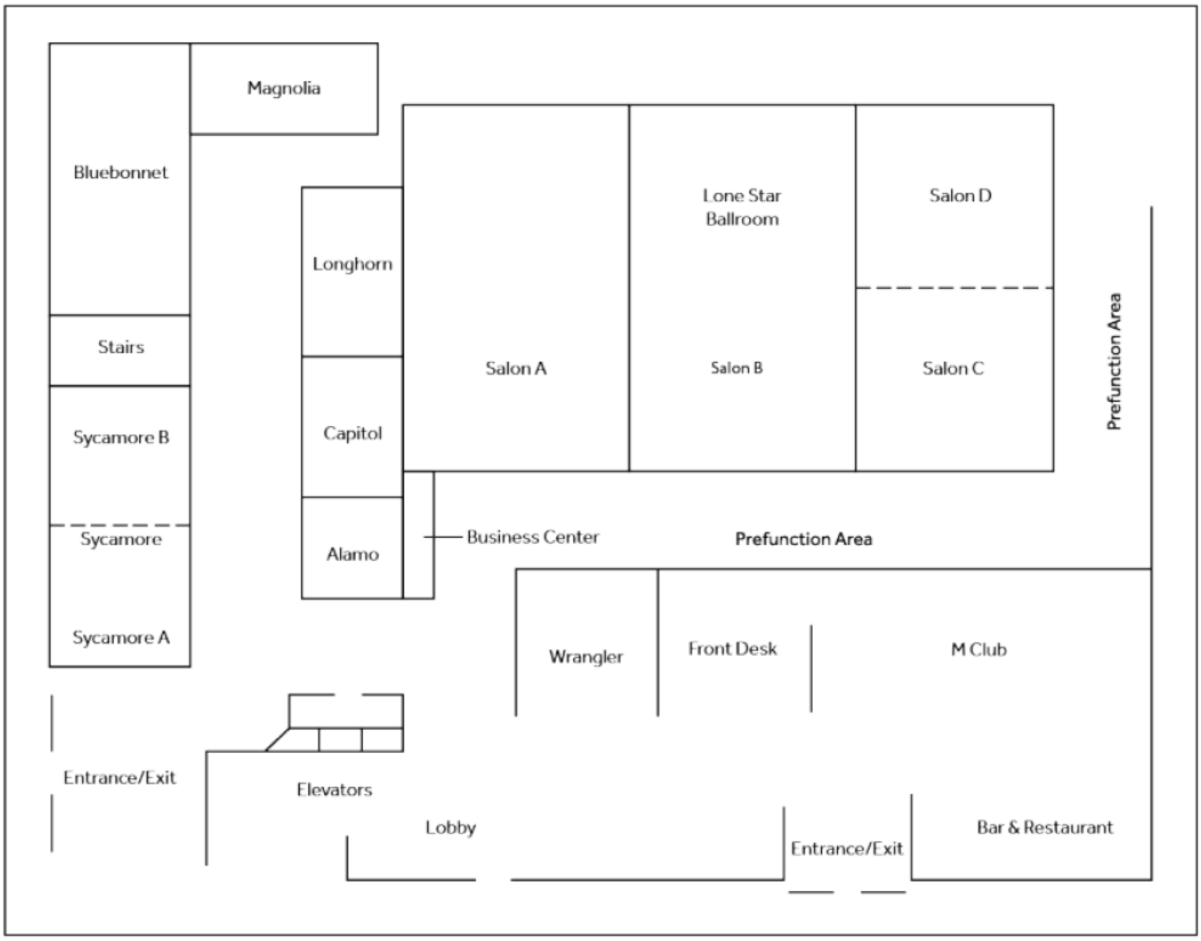
Notes

Track Session

Latest Topics in Verification

Lonestar Ballroom – Salon A+B

FLOOR PLAN



We would be grateful if you could move to the track session as quickly as possible.

Notes

Aditya Devarakonda

NXP Semiconductor

Senior Manager – Design Verification

Leveraging AMS verification and DMS verification for efficiency and quality in Mixed-signal designs

User paper

Abstract

Mixed-signal verification at the SOC level is a unique problem that is currently being tackled by two disciplines of verification- Analog Mixed-Signal (AMS) Verification, and Digital Mixed-Signal Verification (DMS). Each of these disciplines have their own strengths, and weaknesses. Understanding and leveraging those appropriately is needed to achieve the required success in silicon, while using the verification engineering resources efficiently. In this paper I would like to discuss the aspects that need to be considered while determining an effective, and combined AMS and DMS strategy for robust verification of Mixed-signal designs

Biography

Aditya Devarakonda leads Digital Verification for Advanced Power Systems at NXP. Prior to this he held several roles in digital design and digital verification of Mixed-signal ICs at ON, Maxim (ADI), Dialog (Renesas), and Freescale. Aditya holds an MSEE.

Notes

LEVERAGING AMS VERIFICATION AND DMS VERIFICATION FOR EFFICIENCY AND QUALITY IN MIXED- SIGNAL DESIGNS

Aditya, Devarakonda
Advanced Power Systems, NXP

SEP 2023

SECURE CONNECTIONS
FOR A SMARTER WORLD

PUBLIC

AGENDA

What is a Mixed-Signal IC?

Verification of Mixed-Signal ICs

Evolution of DV into DMS

Need for a combined DMS-AMS strategy

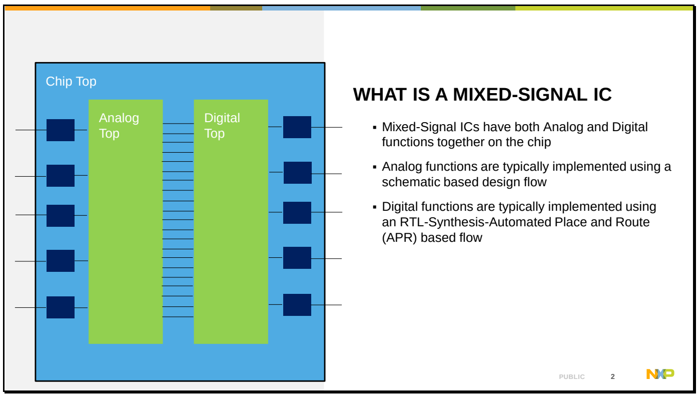
A combined AMS-DMS flow

Transitioning from an AMS-DV to AMS-DMS flow

More on Modeling

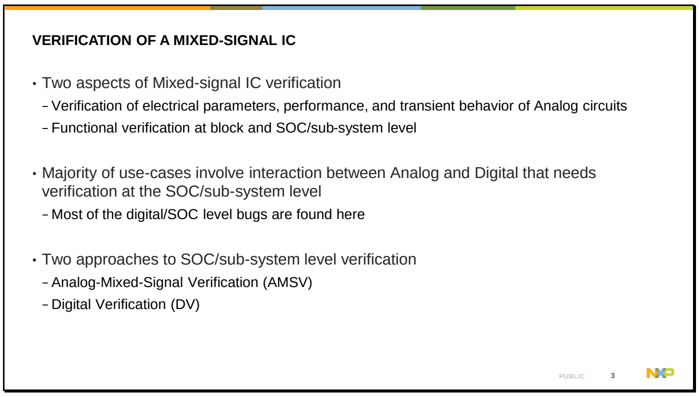
Combined AMS-DMS flow – Some Pitfalls

Conclusions



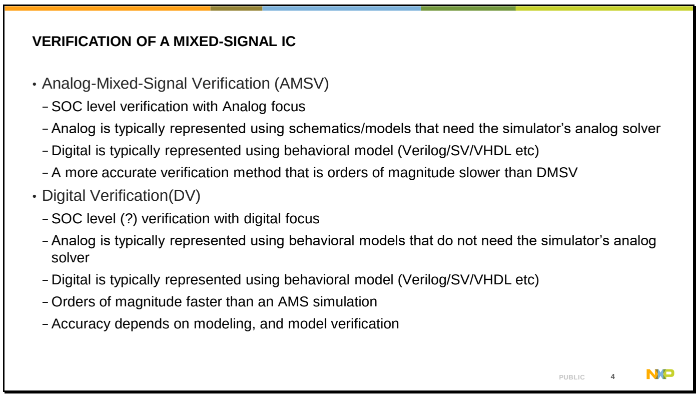
WHAT IS A MIXED-SIGNAL IC

- Mixed-Signal ICs have both Analog and Digital functions together on the chip
- Analog functions are typically implemented using a schematic based design flow
- Digital functions are typically implemented using an RTL-Synthesis-Automated Place and Route (APR) based flow



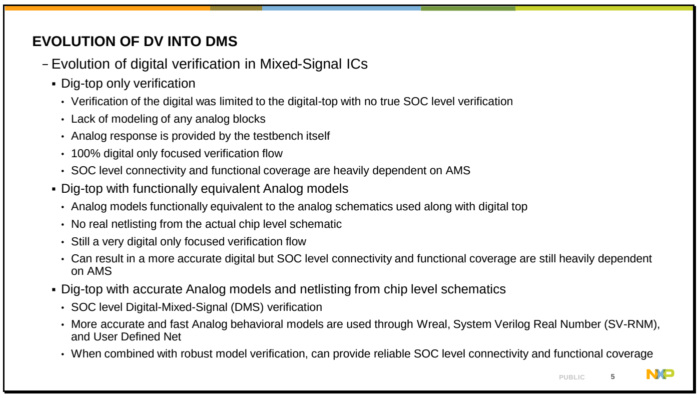
VERIFICATION OF A MIXED-SIGNAL IC

- Two aspects of Mixed-signal IC verification
  - Verification of electrical parameters, performance, and transient behavior of Analog circuits
  - Functional verification at block and SOC/sub-system level
- Majority of use-cases involve interaction between Analog and Digital that needs verification at the SOC/sub-system level
  - Most of the digital/SOC level bugs are found here
- Two approaches to SOC/sub-system level verification
  - Analog-Mixed-Signal Verification (AMSV)
  - Digital Verification (DV)



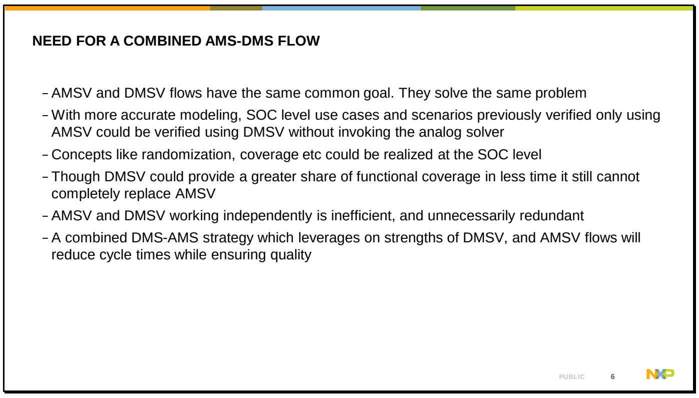
VERIFICATION OF A MIXED-SIGNAL IC

- Analog-Mixed-Signal Verification (AMSV)
  - SOC level verification with Analog focus
  - Analog is typically represented using schematics/models that need the simulator's analog solver
  - Digital is typically represented using behavioral model (Verilog/SV/VHDL etc)
  - A more accurate verification method that is orders of magnitude slower than DMSV
- Digital Verification(DV)
  - SOC level (?) verification with digital focus
  - Analog is typically represented using behavioral models that do not need the simulator's analog solver
  - Digital is typically represented using behavioral model (Verilog/SV/VHDL etc)
  - Orders of magnitude faster than an AMS simulation
  - Accuracy depends on modeling, and model verification



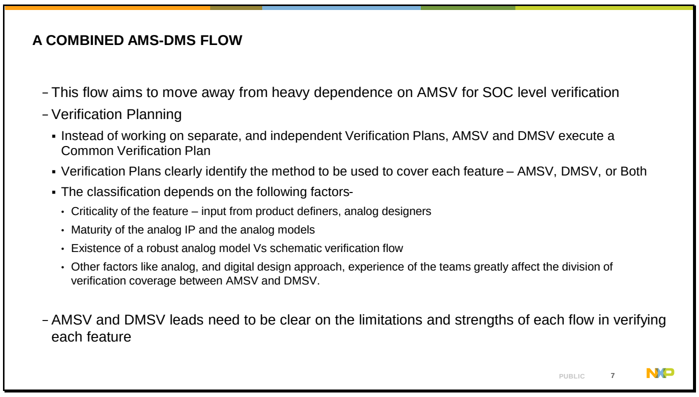
EVOLUTION OF DV INTO DMS

- Evolution of digital verification in Mixed-Signal ICs
  - Dig-top only verification
    - Verification of the digital was limited to the digital-top with no true SOC level verification
    - Lack of modeling of any analog blocks
    - Analog response is provided by the testbench itself
    - 100% digital only focused verification flow
    - SOC level connectivity and functional coverage are heavily dependent on AMS
  - Dig-top with functionally equivalent Analog models
    - Analog models functionally equivalent to the analog schematics used along with digital top
    - No real netlisting from the actual chip level schematic
    - Still a very digital only focused verification flow
    - Can result in a more accurate digital but SOC level connectivity and functional coverage are still heavily dependent on AMS
  - Dig-top with accurate Analog models and netlisting from chip level schematics
    - SOC level Digital-Mixed-Signal (DMS) verification
    - More accurate and fast Analog behavioral models are used through Wreal, System Verilog Real Number (SV-RNM), and User Defined Net
    - When combined with robust model verification, can provide reliable SOC level connectivity and functional coverage



NEED FOR A COMBINED AMS-DMS FLOW

- AMSV and DMSV flows have the same common goal. They solve the same problem
- With more accurate modeling, SOC level use cases and scenarios previously verified only using AMSV could be verified using DMSV without invoking the analog solver
- Concepts like randomization, coverage etc could be realized at the SOC level
- Though DMSV could provide a greater share of functional coverage in less time it still cannot completely replace AMSV
- AMSV and DMSV working independently is inefficient, and unnecessarily redundant
- A combined DMS-AMS strategy which leverages on strengths of DMSV, and AMSV flows will reduce cycle times while ensuring quality



A COMBINED AMS-DMS FLOW

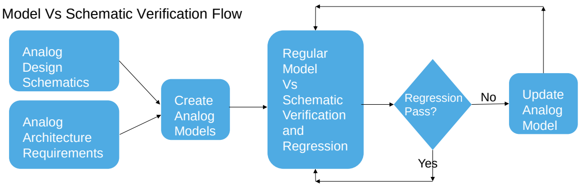
- This flow aims to move away from heavy dependence on AMSV for SOC level verification
- Verification Planning
  - Instead of working on separate, and independent Verification Plans, AMSV and DMSV execute a Common Verification Plan
  - Verification Plans clearly identify the method to be used to cover each feature – AMSV, DMSV, or Both
  - The classification depends on the following factors-
    - Criticality of the feature – input from product definers, analog designers
    - Maturity of the analog IP and the analog models
    - Existence of a robust analog model Vs schematic verification flow
    - Other factors like analog, and digital design approach, experience of the teams greatly affect the division of verification coverage between AMSV and DMSV.
- AMSV and DMSV leads need to be clear on the limitations and strengths of each flow in verifying each feature

TRANSITIONING FROM AMS-DV TO AMS-DMS FLOW

- Alignment from other disciplines
  - For successful AMS-DMS flow execution alignment is needed from Analog Design, Digital Design, AMSV, DMSV, and Modeling teams
  - Design teams need to adopt a more Top-Down design approach
    - Define the chip-top level pins, architecture, sub-block ports
    - Define sub-block level, and Analog-Digital interactions clearly before design implementation
    - Well defined partitioning of analog, and digital blocks at chip, and sub-block level
  - A common testbench for AMS and DV is desirable
    - State of the art tools allow for the development of a common TB for AMS and DMS
    - The test bench config view determines if the analog solver is invoked or not in a given sim there by allowing the same test to be run using AMS or DMS
- Modeling and model verification
  - The integrity and reliability of DMSV depends on the availability of accurate and well-written analog models
  - Analog models should be thoroughly verified against schematics
  - The Model Vs Schematic regression should be run whenever the design changes

MORE ON MODELING

- Model Vs Schematic Verification Flow


Analog Design Schematics
Analog Architecture Requirements
Create Analog Models
Regular Model Vs Schematic Verification and Regression
Regression Pass?
No
Update Analog Model
Yes

- Models could be developed by Analog designers, Modeling engineers, or AMSV engineers
- With a top-down design approach, analog models facilitate early proof of concept of the chip architecture even before any design effort has actually begun
- Analog models could replace, or supplement design architecture definition
- Analog IP development requirements could be extended to include analog model for each IP

COMBINED AMS-DMS FLOW – SOME PIT FALLS

Case 1 – An AMSV heavy approach with very simple analog models

- Given the longer simulation times for AMSV, this approach results in very large verification cycle times
- AMSV effort becomes the bottleneck for tapeout

Case 2 – A DMSV heavy approach with very complex analog models

- In this approach even minute features like analog trims are modeled
- Given the iterative nature of analog design, this needed the models to be updated too often
- DMSV effort becomes the bottleneck as model failures delay regression closure

Case 3 – An AMSV-DMSV flow without a strong top-down design flow

- Without a strong top-down design flow, analog schematics, sub-block ports, and the analog-digital interface change frequently requiring frequent netlist, and testbench debug
- DMSV team spends a lot of time in re-netlisting the design, and in testbench debug

CONCLUSIONS

- A combined approach for SOC level functional verification of Mixed-signal designs has been presented
- Adopting a well balanced, and combined AMS-DMS verification flow will greatly reduce total time and resources spent on verification
- Having a robust analog model creation, and verification flow will ensure reliability in the coverage achieved through DMS
- Some pit falls in the suggested approach also have been presented based on some real case studies

Notes

Bill Tiffany

SigmaSense LLC
Verification Lead

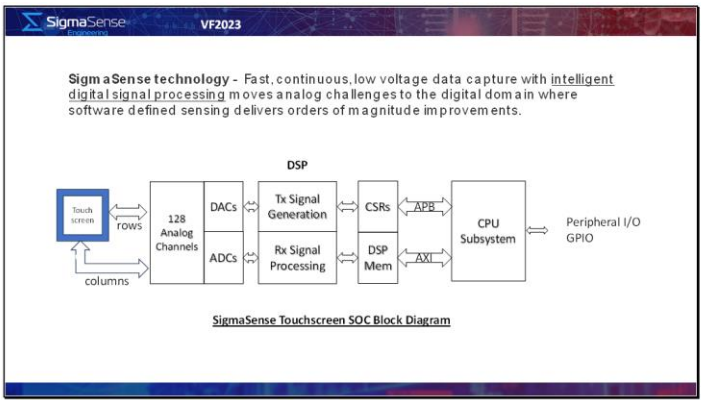
DSP Verification Using MATLAB C Models

User Paper

Abstract
The SigmaSense touchscreen controller IP uses proprietary configurable DSP logic for touchscreen signal generation and detection. MATLAB modelling is first used to refine the design to the point of a bit-width accurate resolution of each point in the signal path. This detailed definition is used to define the rtl implementation requirements. Via the MATHWORKS DPIGEN utility an equivalent C model is produced to serve as the DV predictor. A UVM testbench is created where constrained random simulations are performed, comparing the DUT and MATLAB C model results until they are in agreement.

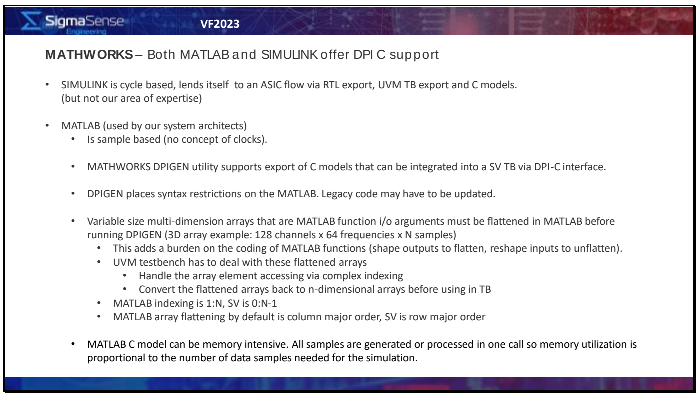
Biography
Bill Tiffany has an extensive background in digital logic design and verification including DSP applications, networking interfaces, microcontrollers, solid state drive controllers and with SigmaSense's latest touch screen controller. In addition to understanding system requirements, continuous learning and sharing within a team is key to success.

Notes



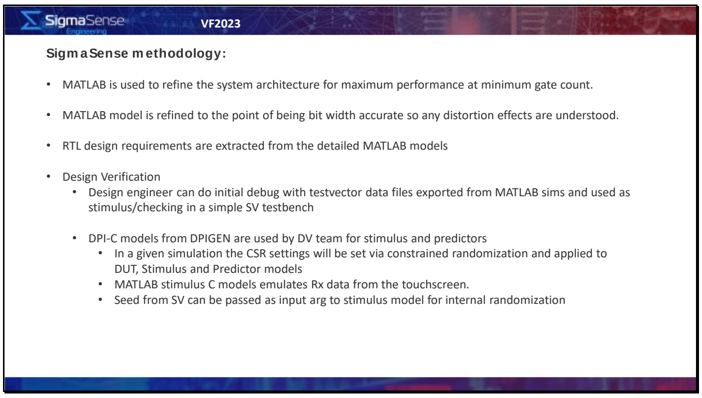
SigmaSense technology - Fast, continuous, low voltage data capture with intelligent digital signal processing moves analog challenges to the digital domain where software defined sensing delivers orders of magnitude improvements.

SigmaSense Touchscreen SOC Block Diagram


SigmaSense
Engineering
VF2023

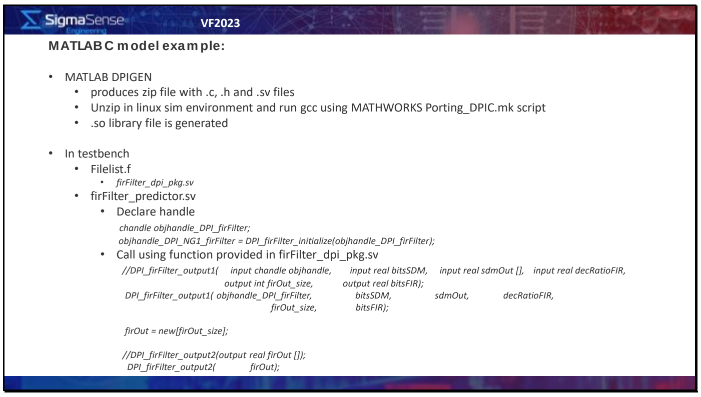
MATHWORKS – Both MATLAB and SIMULINK offer DPI C support

- SIMULINK is cycle based, lends itself to an ASIC flow via RTL export, UVM TB export and C models. (but not our area of expertise)
- MATLAB (used by our system architects)
  - Is sample based (no concept of clocks).
  - MATHWORKS DPIGEN utility supports export of C models that can be integrated into a SV TB via DPI-C interface.
  - DPIGEN places syntax restrictions on the MATLAB. Legacy code may have to be updated.
  - Variable size multi-dimension arrays that are MATLAB function i/o arguments must be flattened in MATLAB before running DPIGEN (3D array example: 128 channels x 64 frequencies x N samples)
    - This adds a burden on the coding of MATLAB functions (shape outputs to flatten, reshape inputs to unflatten).
    - UVM testbench has to deal with these flattened arrays
      - Handle the array element accessing via complex indexing
      - Convert the flattened arrays back to n-dimensional arrays before using in TB
    - MATLAB indexing is 1:N, SV is 0:N-1
    - MATLAB array flattening by default is column major order, SV is row major order
  - MATLAB C model can be memory intensive. All samples are generated or processed in one call so memory utilization is proportional to the number of data samples needed for the simulation.


SigmaSense
Engineering
VF2023

SigmaSense methodology:

- MATLAB is used to refine the system architecture for maximum performance at minimum gate count.
- MATLAB model is refined to the point of being bit width accurate so any distortion effects are understood.
- RTL design requirements are extracted from the detailed MATLAB models
- Design Verification
  - Design engineer can do initial debug with testvector data files exported from MATLAB sims and used as stimulus/checking in a simple SV testbench
  - DPI-C models from DPIGEN are used by DV team for stimulus and predictors
    - In a given simulation the CSR settings will be set via constrained randomization and applied to DUT, Stimulus and Predictor models
    - MATLAB stimulus C models emulates Rx data from the touchscreen.
    - Seed from SV can be passed as input arg to stimulus model for internal randomization


SigmaSense
Engineering
VF2023

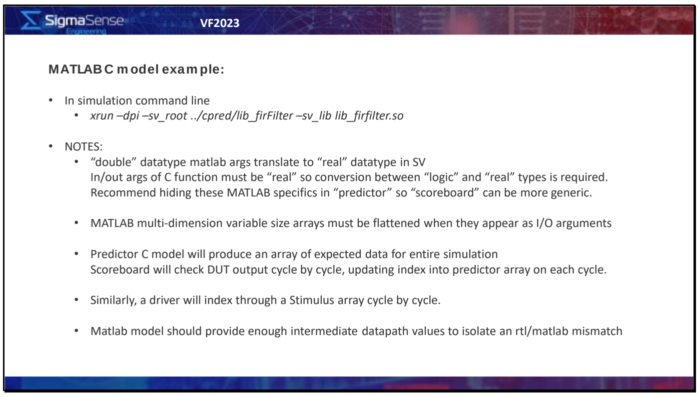
MATLAB C model example:

- MATLAB DPIGEN
  - produces zip file with .c, .h and .sv files
  - Unzip in linux sim environment and run gcc using MATHWORKS Porting_DPIC.mk script
  - .so library file is generated
- In testbench
  - Filelist.f
    - *firFilter_dpi_pkg.sv*
  - firFilter_predictor.sv
    - Declare handle

```
chandle objhandle_DPI_firFilter;
objhandle_DPI_NG1_firFilter = DPI_firFilter_initialize(objhandle_DPI_firFilter);
```

- Call using function provided in firFilter_dpi_pkg.sv

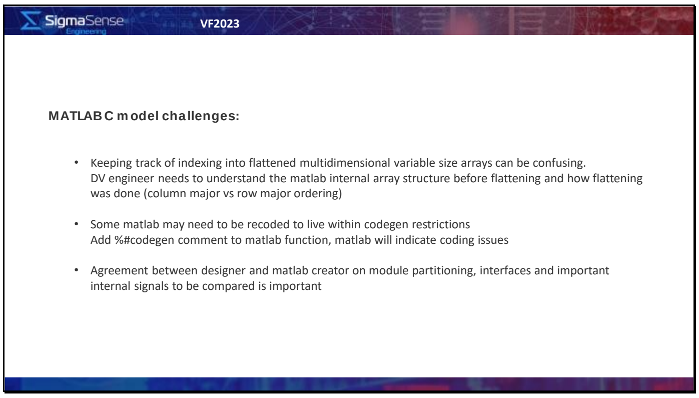
```
 //DPI_firFilter_output1(    input chandle objhandle,      input real bitsSDM,    input real sdmOut [],   input real decRatioFIR,
                            output int firOut_size,        output real bitsFIR);
  DPI_firFilter_output1( objhandle_DPI_firFilter,            bitsSDM,              sdmOut,           decRatioFIR,
                                         firOut_size,           bitsFIR);

  firOut = new[firOut_size];

 //DPI_firFilter_output2(output real firOut []);
  DPI_firFilter_output2(           firOut);
```



SigmaSense Engineering VF2023

MATLAB C model example:

- In simulation command line
  - *xrun –dpi –sv_root ../cpred/lib_firFilter –sv_lib lib_firfilter.so*
- NOTES:
  - “double” datatype matlab args translate to “real” datatype in SV
    In/out args of C function must be “real” so conversion between “logic” and “real” types is required.
    Recommend hiding these MATLAB specifics in “predictor” so “scoreboard” can be more generic.
  - MATLAB multi-dimension variable size arrays must be flattened when they appear as I/O arguments
  - Predictor C model will produce an array of expected data for entire simulation
    Scoreboard will check DUT output cycle by cycle, updating index into predictor array on each cycle.
  - Similarly, a driver will index through a Stimulus array cycle by cycle.
  - Matlab model should provide enough intermediate datapath values to isolate an rtl/matlab mismatch



SigmaSense Engineering VF2023

MATLAB C model challenges:

- Keeping track of indexing into flattened multidimensional variable size arrays can be confusing.
  DV engineer needs to understand the matlab internal array structure before flattening and how flattening was done (column major vs row major ordering)
- Some matlab may need to be recoded to live within codegen restrictions
  Add %#codegen comment to matlab function, matlab will indicate coding issues
- Agreement between designer and matlab creator on module partitioning, interfaces and important internal signals to be compared is important

Ben Delsol

UVMGen

Founder

Methodology focused testbench generation

User Paper

Abstract

UVM testbench, environment and UVC development practices need a boost. UVMGen speeds VIP creation by reactively generating code that uses only the best industry strategies. Now, recent college graduates can create environments that will withstand any guru's code review and they'll be running tests in a matter of hours, not months. At integration levels, simply click in these lower level environments to create cluster and chip level testbenches. UVMGen ensures seamless compatibility for UVC and environment reuse, making development and integration a snap.

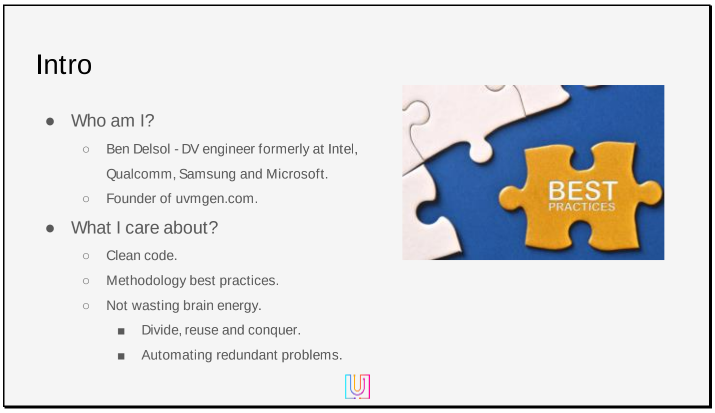
Biography

Ben Delsol has been a DV engineer with a passion for improving quality of work with automation. For more than 15 years, Ben has been a part of creating and observing industry best practices at companies such as Intel, Qualcomm, Samsung and Microsoft. He leaves his mark at these companies creating tools that transform the way work is done. Now Ben has started his own company, UVMGen LLC, which is positioned to change the way the world does DV. Never has a SystemVerilog code generation tool been created with this degree of intelligence and Ben is excited to introduce it here at the Tessolve DV Conference in Austin.

Notes

Methodology focused

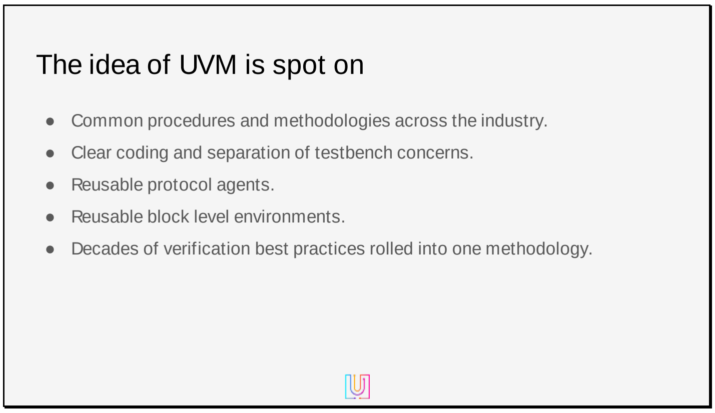
uvmgen.com



Intro

- Who am I?
  - Ben Delsol - DV engineer formerly at Intel, Qualcomm, Samsung and Microsoft.
  - Founder of uvmgen.com.
- What I care about?
  - Clean code.
  - Methodology best practices.
  - Not wasting brain energy.
    - Divide, reuse and conquer.
    - Automating redundant problems.

BEST
PRACTICES

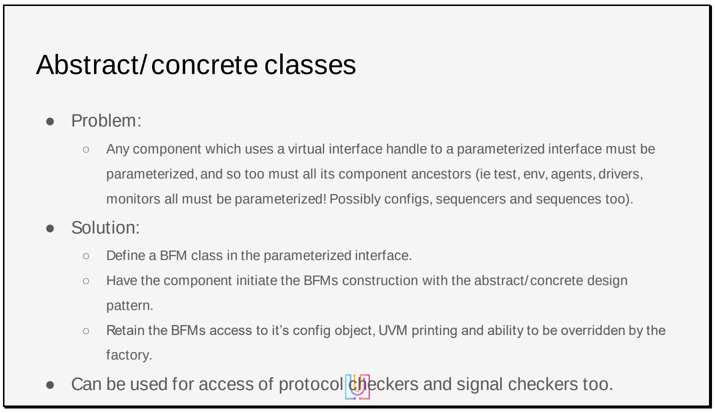


The idea of UVM is spot on

- Common procedures and methodologies across the industry.
- Clear coding and separation of testbench concerns.
- Reusable protocol agents.
- Reusable block level environments.
- Decades of verification best practices rolled into one methodology.

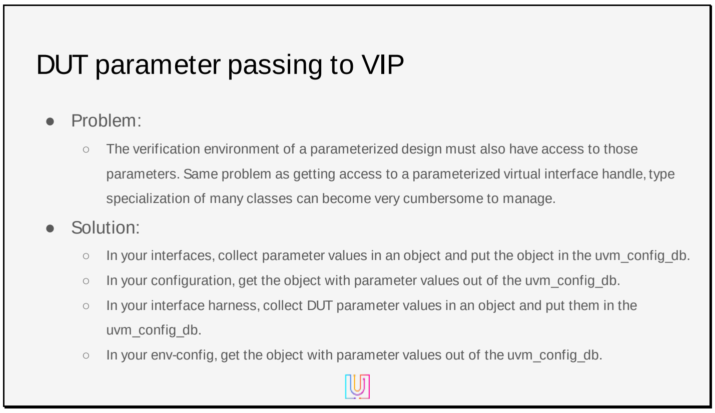
Some best practices from the last 15 years…

- Pass down config object over config db
- Use slave sequences with late response randomization
- Reset methodology: don't kill sequences with the sequencer
- Use virtual sequences over phase jumping
- No virtual sequencers
- Use objections wisely
- Use constraint policies over inheritance
- DUT parameter passing to VIP
- Interface harnesses
- Abstract/concrete classes
- Use sequence, BFM and config factory overrides
- Scale UVC, sub-env, config and TLM instances at runtime
- Conditional instantiation of static verification elements at compile time
- Use standalone testbench for UVC development
- And many more…



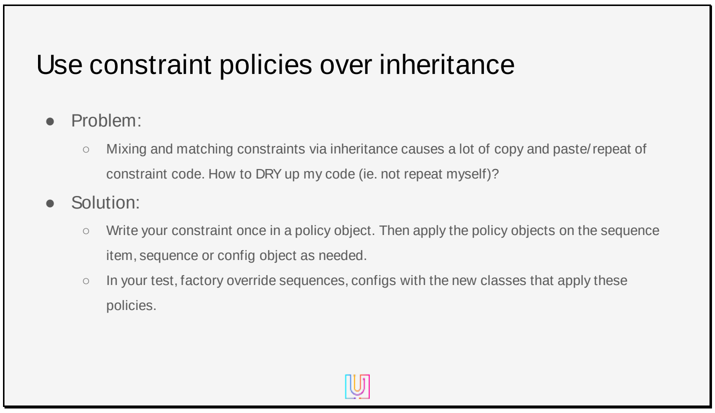
Abstract/ concrete classes

- Problem:
  - Any component which uses a virtual interface handle to a parameterized interface must be parameterized, and so too must all its component ancestors (ie test, env, agents, drivers, monitors all must be parameterized! Possibly configs, sequencers and sequences too).
- Solution:
  - Define a BFM class in the parameterized interface.
  - Have the component initiate the BFMs construction with the abstract/ concrete design pattern.
  - Retain the BFMs access to it's config object, UVM printing and ability to be overridden by the factory.
- Can be used for access of protocol checkers and signal checkers too.



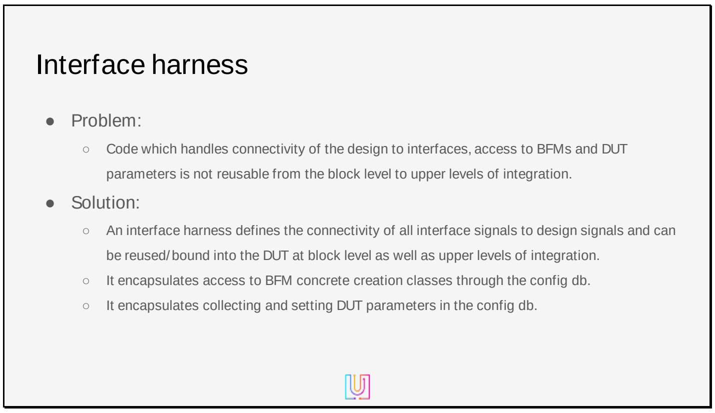
DUT parameter passing to VIP

- Problem:
  - The verification environment of a parameterized design must also have access to those parameters. Same problem as getting access to a parameterized virtual interface handle, type specialization of many classes can become very cumbersome to manage.
- Solution:
  - In your interfaces, collect parameter values in an object and put the object in the uvm_config_db.
  - In your configuration, get the object with parameter values out of the uvm_config_db.
  - In your interface harness, collect DUT parameter values in an object and put them in the uvm_config_db.
  - In your env-config, get the object with parameter values out of the uvm_config_db.



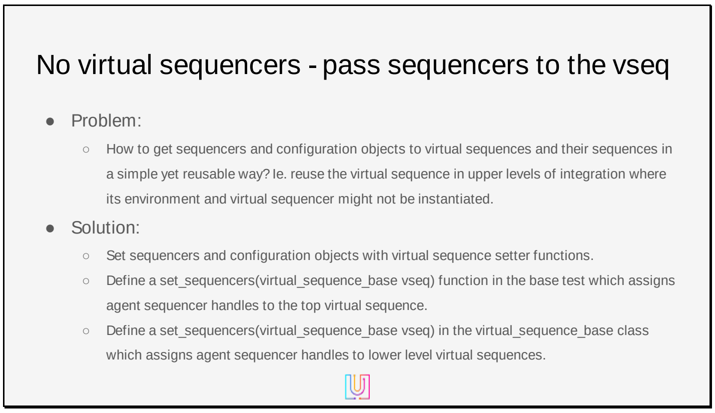
Use constraint policies over inheritance

- Problem:
  - Mixing and matching constraints via inheritance causes a lot of copy and paste/ repeat of constraint code. How to DRY up my code (ie. not repeat myself)?
- Solution:
  - Write your constraint once in a policy object. Then apply the policy objects on the sequence item, sequence or config object as needed.
  - In your test, factory override sequences, configs with the new classes that apply these policies.



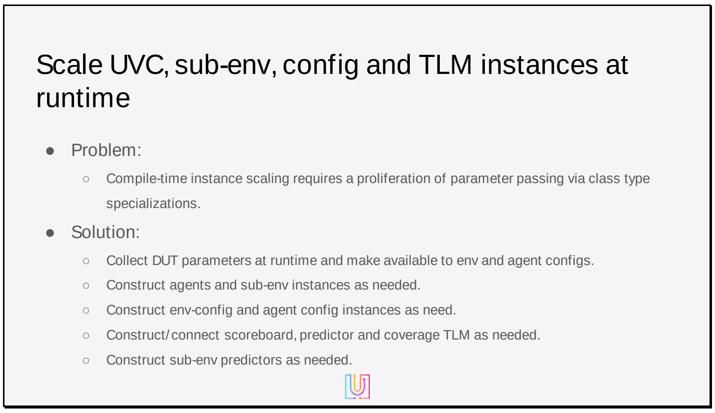
Interface harness

- Problem:
  - Code which handles connectivity of the design to interfaces, access to BFMs and DUT parameters is not reusable from the block level to upper levels of integration.
- Solution:
  - An interface harness defines the connectivity of all interface signals to design signals and can be reused/ bound into the DUT at block level as well as upper levels of integration.
  - It encapsulates access to BFM concrete creation classes through the config db.
  - It encapsulates collecting and setting DUT parameters in the config db.



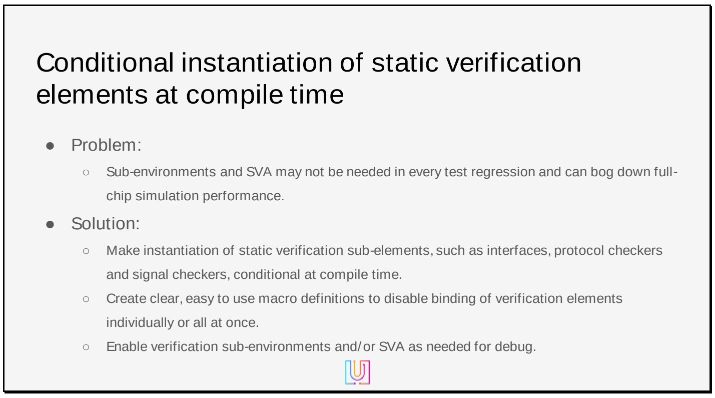
No virtual sequencers - pass sequencers to the vseq

- Problem:
  - How to get sequencers and configuration objects to virtual sequences and their sequences in a simple yet reusable way? Ie. reuse the virtual sequence in upper levels of integration where its environment and virtual sequencer might not be instantiated.
- Solution:
  - Set sequencers and configuration objects with virtual sequence setter functions.
  - Define a set_sequencers(virtual_sequence_base vseq) function in the base test which assigns agent sequencer handles to the top virtual sequence.
  - Define a set_sequencers(virtual_sequence_base vseq) in the virtual_sequence_base class which assigns agent sequencer handles to lower level virtual sequences.



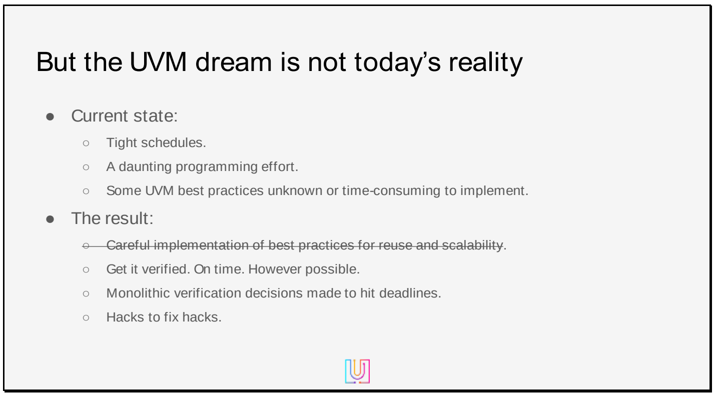
Scale UVC, sub-env, config and TLM instances at runtime

- Problem:
  - Compile-time instance scaling requires a proliferation of parameter passing via class type specializations.
- Solution:
  - Collect DUT parameters at runtime and make available to env and agent configs.
  - Construct agents and sub-env instances as needed.
  - Construct env-config and agent config instances as need.
  - Construct/connect scoreboard, predictor and coverage TLM as needed.
  - Construct sub-env predictors as needed.



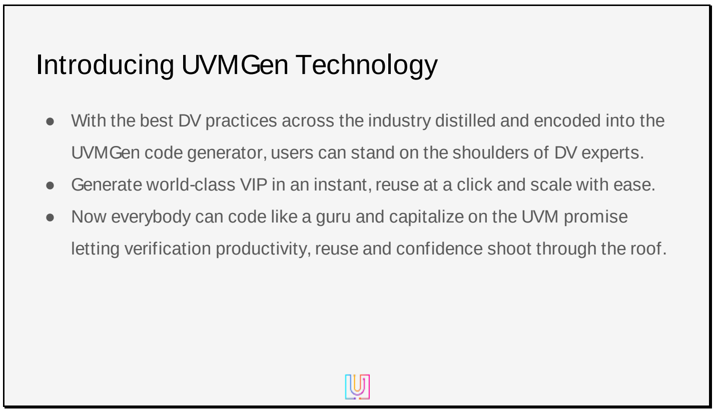
Conditional instantiation of static verification elements at compile time

- Problem:
  - Sub-environments and SVA may not be needed in every test regression and can bog down full-chip simulation performance.
- Solution:
  - Make instantiation of static verification sub-elements, such as interfaces, protocol checkers and signal checkers, conditional at compile time.
  - Create clear, easy to use macro definitions to disable binding of verification elements individually or all at once.
  - Enable verification sub-environments and/or SVA as needed for debug.



But the UVM dream is not today's reality

- Current state:
  - Tight schedules.
  - A daunting programming effort.
  - Some UVM best practices unknown or time-consuming to implement.
- The result:
  - ~~Careful implementation of best practices for reuse and scalability~~.
  - Get it verified. On time. However possible.
  - Monolithic verification decisions made to hit deadlines.
  - Hacks to fix hacks.



Introducing UVMGen Technology

- With the best DV practices across the industry distilled and encoded into the UVMGen code generator, users can stand on the shoulders of DV experts.
- Generate world-class VIP in an instant, reuse at a click and scale with ease.
- Now everybody can code like a guru and capitalize on the UVM promise letting verification productivity, reuse and confidence shoot through the roof.

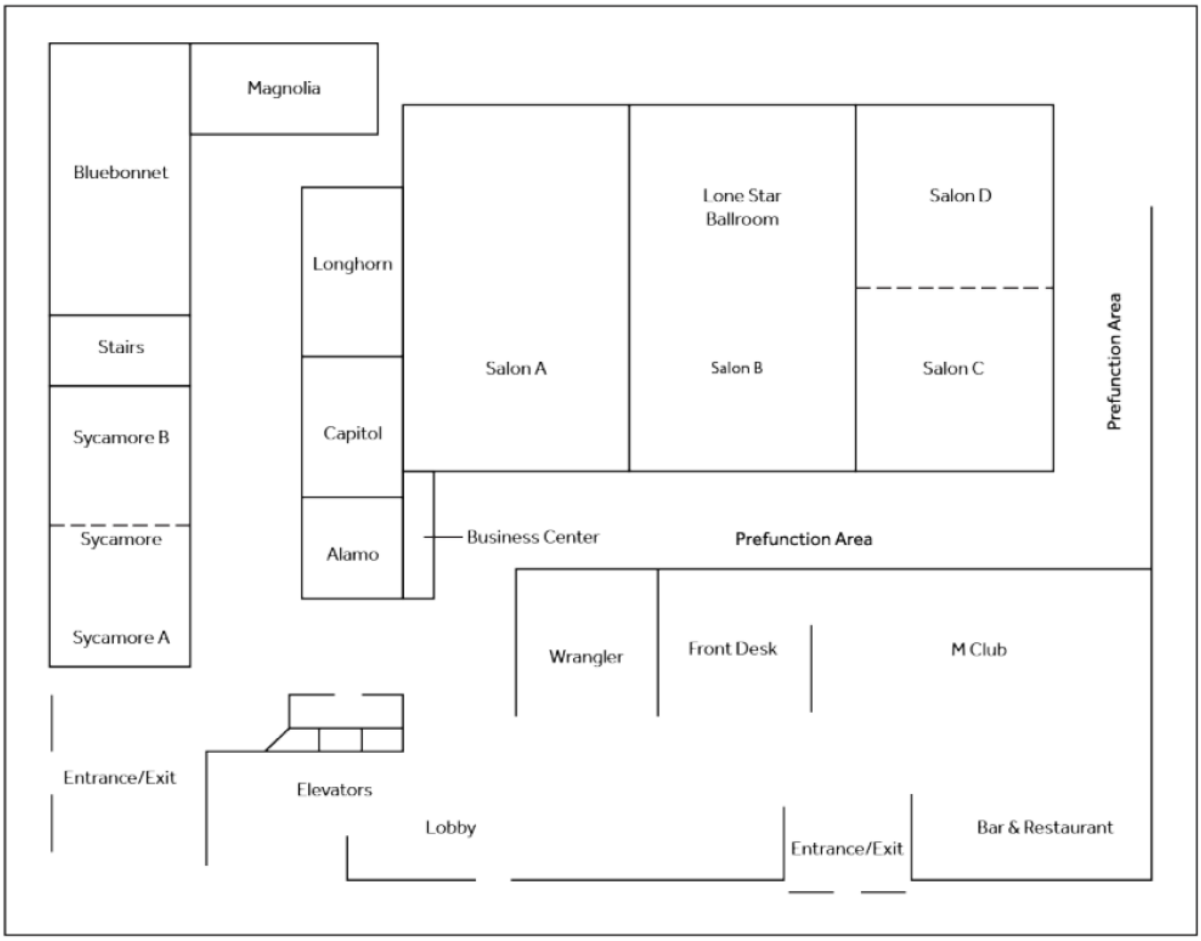
uvmgen.com

Track Session

Training Session - 2

Lonestar Ballroom – Salon C

FLOOR PLAN



We would be grateful if you could move to the track session as quickly as possible.

Doug Smith

Doulos
Engineer / Instructor

Using Non-Determinism with Formal

Gold Sponsor

Abstract

The use of non-determinism with formal is how formal is able to manage large state spaces and still arrive at a quick solution. Non-determinism plays a part in writing our formal constraints, formal targets, and formal abstractions. In this formal tutorial session, we'll explain what non-determinism is, how it's used, and show lots of examples so you can take advantage of non-determinism in verifying your designs.

Biography

Doug Smith is a verification engineer and instructor for Doulos based in the Austin Texas area with expertise in UVM and formal technologies. He has been using formal technology for several decades, performing formal verification on many kinds of designs and formal applications. Likewise, he has provided formal application support at both Jasper and Mentor/Siemens EDA. At Mentor/Siemens EDA, he served as a formal specialist and verification consultant, where he provided both formal consulting and developed an automotive functional safety formal app for performing formal fault campaigns. At Doulos, he delivers training in verification methodologies like UVM, SystemVerilog, and formal technology.

Doug holds a masters degree in Computer Engineering from the University of Cincinnati and a bachelors degree in Physics and Biology from Northern Kentucky University. Currently, he resides in Paige Texas with his wife and family on a small farm where he raises bees, cows, horses, chickens, and pigs and loves driving a tractor.

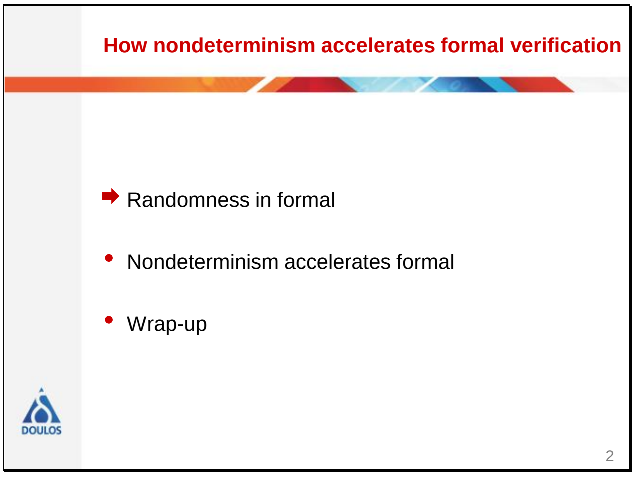
KnowHow WEBINARS

Delivering KnowHow www.doulos.com

How nondeterminism accelerates formal verification

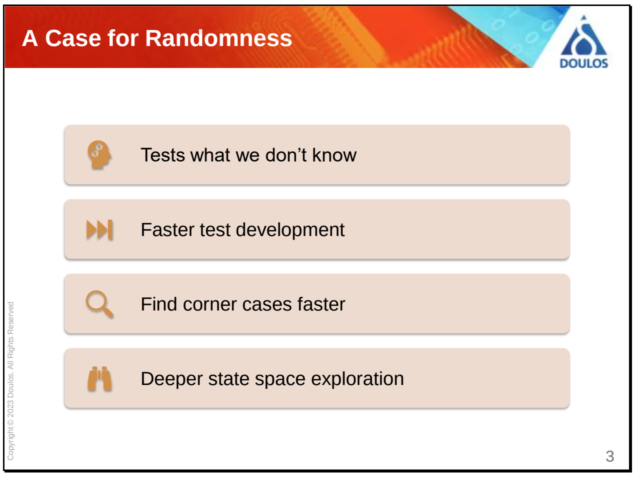
Presenter: Doug Smith
Engineer / Instructor

Webinar partner:



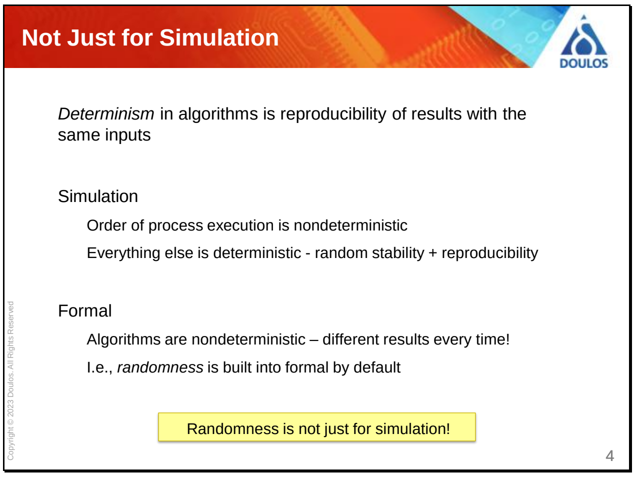
How nondeterminism accelerates formal verification

- Randomness in formal
- Nondeterminism accelerates formal
- Wrap-up



A Case for Randomness

- Tests what we don’t know
- Faster test development
- Find corner cases faster
- Deeper state space exploration



Not Just for Simulation

Determinism in algorithms is reproducibility of results with the same inputs

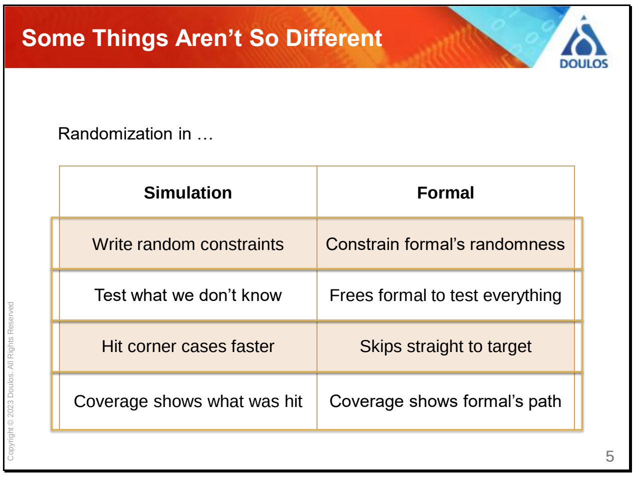
Simulation

- Order of process execution is nondeterministic
- Everything else is deterministic - random stability + reproducibility

Formal

- Algorithms are nondeterministic – different results every time!
- I.e., *randomness* is built into formal by default

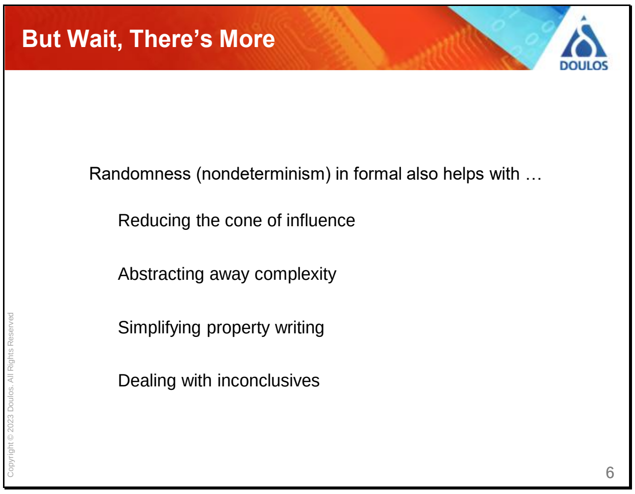
Randomness is not just for simulation!



Some Things Aren't So Different

Randomization in …

Simulation	Formal
Write random constraints	Constrain formal's randomness
Test what we don't know	Frees formal to test everything
Hit corner cases faster	Skips straight to target
Coverage shows what was hit	Coverage shows formal's path



But Wait, There's More

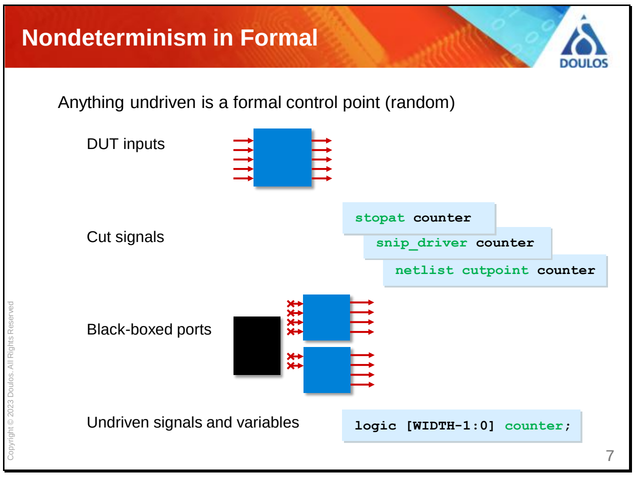
Randomness (nondeterminism) in formal also helps with …

Reducing the cone of influence

Abstracting away complexity

Simplifying property writing

Dealing with inconclusives



Nondeterminism in Formal

Anything undriven is a formal control point (random)

DUT inputs

Cut signals

```
stopat counter
```

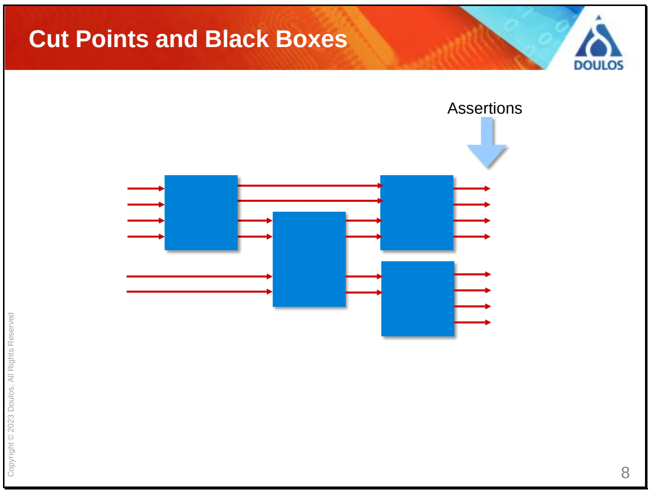
```
snip_driver counter
```

```
netlist cutpoint counter
```

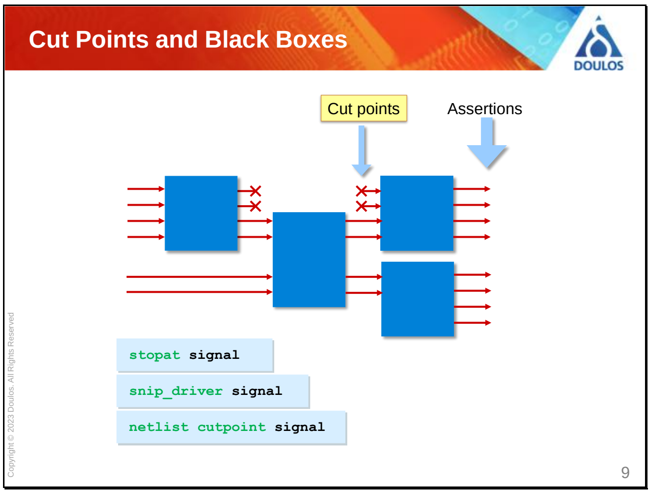
Black-boxed ports

Undriven signals and variables

```
logic [WIDTH-1:0] counter;
```

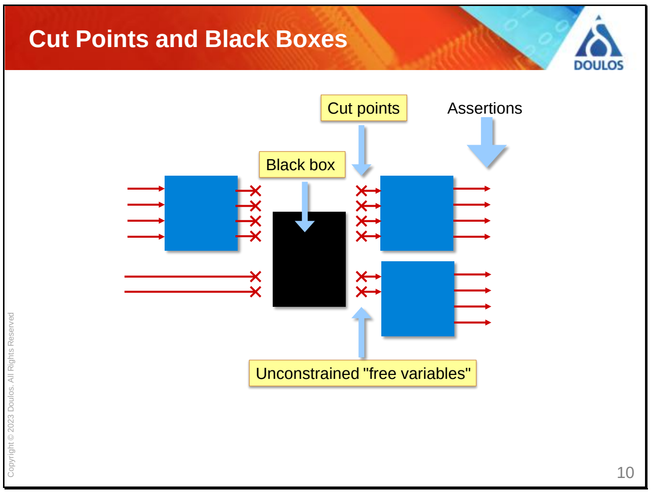


Cut Points and Black Boxes

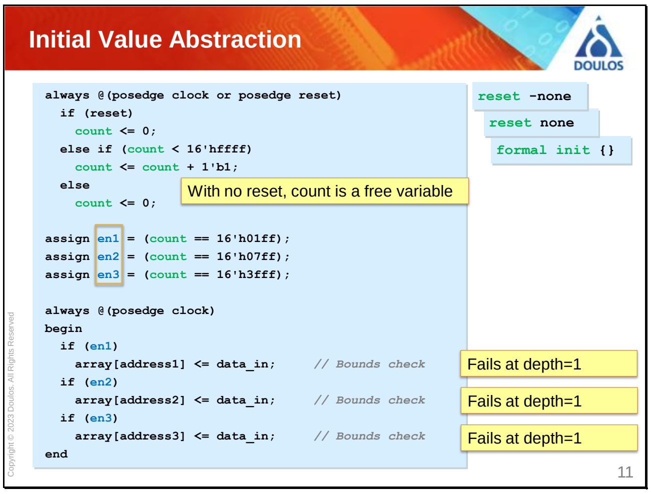


Cut Points and Black Boxes

```
stopat signal
snip_driver signal
netlist cutpoint signal
```



Cut Points and Black Boxes



Initial Value Abstraction

```
always @(posedge clock or posedge reset)
  if (reset)
    count <= 0;
  else if (count < 16'hffff)
    count <= count + 1'b1;
  else
    count <= 0;

assign en1 = (count == 16'h01ff);
assign en2 = (count == 16'h07ff);
assign en3 = (count == 16'h3fff);

always @(posedge clock)
begin
  if (en1)
    array[address1] <= data_in;     // Bounds check
  if (en2)
    array[address2] <= data_in;     // Bounds check
  if (en3)
    array[address3] <= data_in;     // Bounds check
end
```

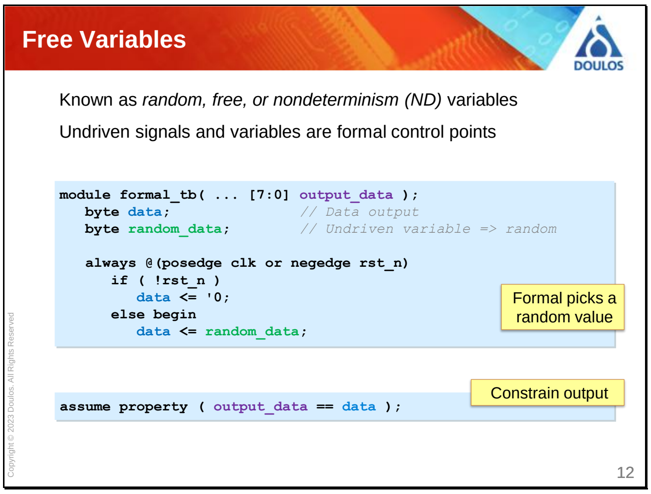
With no reset, count is a free variable

Fails at depth=1

Fails at depth=1

Fails at depth=1

```
reset -none
reset none
formal init {}
```



Free Variables

Known as *random, free, or nondeterminism (ND)* variables

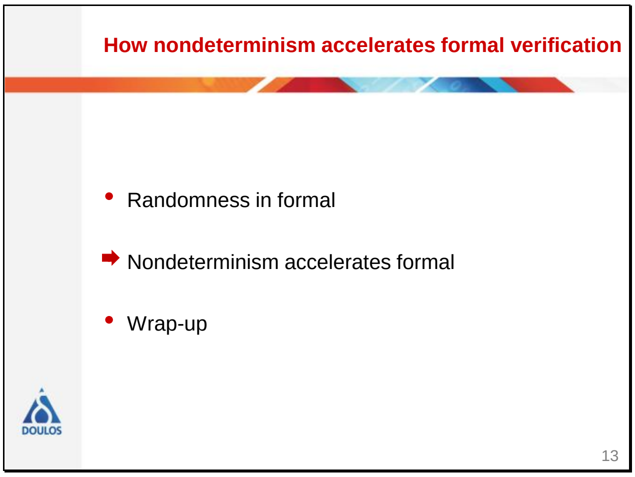
Undriven signals and variables are formal control points

```
module formal_tb( ... [7:0] output_data );
   byte data;                   // Data output
   byte random_data;            // Undriven variable => random

   always @(posedge clk or negedge rst_n)
      if ( !rst_n )
         data <= '0;
      else begin
         data <= random_data;
```

Formal picks a random value

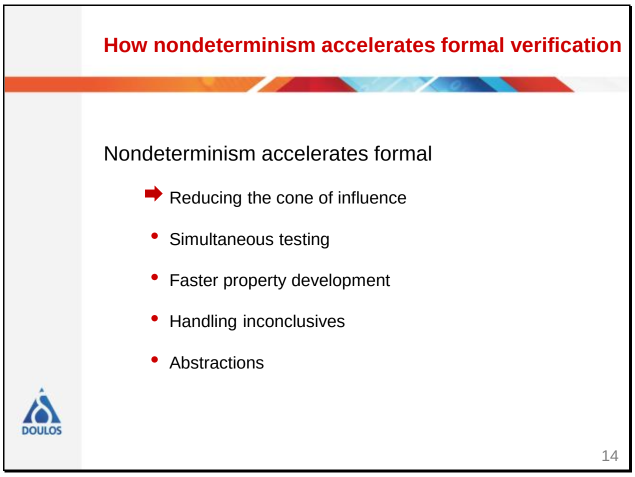
```
assume property ( output_data == data );
```

Constrain output



How nondeterminism accelerates formal verification

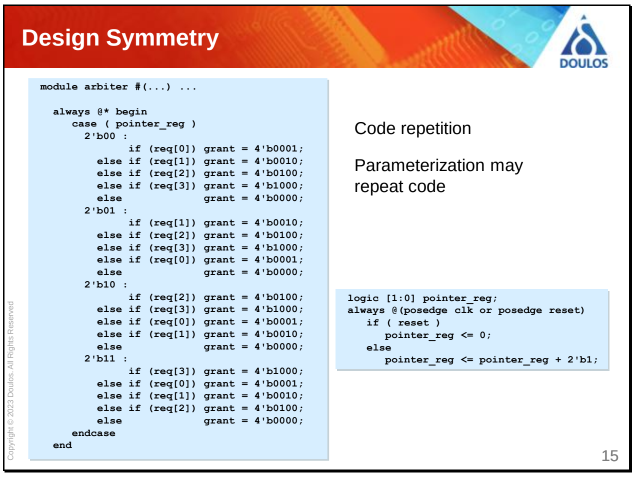
- Randomness in formal
- ➡ Nondeterminism accelerates formal
- Wrap-up



How nondeterminism accelerates formal verification

Nondeterminism accelerates formal

- ➡ Reducing the cone of influence
- Simultaneous testing
- Faster property development
- Handling inconclusives
- Abstractions



Design Symmetry

```
module arbiter #(...) ...

  always @* begin
     case ( pointer_reg )
       2'b00 :
                if (req[0]) grant = 4'b0001;
           else if (req[1]) grant = 4'b0010;
           else if (req[2]) grant = 4'b0100;
           else if (req[3]) grant = 4'b1000;
           else             grant = 4'b0000;
       2'b01 :
                if (req[1]) grant = 4'b0010;
           else if (req[2]) grant = 4'b0100;
           else if (req[3]) grant = 4'b1000;
           else if (req[0]) grant = 4'b0001;
           else             grant = 4'b0000;
       2'b10 :
                if (req[2]) grant = 4'b0100;
           else if (req[3]) grant = 4'b1000;
           else if (req[0]) grant = 4'b0001;
           else if (req[1]) grant = 4'b0010;
           else             grant = 4'b0000;
       2'b11 :
                if (req[3]) grant = 4'b1000;
           else if (req[0]) grant = 4'b0001;
           else if (req[1]) grant = 4'b0010;
           else if (req[2]) grant = 4'b0100;
           else             grant = 4'b0000;
     endcase
  end
```

Code repetition

Parameterization may repeat code

```
logic [1:0] pointer_reg;
always @(posedge clk or posedge reset)
   if ( reset )
      pointer_reg <= 0;
   else
      pointer_reg <= pointer_reg + 2'b1;
```

Data Independence

```
module arbiter #(...) ...

  always @* begin
     case ( pointer_reg )
       2'b00 :
                 if (req[0]) grant = 4'b0001;
            else if (req[1]) grant = 4'b0010;
            else if (req[2]) grant = 4'b0100;
            else if (req[3]) grant = 4'b1000;
            else             grant = 4'b0000;
       2'b01 :
                 if (req[1]) grant = 4'b0010;
            else if (req[2]) grant = 4'b0100;
            else if (req[3]) grant = 4'b1000;
            else if (req[0]) grant = 4'b0001;
            else             grant = 4'b0000;
       2'b10 :
                 if (req[2]) grant = 4'b0100;
            else if (req[3]) grant = 4'b1000;
            else if (req[0]) grant = 4'b0001;
            else if (req[1]) grant = 4'b0010;
            else             grant = 4'b0000;
       2'b11 :
                 if (req[3]) grant = 4'b1000;
            else if (req[0]) grant = 4'b0001;
            else if (req[1]) grant = 4'b0010;
            else if (req[2]) grant = 4'b0100;
            else             grant = 4'b0000;
     endcase
  end
```

Consider …

```
     if (req[0]) grant = 4'b0001;
else if (req[1]) grant = 4'b0010;
else if (req[2]) grant = 4'b0100;
else if (req[3]) grant = 4'b1000;
else             grant = 4'b0000;
```

This is equivalent to …

```
req[0] -> grant[0]
req[1] -> grant[1]
req[2] -> grant[2]
req[3] -> grant[3]
```

Or …

```
req[pointer_reg] -> grant[pointer_reg]
```

Each request and grant is independent of each other

Adding Nondeterminism

Free variable

```
module arbiter ...

  always @* begin
     case ( pointer_reg )
       2'b00 :
                 if (req[0]) grant = 4'b0001;
            else if (req[1]) grant = 4'b0010;
            else if (req[2]) grant = 4'b0100;
            else if (req[3]) grant = 4'b1000;
            else             grant = 4'b0000;
       2'b01 :
                 if (req[1]) grant = 4'b0010;
            else if (req[2]) grant = 4'b0100;
            else if (req[3]) grant = 4'b1000;
            else if (req[0]) grant = 4'b0001;
            else             grant = 4'b0000;
       2'b10 :
                 if (req[2]) grant = 4'b0100;
            else if (req[3]) grant = 4'b1000;
            else if (req[0]) grant = 4'b0001;
            else if (req[1]) grant = 4'b0010;
            else             grant = 4'b0000;
       2'b11 :
                 if (req[3]) grant = 4'b1000;
            else if (req[0]) grant = 4'b0001;
            else if (req[1]) grant = 4'b0010;
            else if (req[2]) grant = 4'b0100;
            else             grant = 4'b0000;
     endcase
  end
```

```
stopat pointer_reg
```

```
snip_driver pointer_reg
```

```
netlist cutpoint pointer_reg
```

Any starting point okay

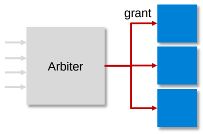
Functionality modeled by the RTL

Reduced Cone of Influence

Free variable

```
module arbiter ...

  always @* begin
     case ( pointer_reg )
       2'b00 :
                 if (req[0]) grant = 4'b0001;
            else if (req[1]) grant = 4'b0010;
            else if (req[2]) grant = 4'b0100;
            else if (req[3]) grant = 4'b1000;
            else             grant = 4'b0000;
       2'b01 :
                 if (req[1]) grant = 4'b0010;
            else if (req[2]) grant = 4'b0100;
            else if (req[3]) grant = 4'b1000;
            else if (req[0]) grant = 4'b0001;
            else             grant = 4'b0000;
       2'b10 :
                 if (req[2]) grant = 4'b0100;
            else if (req[3]) grant = 4'b1000;
            else if (req[0]) grant = 4'b0001;
            else if (req[1]) grant = 4'b0010;
            else             grant = 4'b0000;
       2'b11 :
                 if (req[3]) grant = 4'b1000;
            else if (req[0]) grant = 4'b0001;
            else if (req[1]) grant = 4'b0010;
            else if (req[2]) grant = 4'b0100;
            else             grant = 4'b0000;
     endcase
  end
```

All upstream logic removed!



```
stopat grant
```

```
snip_driver grant
```

```
netlist cutpoint grant
```

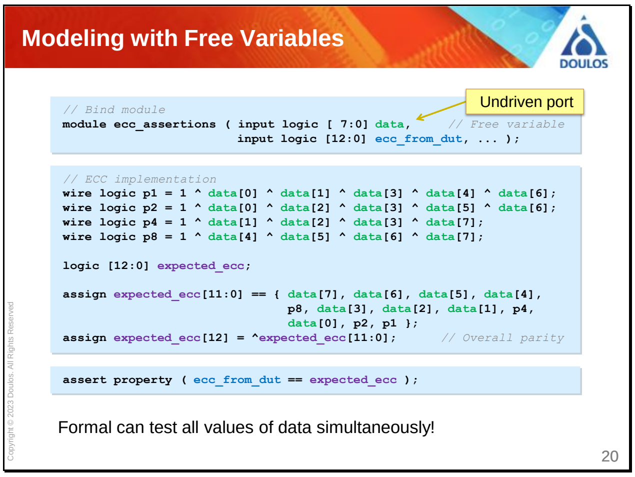
Constrain behavior

```
assume property (
       $onehot0(grant));
```

How nondeterminism accelerates formal verification

Nondeterminism accelerates formal

- Reducing the cone of influence
- ➡ Simultaneous testing
- Faster property development
- Handling inconclusives
- Abstractions



Modeling with Free Variables

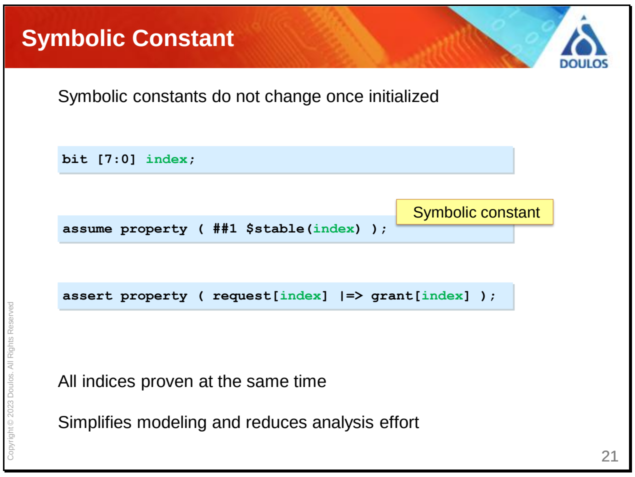
```
// Bind module
module ecc_assertions ( input logic [ 7:0] data,      // Free variable
                        input logic [12:0] ecc_from_dut, ... );
```

Undriven port

```
// ECC implementation
wire logic p1 = 1 ^ data[0] ^ data[1] ^ data[3] ^ data[4] ^ data[6];
wire logic p2 = 1 ^ data[0] ^ data[2] ^ data[3] ^ data[5] ^ data[6];
wire logic p4 = 1 ^ data[1] ^ data[2] ^ data[3] ^ data[7];
wire logic p8 = 1 ^ data[4] ^ data[5] ^ data[6] ^ data[7];

logic [12:0] expected_ecc;

assign expected_ecc[11:0] == { data[7], data[6], data[5], data[4],
                               p8, data[3], data[2], data[1], p4,
                               data[0], p2, p1 };
assign expected_ecc[12] = ^expected_ecc[11:0];      // Overall parity
```

```
assert property ( ecc_from_dut == expected_ecc );
```

Formal can test all values of data simultaneously!



Symbolic Constant

Symbolic constants do not change once initialized

```
bit [7:0] index;
```

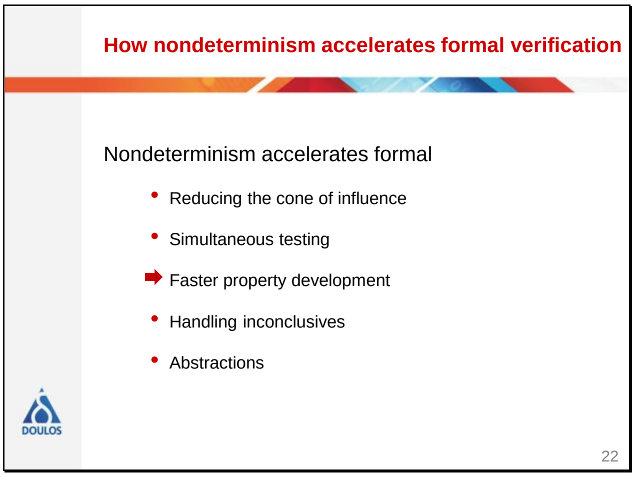
```
assume property ( ##1 $stable(index) );
```

Symbolic constant

```
assert property ( request[index] |=> grant[index] );
```

All indices proven at the same time

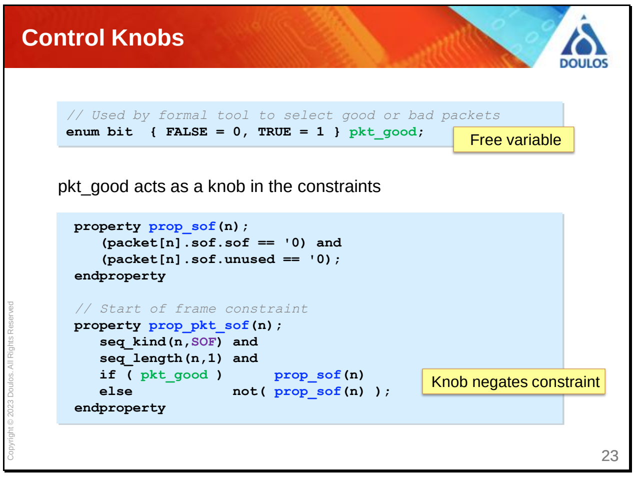
Simplifies modeling and reduces analysis effort



How nondeterminism accelerates formal verification

Nondeterminism accelerates formal

- Reducing the cone of influence
- Simultaneous testing
- ➡ Faster property development
- Handling inconclusives
- Abstractions



Control Knobs

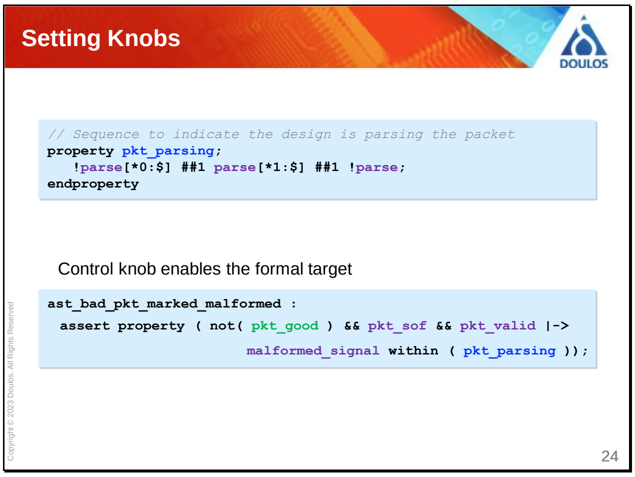
```
// Used by formal tool to select good or bad packets
enum bit  { FALSE = 0, TRUE = 1 } pkt_good;
```

Free variable

pkt_good acts as a knob in the constraints

```
property prop_sof(n);
   (packet[n].sof.sof == '0) and
   (packet[n].sof.unused == '0);
endproperty

// Start of frame constraint
property prop_pkt_sof(n);
   seq_kind(n,SOF) and
   seq_length(n,1) and
   if ( pkt_good )       prop_sof(n)
   else              not( prop_sof(n) );
endproperty
```

Knob negates constraint

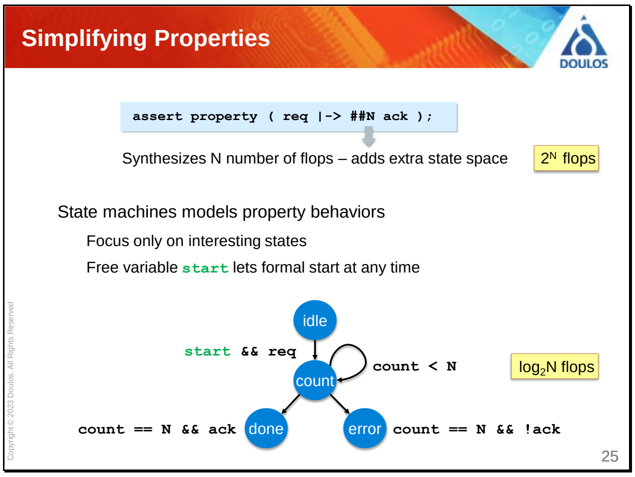


Setting Knobs

```
// Sequence to indicate the design is parsing the packet
property pkt_parsing;
   !parse[*0:$] ##1 parse[*1:$] ##1 !parse;
endproperty
```

Control knob enables the formal target

```
ast_bad_pkt_marked_malformed :
  assert property ( not( pkt_good ) && pkt_sof && pkt_valid |->
                              malformed_signal within ( pkt_parsing ));
```



Simplifying Properties

```
assert property ( req |-> ##N ack );
```

Synthesizes N number of flops – adds extra state space

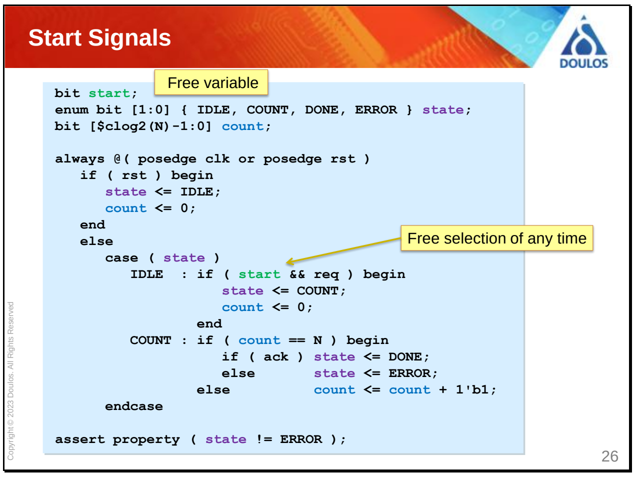
2^N flops

State machines models property behaviors

Focus only on interesting states

Free variable `start` lets formal start at any time

$\log_2 N$ flops

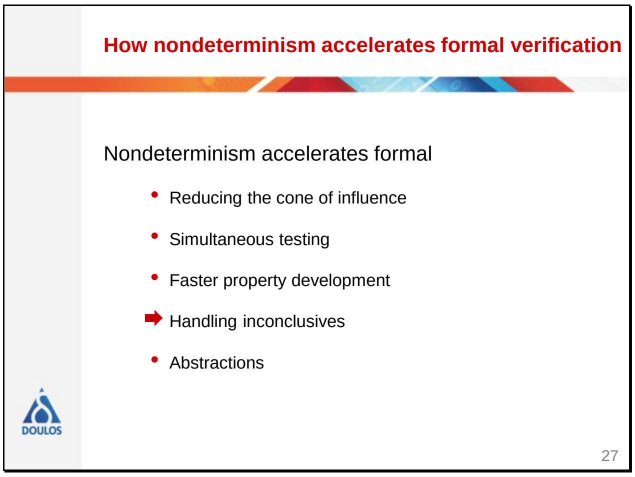


Start Signals

Free variable

```
bit start;
enum bit [1:0] { IDLE, COUNT, DONE, ERROR } state;
bit [$clog2(N)-1:0] count;

always @( posedge clk or posedge rst )
   if ( rst ) begin
      state <= IDLE;
      count <= 0;
   end
   else
      case ( state )
         IDLE  : if ( start && req ) begin
                    state <= COUNT;
                    count <= 0;
                 end
         COUNT : if ( count == N ) begin
                    if ( ack ) state <= DONE;
                    else       state <= ERROR;
                 else          count <= count + 1'b1;
      endcase

assert property ( state != ERROR );
```

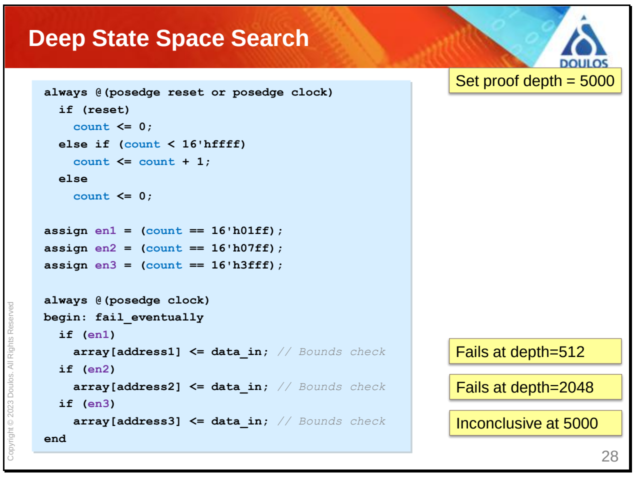
Free selection of any time



How nondeterminism accelerates formal verification

Nondeterminism accelerates formal

- Reducing the cone of influence
- Simultaneous testing
- Faster property development
- ➡ Handling inconclusives
- Abstractions



Deep State Space Search

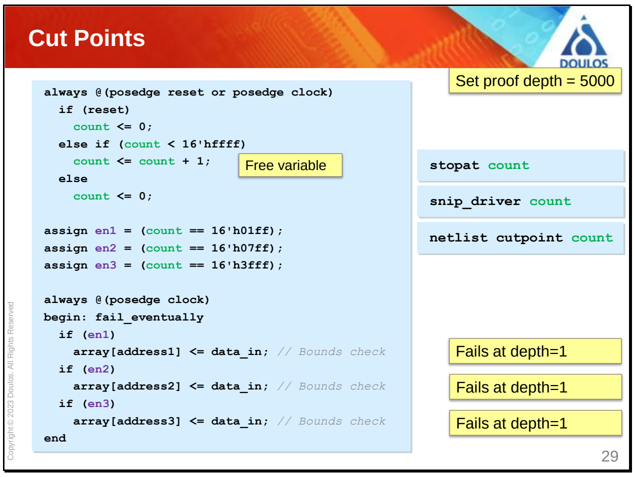
```
always @(posedge reset or posedge clock)
  if (reset)
    count <= 0;
  else if (count < 16'hffff)
    count <= count + 1;
  else
    count <= 0;

assign en1 = (count == 16'h01ff);
assign en2 = (count == 16'h07ff);
assign en3 = (count == 16'h3fff);

always @(posedge clock)
begin: fail_eventually
  if (en1)
    array[address1] <= data_in; // Bounds check
  if (en2)
    array[address2] <= data_in; // Bounds check
  if (en3)
    array[address3] <= data_in; // Bounds check
end
```

Set proof depth = 5000

Fails at depth=512

Fails at depth=2048

Inconclusive at 5000



Cut Points

```
always @(posedge reset or posedge clock)
  if (reset)
    count <= 0;
  else if (count < 16'hffff)
    count <= count + 1;
  else
    count <= 0;

assign en1 = (count == 16'h01ff);
assign en2 = (count == 16'h07ff);
assign en3 = (count == 16'h3fff);

always @(posedge clock)
begin: fail_eventually
  if (en1)
    array[address1] <= data_in; // Bounds check
  if (en2)
    array[address2] <= data_in; // Bounds check
  if (en3)
    array[address3] <= data_in; // Bounds check
end
```

Free variable

Set proof depth = 5000

```
stopat count
```

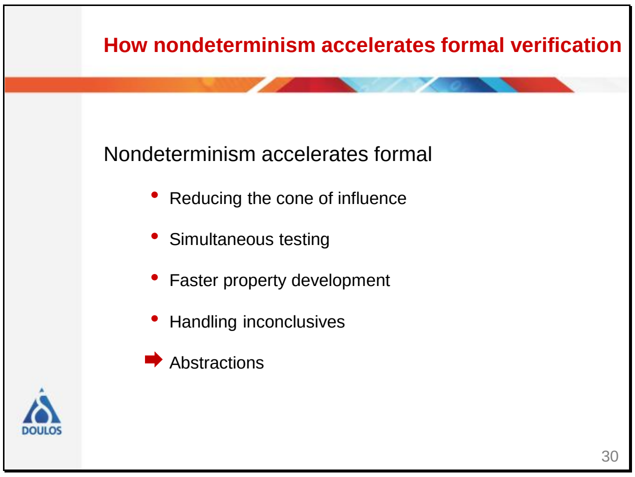
```
snip_driver count
```

```
netlist cutpoint count
```

Fails at depth=1

Fails at depth=1

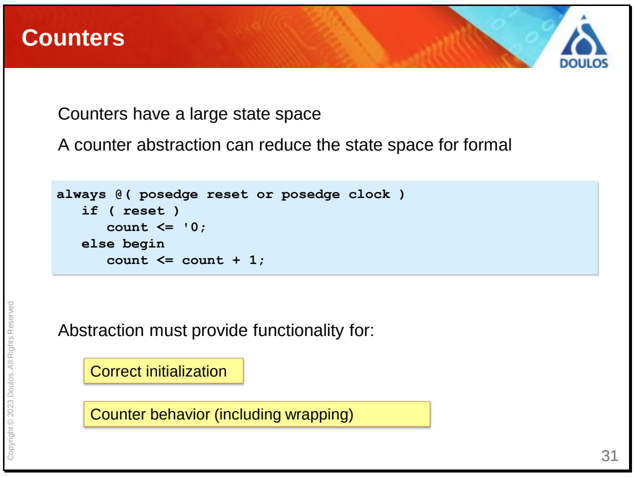
Fails at depth=1



How nondeterminism accelerates formal verification

Nondeterminism accelerates formal

- Reducing the cone of influence
- Simultaneous testing
- Faster property development
- Handling inconclusives
- ➡ Abstractions



Counters

Counters have a large state space

A counter abstraction can reduce the state space for formal

```
always @( posedge reset or posedge clock )
   if ( reset )
      count <= '0;
   else begin
      count <= count + 1;
```

Abstraction must provide functionality for:

Correct initialization

Counter behavior (including wrapping)

Counter Abstraction

Nondeterminism inserted using free variables

```
always @( posedge reset or posedge clock )
  if ( reset )
    count <= 0;
  else if ( count == MAX )
    count <= 0;
  else
    count <= count + 1;

logic [3:0] increment;

assume property (count == $past(count) + increment) % MAX);
```

Free variables

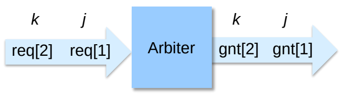
Add cut point

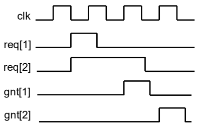
```
stopat count
```

```
snip_driver count
```

```
netlist cutpoint count
```

Priority Arbiter Example





Requirements:

j < k

req[j] has higher priority than req[k]

req[j] implies req[k] not granted until req[j] granted first

Simultaneous Checking

```
bit [$clog2(N)-1:0] j, k;
```

Free variables

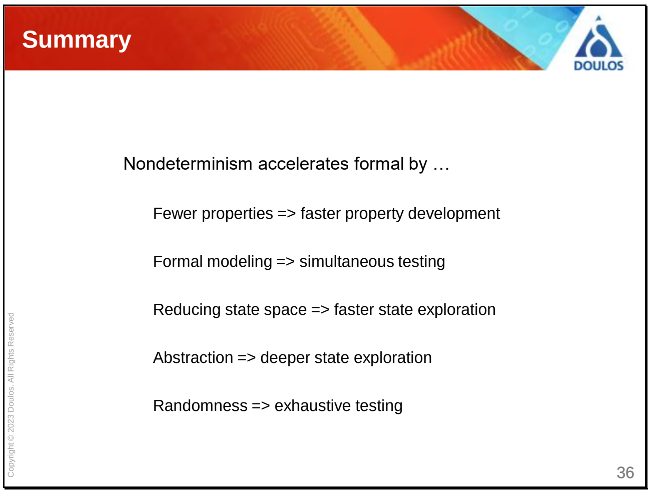
```
// Symbolic constants
assume property ( ##1 $stable(j) && $stable(k) );

// Priority encoded
assume property ( j < k );

// Keep within range
assume property ( k < N );
```

All combinations of *j* and *k* checked simultaneously!

```
assert property ( req[j] |-> !gnt[k] s_until_with gnt[j] );
```

How nondeterminism accelerates formal verification

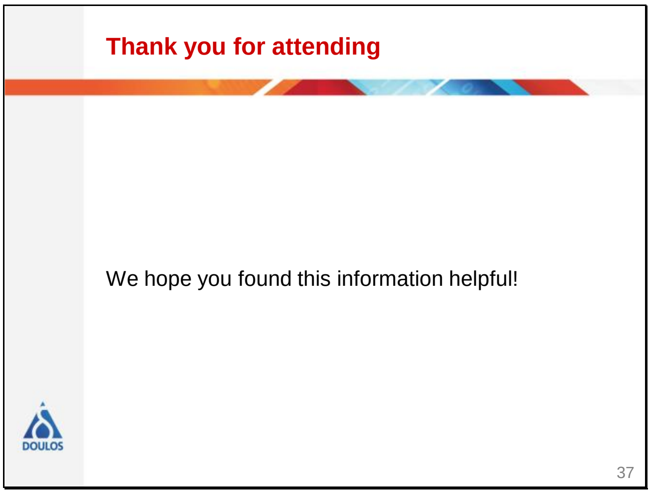
- Randomness in formal
- Nondeterminism accelerates formal
- ➡ Wrap-up



Summary

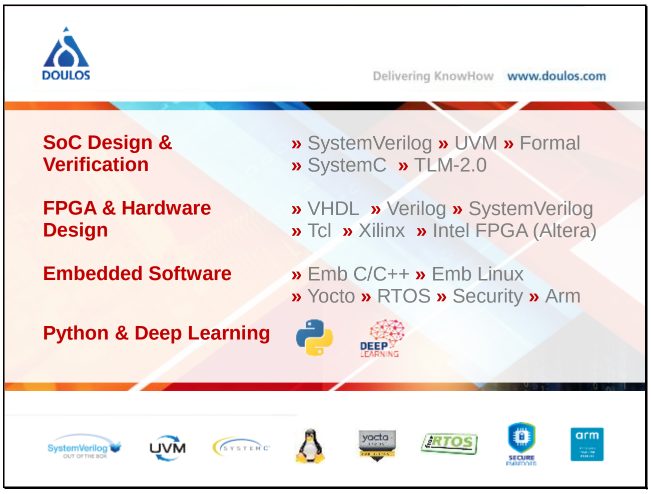
Nondeterminism accelerates formal by …

- Fewer properties => faster property development
- Formal modeling => simultaneous testing
- Reducing state space => faster state exploration
- Abstraction => deeper state exploration
- Randomness => exhaustive testing



Thank you for attending

We hope you found this information helpful!



Delivering KnowHow www.doulos.com

SoC Design & Verification » SystemVerilog » UVM » Formal » SystemC » TLM-2.0

FPGA & Hardware Design » VHDL » Verilog » SystemVerilog » Tcl » Xilinx » Intel FPGA (Altera)

Embedded Software » Emb C/C++ » Emb Linux » Yocto » RTOS » Security » Arm

Python & Deep Learning

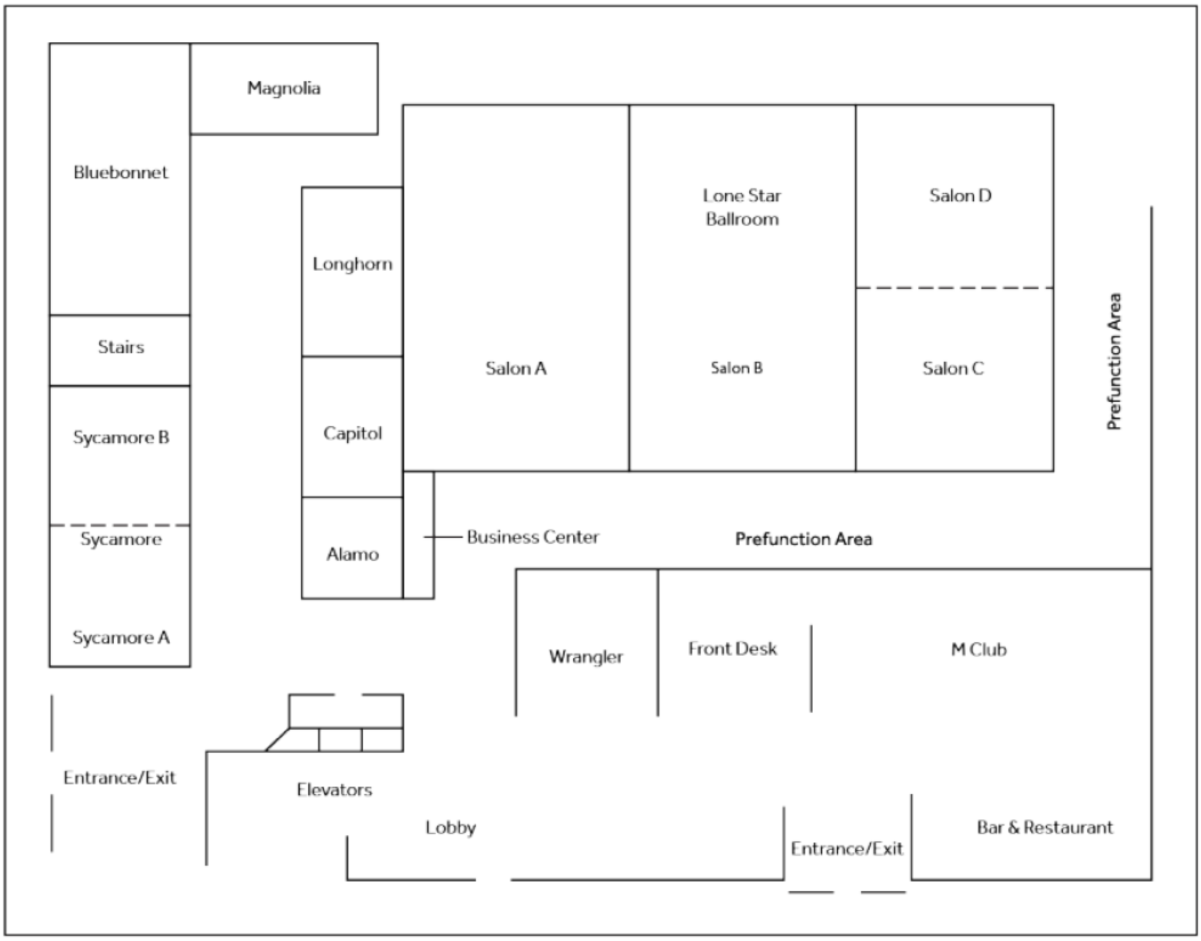
DEEP LEARNING
SystemVerilog OUT OF THE BOX
UVM
SYSTEMC
yocto
RTOS
SECURE
arm

Track Session

VHDL Verification

Lonestar Ballroom – Salon D

FLOOR PLAN



We would be grateful if you could move to the track session as quickly as possible.

Notes

Jim Lewis

SynthWorks Design Inc
VHDL Verification Specialist

OSVVM in a Nutshell, VHDL's #1 Verification Methodology

User Paper

Abstract
OSVVM is an advanced verification methodology that defines a VHDL verification framework, verification utility library, verification component library, scripting API, and co-simulation capability that simplifies your FPGA or ASIC verification project from start to finish. Using these libraries you can create a readable, powerful, and concise testbench that will boost productivity for either low level block tests (unit tests) or complex FPGA and ASIC tests.

OSVVM is developed by the same VHDL experts who have helped develop VHDL standards. We have used our expert VHDL skills to create advanced verification capabilities that provide:

- A structured transaction-based verification framework using verification components.
- A common, shared transaction API for address bus (AXI4, Avalon, …) and streaming (AXI Stream, UART) verification components.
- Improved readability and reviewability by the whole team including software and system engineers.
- Improved reuse and reduced project schedules.
- Buzz word features including Constrained Random, Functional Coverage, Scoreboards, FIFOs, Memory Models, error logging and reporting, and message filtering that are simple to use and work like built-in language features.
- A common scripting API to run all simulators. OSVVM scripting supports GHDL, NVC, Aldec Riviera-PRO and ActiveHDL, Siemens Questa and ModelSim, Synopsys VCS, and Cadence Xcelium.

- A Co-simulation capability that supports running software (C++) in a hardware simulation environment.
- Unmatched test reporting with HTML based test suite reports, test case reports, and logs that facilitate debug and test artifact collection.
- Support for continuous integration (CI/CD) with JUnit XML test suite reporting.
- A rival to the verification capabilities of SystemVerilog + UVM.

OSVVM has grown rapidly during the COVID years, giving us better capability, better test reporting (HTML and Junit), and scripting that is simple to use (and works with most VHDL simulators). This presentation will show how these advances fit into the overall OSVVM Methodology.

Looking to improve your VHDL verification methodology? OSVVM provides a complete solution for VHDL ASIC or FPGA verification. There is no new language to learn. It is simple, powerful, and concise. Any VHDL engineer can write either tests or verification components.

Each piece/capability of OSVVM can be used separately. Hence, you can learn and adopt pieces as you need them.

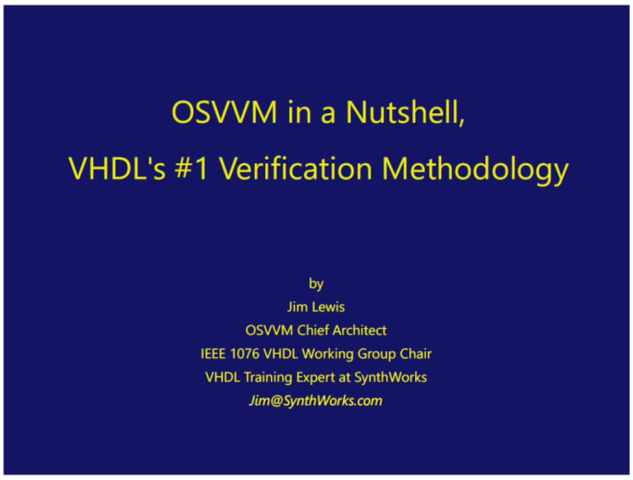
Maybe your EDA vendor has suggested that you should be using SystemVerilog for verification. According to the 2022 Wilson Verification Survey [1], for both FPGA design and verification, VHDL is used more often than Verilog or SystemVerilog. Likewise, in the survey you will find that OSVVM is the #1 VHDL verification methodology.

[1] https://blogs.sw.siemens.com/verificationhorizons/2022/11/21/part-6-the-2022-wilson-research-group-functional-verification-study/

Biography

Jim Lewis is an innovator and leader in the VHDL community. He has 30 plus years of design and teaching experience. He is the Chair of the IEEE 1076 VHDL Standards Working Group. He is a co-founder of the Open Source VHDL Verification Methodology (OSVVM) and the chief architect of the packages and methodology. He is an expert VHDL trainer for SynthWorks Design Inc. In his design practice, he has created designs for print servers, IMA E1/T1 networking, fighter jets, video phones, and space craft.

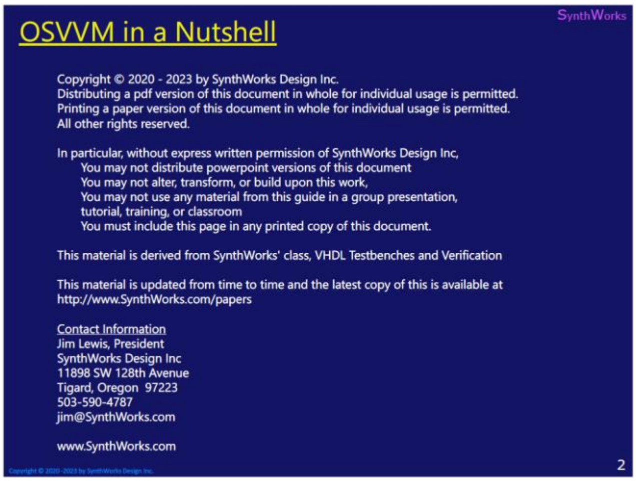
Whether teaching, developing OSVVM, consulting on VHDL design and verification projects, or working on the IEEE VHDL standard, Mr Lewis brings a deep understanding of VHDL to architect solutions that solve difficult problems in simple ways.



OSVVM in a Nutshell,

VHDL's #1 Verification Methodology

by
Jim Lewis
OSVVM Chief Architect
IEEE 1076 VHDL Working Group Chair
VHDL Training Expert at SynthWorks
Jim@SynthWorks.com



SynthWorks

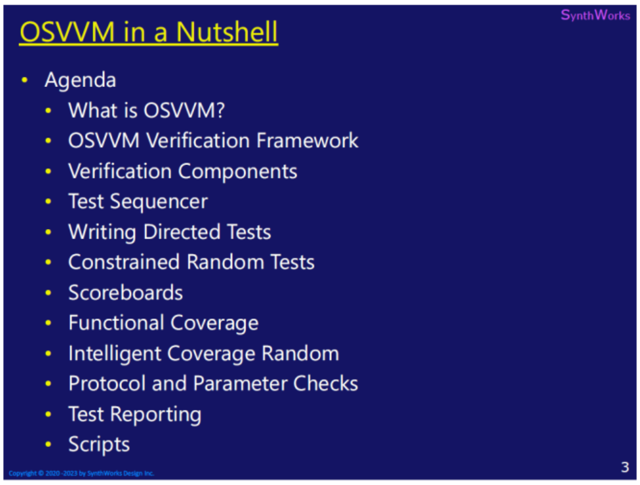
OSVVM in a Nutshell

This material is derived from SynthWorks' class, VHDL Testbenches and Verification

This material is updated from time to time and the latest copy of this is available at http://www.SynthWorks.com/papers

Contact Information
Jim Lewis, President
SynthWorks Design Inc
11898 SW 128th Avenue
Tigard, Oregon 97223
503-590-4787
jim@SynthWorks.com

www.SynthWorks.com



SynthWorks

OSVVM in a Nutshell

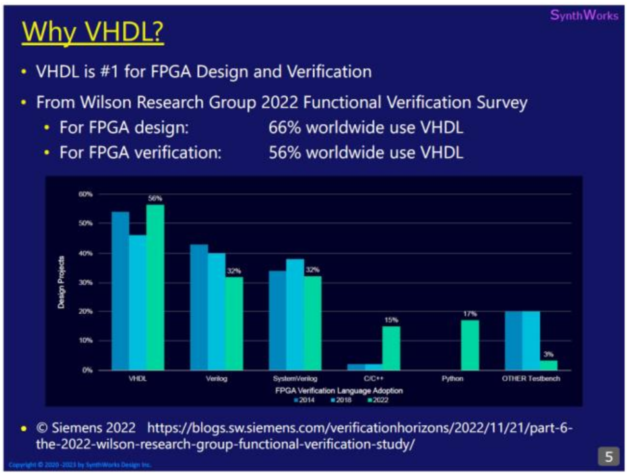
- Agenda
  - What is OSVVM?
  - OSVVM Verification Framework
  - Verification Components
  - Test Sequencer
  - Writing Directed Tests
  - Constrained Random Tests
  - Scoreboards
  - Functional Coverage
  - Intelligent Coverage Random
  - Protocol and Parameter Checks
  - Test Reporting
  - Scripts

SynthWorks

Background

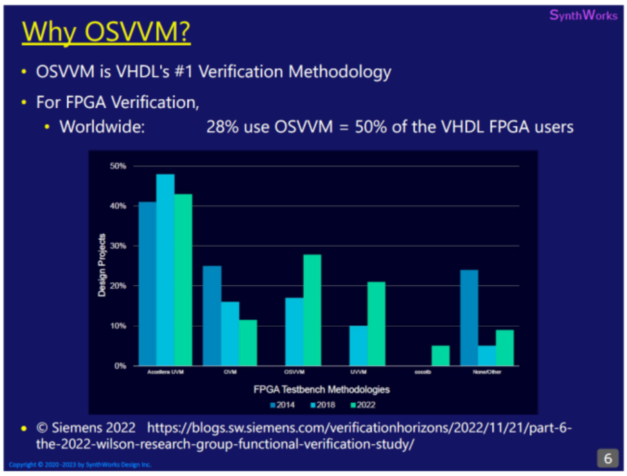
- About Jim Lewis
  - 30 years: VHDL Design and Verification
  - 20+ years: Active in IEEE VHDL WGs
  - 15+ years: IEEE VHDL WG chair
  - Chief Architect of OSVVM
  - VHDL Consultant and Trainer for SynthWorks
- SynthWorks provides VHDL Training
  - Comprehensive VHDL Introduction
  - VHDL Coding for Synthesis
  - Advanced VHDL Testbenches and Verification – OSVVM Bootcamp

OSVVM is developed by the same VHDL experts who have helped develop VHDL standards.



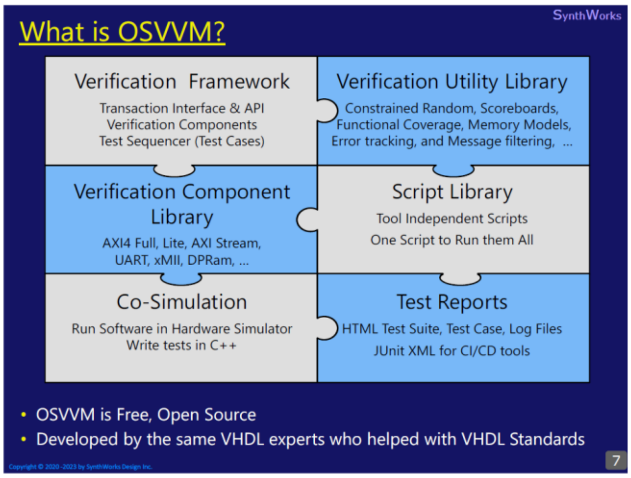
Why VHDL?

- VHDL is #1 for FPGA Design and Verification
- From Wilson Research Group 2022 Functional Verification Survey
  - For FPGA design: 66% worldwide use VHDL
  - For FPGA verification: 56% worldwide use VHDL
- © Siemens 2022 https://blogs.sw.siemens.com/verificationhorizons/2022/11/21/part-6-the-2022-wilson-research-group-functional-verification-study/



Why OSVVM?

- OSVVM is VHDL's #1 Verification Methodology
- For FPGA Verification,
  - Worldwide: 28% use OSVVM = 50% of the VHDL FPGA users
- © Siemens 2022 https://blogs.sw.siemens.com/verificationhorizons/2022/11/21/part-6-the-2022-wilson-research-group-functional-verification-study/



What is OSVVM?

Verification Framework
Transaction Interface & API
Verification Components
Test Sequencer (Test Cases)

Verification Utility Library
Constrained Random, Scoreboards, Functional Coverage, Memory Models, Error tracking, and Message filtering, …

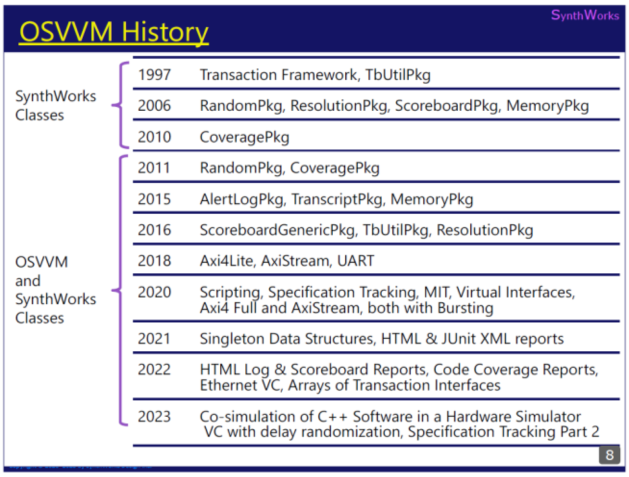
Verification Component Library
AXI4 Full, Lite, AXI Stream, UART, xMII, DPRam, …

Script Library
Tool Independent Scripts
One Script to Run them All

Co-Simulation
Run Software in Hardware Simulator
Write tests in C++

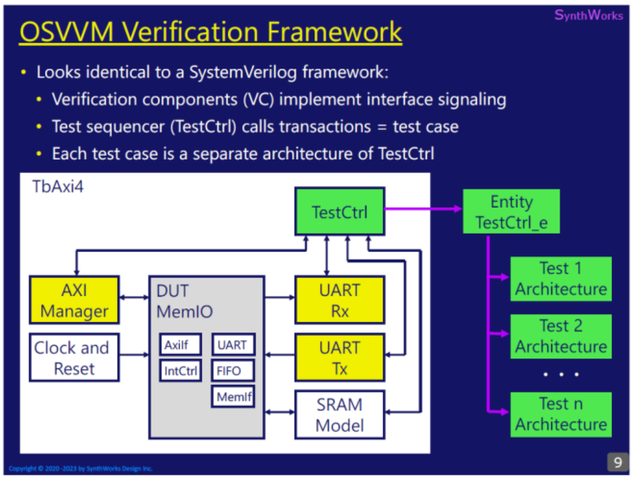
Test Reports
HTML Test Suite, Test Case, Log Files
JUnit XML for CI/CD tools

- OSVVM is Free, Open Source
- Developed by the same VHDL experts who helped with VHDL Standards


SynthWorks

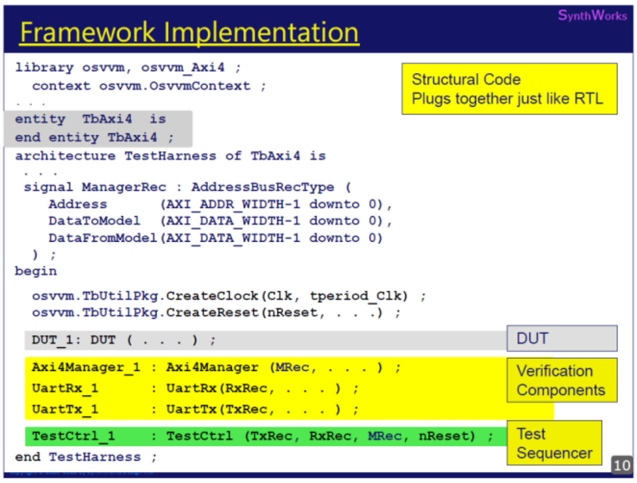
OSVVM History

SynthWorks Classes	1997	Transaction Framework, TbUtilPkg
	2006	RandomPkg, ResolutionPkg, ScoreboardPkg, MemoryPkg
	2010	CoveragePkg
OSVVM and SynthWorks Classes	2011	RandomPkg, CoveragePkg
	2015	AlertLogPkg, TranscriptPkg, MemoryPkg
	2016	ScoreboardGenericPkg, TbUtilPkg, ResolutionPkg
	2018	Axi4Lite, AxiStream, UART
	2020	Scripting, Specification Tracking, MIT, Virtual Interfaces, Axi4 Full and AxiStream, both with Bursting
	2021	Singleton Data Structures, HTML & JUnit XML reports
	2022	HTML Log & Scoreboard Reports, Code Coverage Reports, Ethernet VC, Arrays of Transaction Interfaces
	2023	Co-simulation of C++ Software in a Hardware Simulator VC with delay randomization, Specification Tracking Part 2



OSVVM Verification Framework

- Looks identical to a SystemVerilog framework:
  - Verification components (VC) implement interface signaling
  - Test sequencer (TestCtrl) calls transactions = test case
  - Each test case is a separate architecture of TestCtrl



Framework Implementation

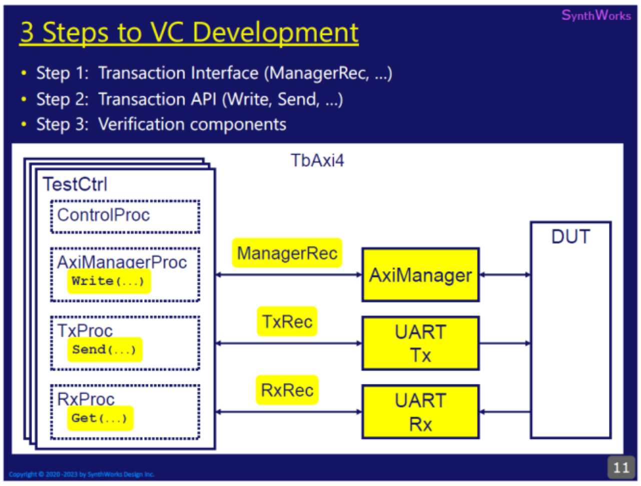
```
library osvvm, osvvm_Axi4 ;
  context osvvm.OsvvmContext ;
. . .
entity  TbAxi4  is
end entity TbAxi4 ;
architecture TestHarness of TbAxi4 is
 . . .
 signal ManagerRec : AddressBusRecType (
    Address       (AXI_ADDR_WIDTH-1 downto 0),
    DataToModel   (AXI_DATA_WIDTH-1 downto 0),
    DataFromModel (AXI_DATA_WIDTH-1 downto 0)
  ) ;
begin

  osvvm.TbUtilPkg.CreateClock(Clk, tperiod_Clk) ;
  osvvm.TbUtilPkg.CreateReset(nReset, . . .) ;

  DUT_1: DUT ( . . . ) ;

  Axi4Manager_1 : Axi4Manager (MRec, . . . ) ;
  UartRx_1      : UartRx(RxRec, . . . ) ;
  UartTx_1      : UartTx(TxRec, . . . ) ;

  TestCtrl_1    : TestCtrl (TxRec, RxRec, MRec, nReset) ;
end TestHarness ;
```

Structural Code
Plugs together just like RTL

DUT

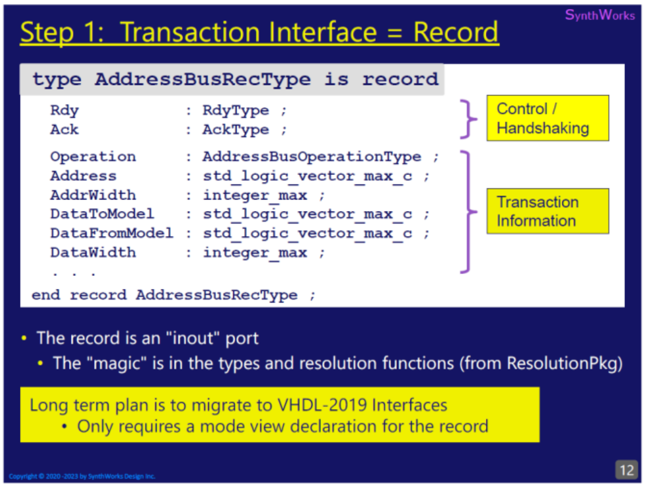
Verification Components

Test Sequencer



3 Steps to VC Development

- Step 1: Transaction Interface (ManagerRec, ...)
- Step 2: Transaction API (Write, Send, ...)
- Step 3: Verification components



Step 1: Transaction Interface = Record

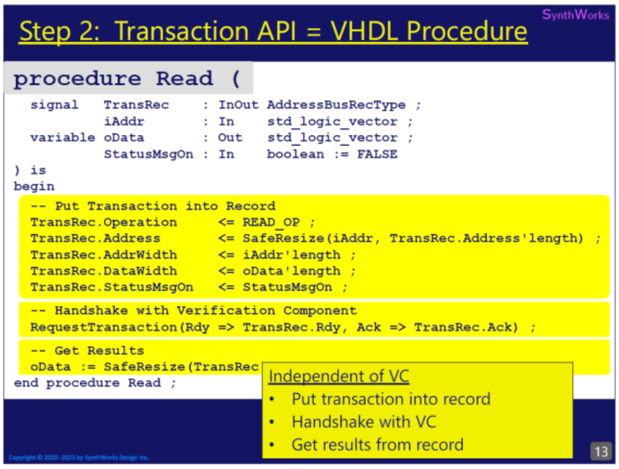
```
type AddressBusRecType is record
  Rdy           : RdyType ;
  Ack           : AckType ;

  Operation     : AddressBusOperationType ;
  Address       : std_logic_vector_max_c ;
  AddrWidth     : integer_max ;
  DataToModel   : std_logic_vector_max_c ;
  DataFromModel : std_logic_vector_max_c ;
  DataWidth     : integer_max ;
  . . .
end record AddressBusRecType ;
```

Control / Handshaking

Transaction Information

- The record is an "inout" port
  - The "magic" is in the types and resolution functions (from ResolutionPkg)

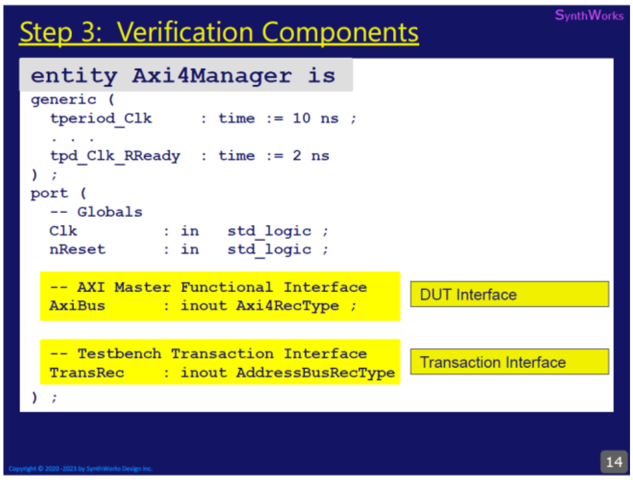
Long term plan is to migrate to VHDL-2019 Interfaces
- Only requires a mode view declaration for the record



Step 2: Transaction API = VHDL Procedure

```
procedure Read (
  signal   TransRec    : InOut AddressBusRecType ;
           iAddr       : In    std_logic_vector ;
  variable oData       : Out   std_logic_vector ;
           StatusMsgOn : In    boolean := FALSE
) is
begin
  -- Put Transaction into Record
  TransRec.Operation     <= READ_OP ;
  TransRec.Address       <= SafeResize(iAddr, TransRec.Address'length) ;
  TransRec.AddrWidth     <= iAddr'length ;
  TransRec.DataWidth     <= oData'length ;
  TransRec.StatusMsgOn   <= StatusMsgOn ;
  -- Handshake with Verification Component
  RequestTransaction(Rdy => TransRec.Rdy, Ack => TransRec.Ack) ;
  -- Get Results
  oData := SafeResize(TransRec
end procedure Read ;
```

Independent of VC
- Put transaction into record
- Handshake with VC
- Get results from record

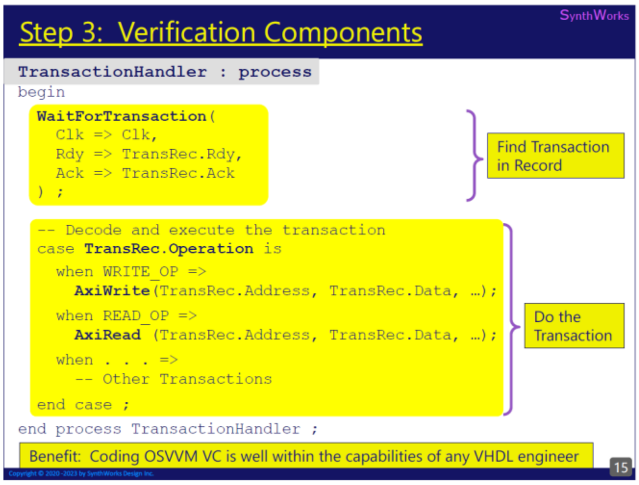


Step 3: Verification Components

```
entity Axi4Manager is
generic (
  tperiod_Clk     : time := 10 ns ;
  . . .
  tpd_Clk_RReady  : time := 2 ns
) ;
port (
  -- Globals
  Clk          : in    std_logic ;
  nReset       : in    std_logic ;

  -- AXI Master Functional Interface
  AxiBus       : inout Axi4RecType ;

  -- Testbench Transaction Interface
  TransRec     : inout AddressBusRecType
) ;
```

DUT Interface

Transaction Interface



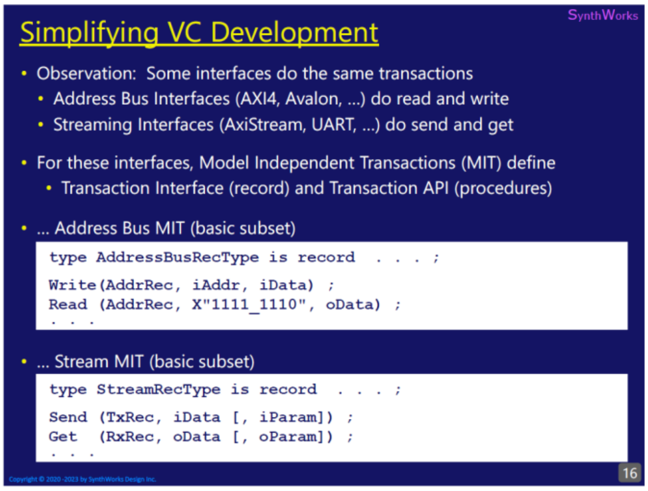
Step 3: Verification Components

```
TransactionHandler : process
begin
  WaitForTransaction(
    Clk => Clk,
    Rdy => TransRec.Rdy,
    Ack => TransRec.Ack
  ) ;

  -- Decode and execute the transaction
  case TransRec.Operation is
    when WRITE_OP =>
      AxiWrite(TransRec.Address, TransRec.Data, …);
    when READ_OP =>
      AxiRead (TransRec.Address, TransRec.Data, …);
    when . . . =>
      -- Other Transactions
  end case ;
end process TransactionHandler ;
```

Find Transaction in Record

Do the Transaction

Benefit: Coding OSVVM VC is well within the capabilities of any VHDL engineer



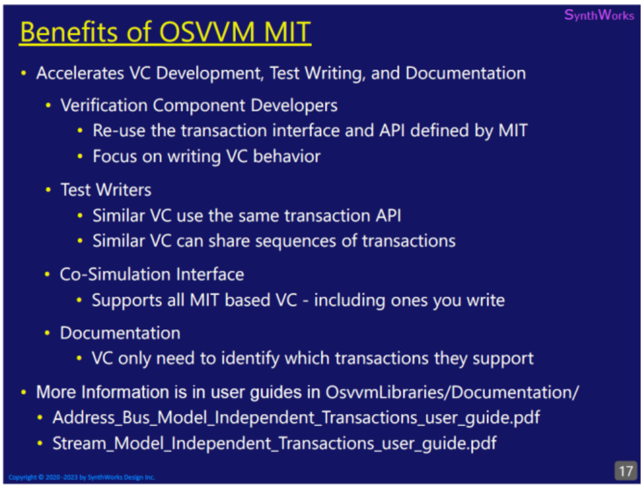
Simplifying VC Development

- Observation: Some interfaces do the same transactions
  - Address Bus Interfaces (AXI4, Avalon, …) do read and write
  - Streaming Interfaces (AxiStream, UART, …) do send and get
- For these interfaces, Model Independent Transactions (MIT) define
  - Transaction Interface (record) and Transaction API (procedures)
- … Address Bus MIT (basic subset)

```
type AddressBusRecType is record  . . . ;
Write(AddrRec, iAddr, iData) ;
Read (AddrRec, X"1111_1110", oData) ;
. . .
```

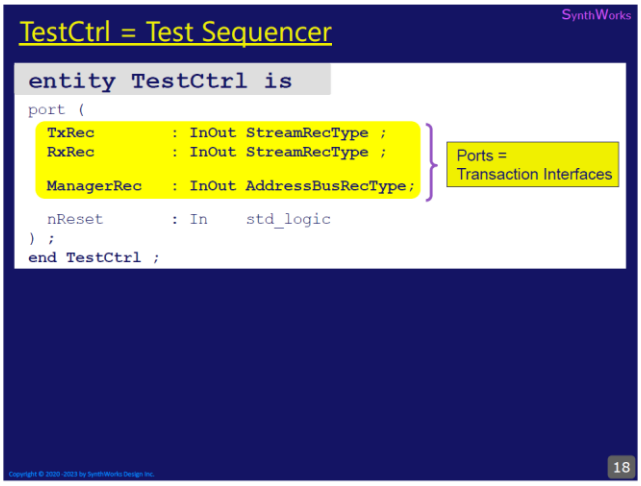
- … Stream MIT (basic subset)

```
type StreamRecType is record  . . . ;
Send (TxRec, iData [, iParam]) ;
Get  (RxRec, oData [, oParam]) ;
. . .
```



Benefits of OSVVM MIT

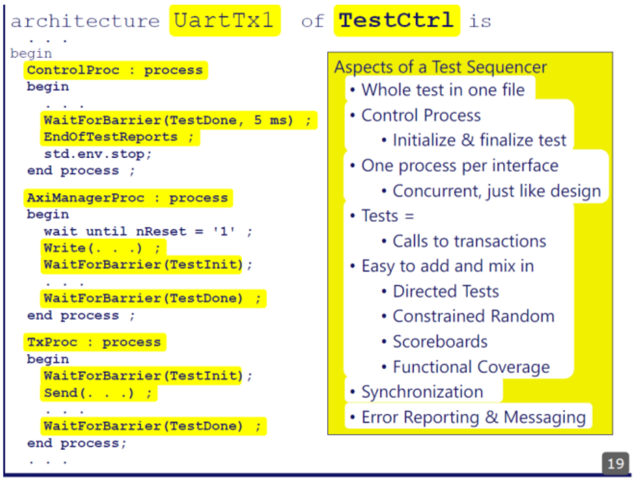
- Accelerates VC Development, Test Writing, and Documentation
  - Verification Component Developers
    - Re-use the transaction interface and API defined by MIT
    - Focus on writing VC behavior
  - Test Writers
    - Similar VC use the same transaction API
    - Similar VC can share sequences of transactions
  - Co-Simulation Interface
    - Supports all MIT based VC - including ones you write
  - Documentation
    - VC only need to identify which transactions they support
- More Information is in user guides in OsvvmLibraries/Documentation/
  - Address_Bus_Model_Independent_Transactions_user_guide.pdf
  - Stream_Model_Independent_Transactions_user_guide.pdf



TestCtrl = Test Sequencer

```
entity TestCtrl is
port (
  TxRec          : InOut StreamRecType ;
  RxRec          : InOut StreamRecType ;

  ManagerRec     : InOut AddressBusRecType;

  nReset         : In    std_logic
) ;
end TestCtrl ;
```

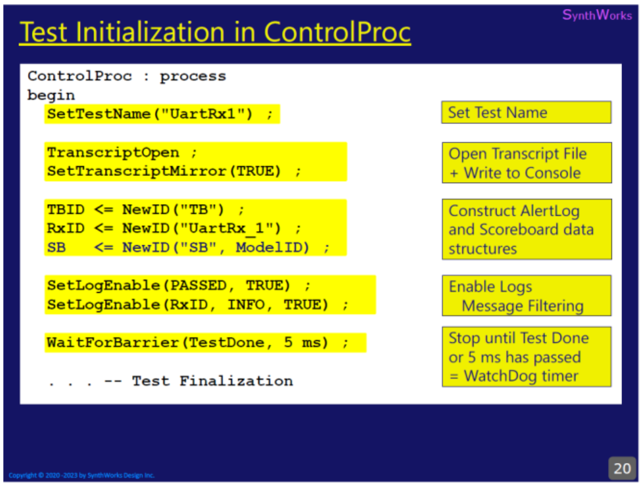
Ports =
Transaction Interfaces



```
architecture UartTx1 of TestCtrl is
  . . .
begin
  ControlProc : process
  begin
    . . .
    WaitForBarrier(TestDone, 5 ms) ;
    EndOfTestReports ;
    std.env.stop;
  end process ;

  AxiManagerProc : process
  begin
    wait until nReset = '1' ;
    Write(. . .) ;
    WaitForBarrier(TestInit);
    . . .
    WaitForBarrier(TestDone) ;
  end process ;

  TxProc : process
  begin
    WaitForBarrier(TestInit);
    Send(. . .) ;
    . . .
    WaitForBarrier(TestDone) ;
  end process;
  . . .
```

Aspects of a Test Sequencer

- Whole test in one file
- Control Process
  - Initialize & finalize test
- One process per interface
  - Concurrent, just like design
- Tests =
  - Calls to transactions
- Easy to add and mix in
  - Directed Tests
  - Constrained Random
  - Scoreboards
  - Functional Coverage
- Synchronization
- Error Reporting & Messaging



Test Initialization in ControlProc

```
ControlProc : process
begin
  SetTestName("UartRx1") ;

  TranscriptOpen ;
  SetTranscriptMirror(TRUE) ;

  TBID <= NewID("TB") ;
  RxID <= NewID("UartRx_1") ;
  SB   <= NewID("SB", ModelID) ;

  SetLogEnable(PASSED, TRUE) ;
  SetLogEnable(RxID, INFO, TRUE) ;

  WaitForBarrier(TestDone, 5 ms) ;

  . . . -- Test Finalization
```

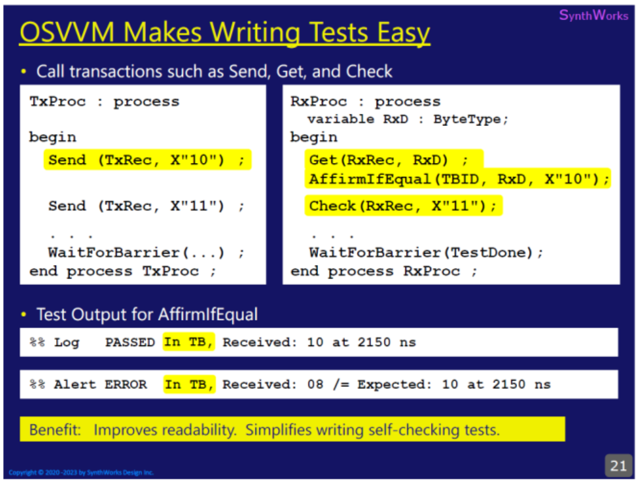
Set Test Name

Open Transcript File
+ Write to Console

Construct AlertLog
and Scoreboard data
structures

Enable Logs
Message Filtering

Stop until Test Done
or 5 ms has passed
= WatchDog timer



OSVVM Makes Writing Tests Easy

- Call transactions such as Send, Get, and Check

```
TxProc : process

begin
  Send (TxRec, X"10") ;

  Send (TxRec, X"11") ;

  . . .
  WaitForBarrier(...) ;
end process TxProc ;
```

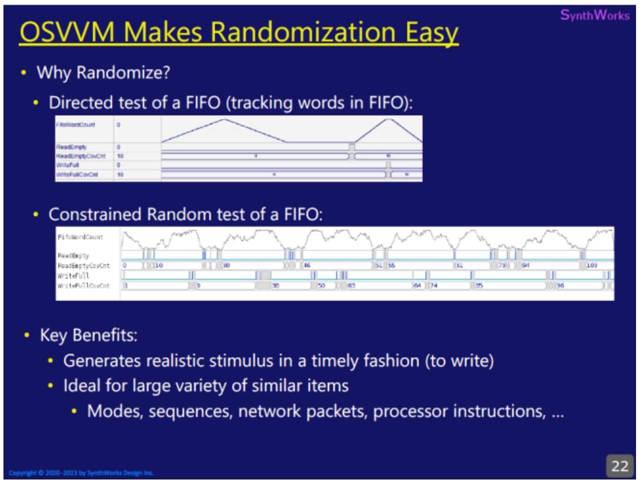
```
RxProc : process
  variable RxD : ByteType;
begin
  Get(RxRec, RxD) ;
  AffirmIfEqual(TBID, RxD, X"10");

  Check(RxRec, X"11");

  . . .
  WaitForBarrier(TestDone);
end process RxProc ;
```

- Test Output for AffirmIfEqual

```
%% Log   PASSED In TB, Received: 10 at 2150 ns
```

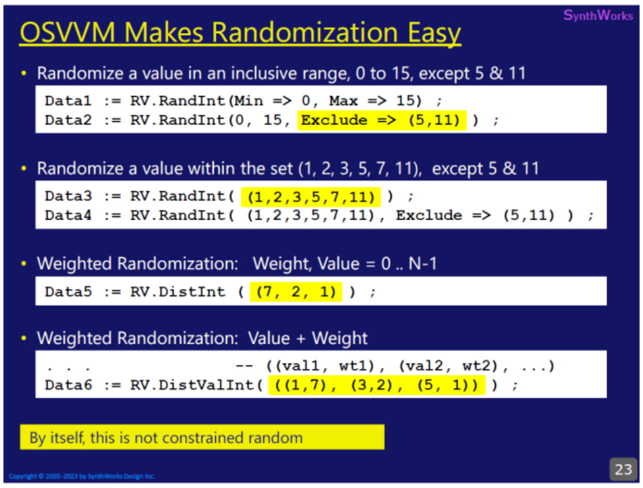
```
%% Alert ERROR  In TB, Received: 08 /= Expected: 10 at 2150 ns
```

Benefit: Improves readability. Simplifies writing self-checking tests.



OSVVM Makes Randomization Easy

- Why Randomize?
  - Directed test of a FIFO (tracking words in FIFO):
  - Constrained Random test of a FIFO:
- Key Benefits:
  - Generates realistic stimulus in a timely fashion (to write)
  - Ideal for large variety of similar items
    - Modes, sequences, network packets, processor instructions, …



OSVVM Makes Randomization Easy

- Randomize a value in an inclusive range, 0 to 15, except 5 & 11

```
Data1 := RV.RandInt(Min => 0, Max => 15) ;
Data2 := RV.RandInt(0, 15, Exclude => (5,11) ) ;
```

- Randomize a value within the set (1, 2, 3, 5, 7, 11), except 5 & 11

```
Data3 := RV.RandInt( (1,2,3,5,7,11) ) ;
Data4 := RV.RandInt( (1,2,3,5,7,11), Exclude => (5,11) ) ;
```

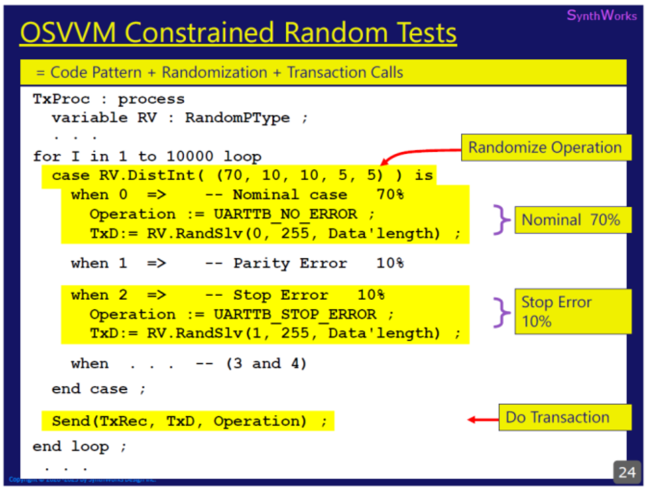
- Weighted Randomization: Weight, Value = 0 .. N-1

```
Data5 := RV.DistInt ( (7, 2, 1) ) ;
```

- Weighted Randomization: Value + Weight

```
. . .                  -- ((val1, wt1), (val2, wt2), ...)
Data6 := RV.DistValInt( ((1,7), (3,2), (5, 1)) ) ;
```

By itself, this is not constrained random



OSVVM Constrained Random Tests

= Code Pattern + Randomization + Transaction Calls

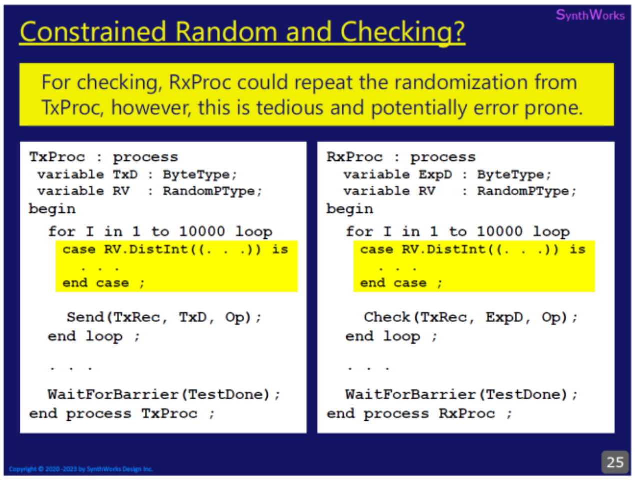
```
TxProc : process
  variable RV : RandomPType ;
  . . .
for I in 1 to 10000 loop
  case RV.DistInt( (70, 10, 10, 5, 5) ) is
    when 0  =>     -- Nominal case   70%
      Operation := UARTTB_NO_ERROR ;
      TxD:= RV.RandSlv(0, 255, Data'length) ;

    when 1  =>     -- Parity Error   10%

    when 2  =>     -- Stop Error   10%
      Operation := UARTTB_STOP_ERROR ;
      TxD:= RV.RandSlv(1, 255, Data'length) ;

    when  . . .   -- (3 and 4)
  end case ;

  Send(TxRec, TxD, Operation) ;
end loop ;
  . . .
```

Randomize Operation

Nominal 70%

Stop Error 10%

Do Transaction

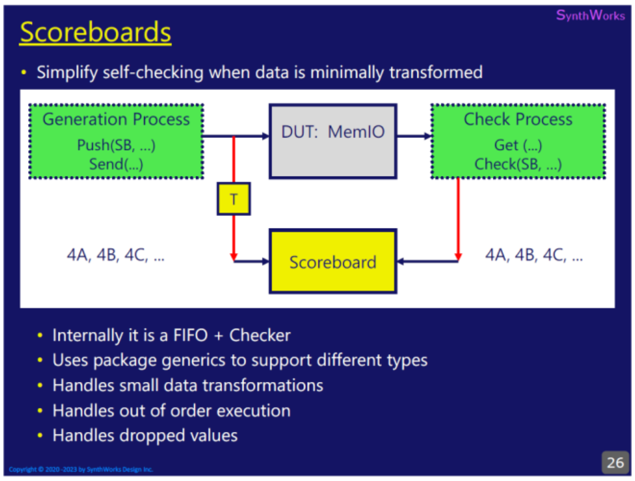


Constrained Random and Checking?

For checking, RxProc could repeat the randomization from TxProc, however, this is tedious and potentially error prone.

```
TxProc : process
  variable TxD : ByteType;
  variable RV  : RandomPType;
begin
  for I in 1 to 10000 loop
    case RV.DistInt((. . .)) is
      . . .
    end case ;

    Send(TxRec, TxD, Op);
  end loop ;

  . . .

  WaitForBarrier(TestDone);
end process TxProc ;
```

```
RxProc : process
  variable ExpD : ByteType;
  variable RV   : RandomPType;
begin
  for I in 1 to 10000 loop
    case RV.DistInt((. . .)) is
      . . .
    end case ;

    Check(TxRec, ExpD, Op);
  end loop ;

  . . .

  WaitForBarrier(TestDone);
end process RxProc ;
```



Scoreboards

- Simplify self-checking when data is minimally transformed

Generation Process
Push(SB, ...)
Send(...)

DUT: MemIO

Check Process
Get (...)
Check(SB, ...)

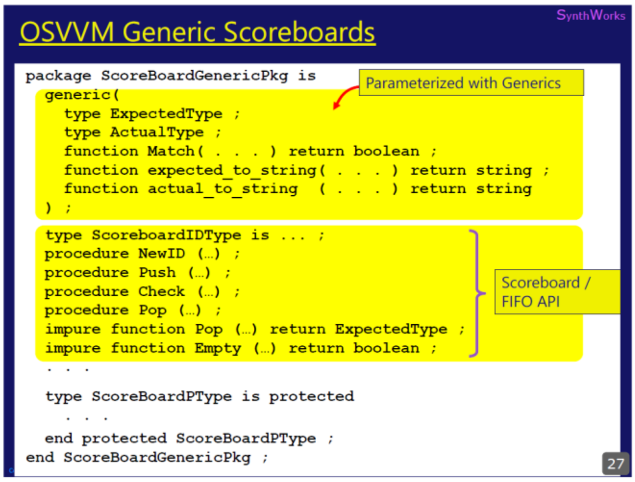
T

Scoreboard

4A, 4B, 4C, ...

4A, 4B, 4C, ...

- Internally it is a FIFO + Checker
- Uses package generics to support different types
- Handles small data transformations
- Handles out of order execution
- Handles dropped values

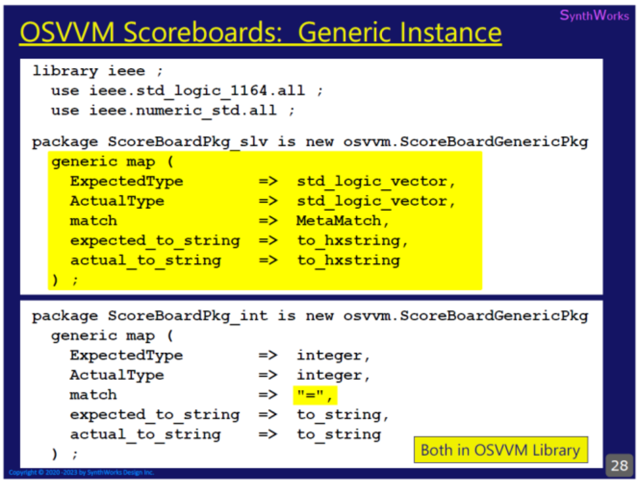


OSVVM Generic Scoreboards

Parameterized with Generics

```
package ScoreBoardGenericPkg is
  generic(
    type ExpectedType ;
    type ActualType ;
    function Match( . . . ) return boolean ;
    function expected_to_string( . . . ) return string ;
    function actual_to_string  ( . . . ) return string
  ) ;
  type ScoreboardIDType is ... ;
  procedure NewID  (…) ;
  procedure Push  (…) ;
  procedure Check  (…) ;
  procedure Pop  (…) ;
  impure function Pop (…) return ExpectedType ;
  impure function Empty (…) return boolean ;
  . . .

  type ScoreBoardPType is protected
    . . .
  end protected ScoreBoardPType ;
end ScoreBoardGenericPkg ;
```

Scoreboard / FIFO API

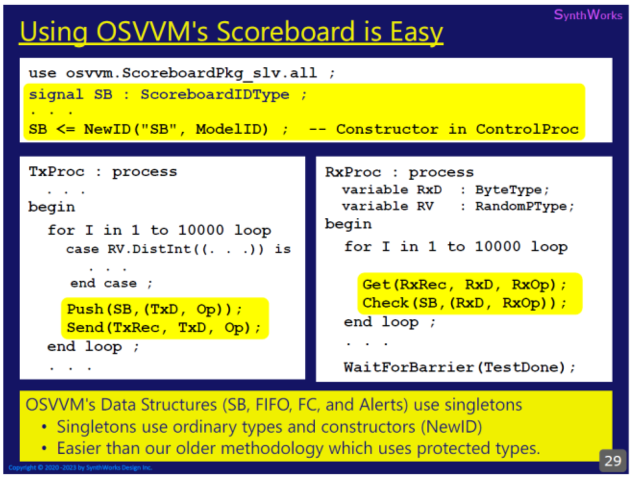


OSVVM Scoreboards: Generic Instance

```
library ieee ;
  use ieee.std_logic_1164.all ;
  use ieee.numeric_std.all ;

package ScoreBoardPkg_slv is new osvvm.ScoreBoardGenericPkg
  generic map (
    ExpectedType        =>  std_logic_vector,
    ActualType          =>  std_logic_vector,
    match               =>  MetaMatch,
    expected_to_string  =>  to_hxstring,
    actual_to_string    =>  to_hxstring
  ) ;
```

```
package ScoreBoardPkg_int is new osvvm.ScoreBoardGenericPkg
  generic map (
    ExpectedType        =>  integer,
    ActualType          =>  integer,
    match               =>  "=",
    expected_to_string  =>  to_string,
    actual_to_string    =>  to_string
  ) ;
```

Both in OSVVM Library



Using OSVVM's Scoreboard is Easy

SynthWorks

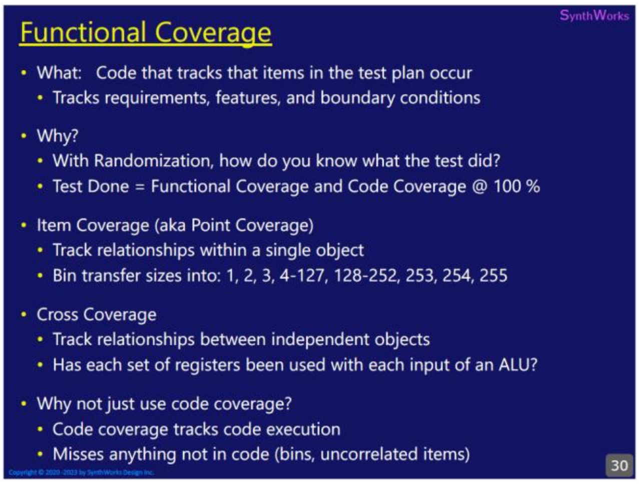
```
use osvvm.ScoreboardPkg_slv.all ;
signal SB : ScoreboardIDType ;
. . .
SB <= NewID("SB", ModelID) ;  -- Constructor in ControlProc
```

```
TxProc : process
  . . .
begin
  for I in 1 to 10000 loop
    case RV.DistInt((. . .)) is
      . . .
     end case ;

     Push(SB,(TxD, Op));
     Send(TxRec, TxD, Op);
  end loop ;
  . . .
```

```
RxProc : process
  variable RxD  : ByteType;
  variable RV   : RandomPType;
begin
  for I in 1 to 10000 loop

    Get(RxRec, RxD, RxOp);
    Check(SB,(RxD, RxOp));
  end loop ;
  . . .

  WaitForBarrier(TestDone);
```

OSVVM's Data Structures (SB, FIFO, FC, and Alerts) use singletons

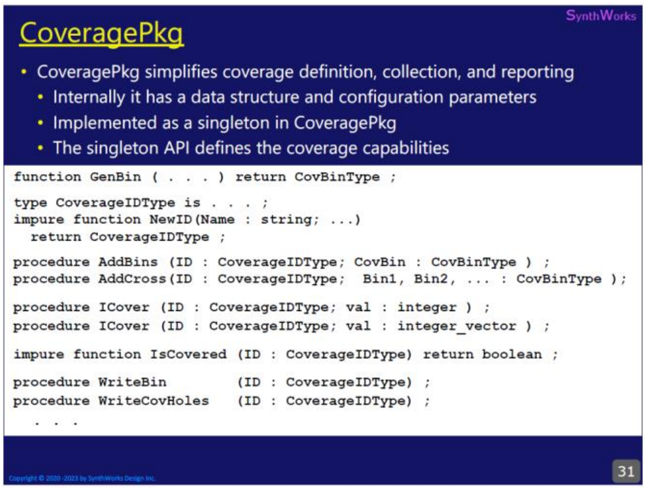
- Singletons use ordinary types and constructors (NewID)
- Easier than our older methodology which uses protected types.



Functional Coverage

SynthWorks

- What: Code that tracks that items in the test plan occur
  - Tracks requirements, features, and boundary conditions
- Why?
  - With Randomization, how do you know what the test did?
  - Test Done = Functional Coverage and Code Coverage @ 100 %
- Item Coverage (aka Point Coverage)
  - Track relationships within a single object
  - Bin transfer sizes into: 1, 2, 3, 4-127, 128-252, 253, 254, 255
- Cross Coverage
  - Track relationships between independent objects
  - Has each set of registers been used with each input of an ALU?
- Why not just use code coverage?
  - Code coverage tracks code execution
  - Misses anything not in code (bins, uncorrelated items)

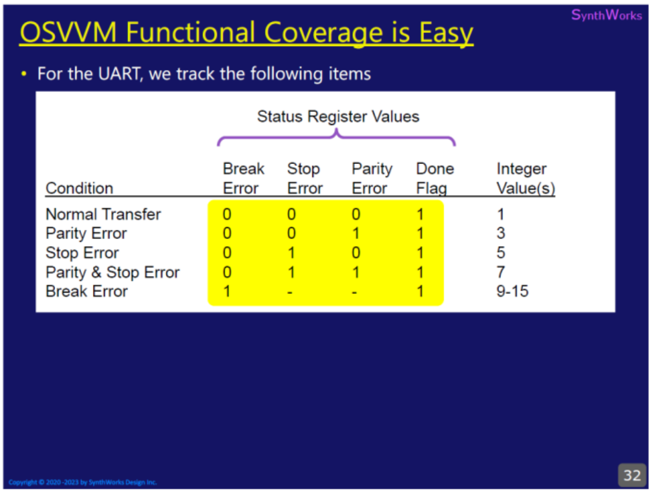


CoveragePkg

SynthWorks

- CoveragePkg simplifies coverage definition, collection, and reporting
  - Internally it has a data structure and configuration parameters
  - Implemented as a singleton in CoveragePkg
  - The singleton API defines the coverage capabilities

```
function GenBin ( . . . ) return CovBinType ;

type CoverageIDType is . . . ;
impure function NewID(Name : string; ...)
  return CoverageIDType ;

procedure AddBins (ID : CoverageIDType; CovBin : CovBinType ) ;
procedure AddCross(ID : CoverageIDType;  Bin1, Bin2, ... : CovBinType );

procedure ICover (ID : CoverageIDType; val : integer ) ;
procedure ICover (ID : CoverageIDType; val : integer_vector ) ;

impure function IsCovered (ID : CoverageIDType) return boolean ;

procedure WriteBin        (ID : CoverageIDType) ;
procedure WriteCovHoles   (ID : CoverageIDType) ;
  . . .
```

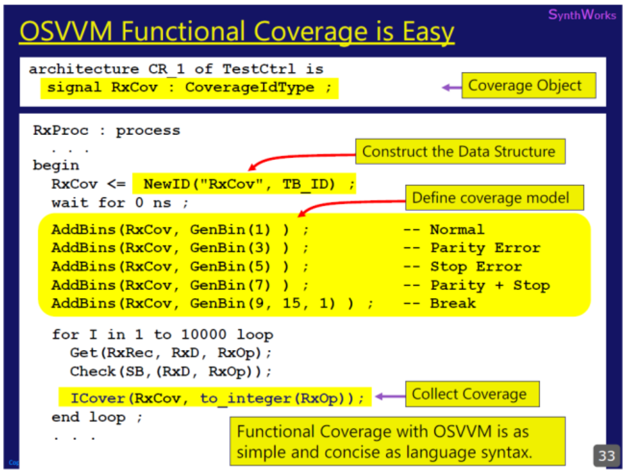


OSVVM Functional Coverage is Easy

SynthWorks

- For the UART, we track the following items

	Status Register Values				
Condition	Break Error	Stop Error	Parity Error	Done Flag	Integer Value(s)
Normal Transfer	0	0	0	1	1
Parity Error	0	0	1	1	3
Stop Error	0	1	0	1	5
Parity & Stop Error	0	1	1	1	7
Break Error	1	-	-	1	9-15



OSVVM Functional Coverage is Easy

SynthWorks

```
architecture CR_1 of TestCtrl is
  signal RxCov : CoverageIdType ;
```

Coverage Object

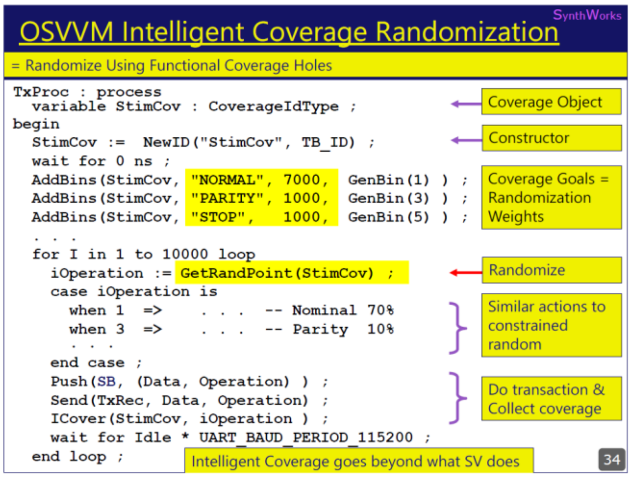
```
RxProc : process
  . . .
begin
  RxCov <=  NewID("RxCov", TB_ID) ;
  wait for 0 ns ;

  AddBins(RxCov, GenBin(1) ) ;               -- Normal
  AddBins(RxCov, GenBin(3) ) ;               -- Parity Error
  AddBins(RxCov, GenBin(5) ) ;               -- Stop Error
  AddBins(RxCov, GenBin(7) ) ;               -- Parity + Stop
  AddBins(RxCov, GenBin(9, 15, 1) ) ;        -- Break

  for I in 1 to 10000 loop
    Get(RxRec, RxD, RxOp);
    Check(SB,(RxD, RxOp));

    ICover(RxCov, to_integer(RxOp));
  end loop ;
  . . .
```

Construct the Data Structure

Define coverage model

Collect Coverage

Functional Coverage with OSVVM is as simple and concise as language syntax.



OSVVM Intelligent Coverage Randomization

= Randomize Using Functional Coverage Holes

SynthWorks

```
TxProc : process
  variable StimCov : CoverageIdType ;
begin
  StimCov :=  NewID("StimCov", TB_ID) ;
  wait for 0 ns ;
  AddBins(StimCov, "NORMAL", 7000,  GenBin(1) ) ;
  AddBins(StimCov, "PARITY", 1000,  GenBin(3) ) ;
  AddBins(StimCov, "STOP",   1000,  GenBin(5) ) ;
  . . .
  for I in 1 to 10000 loop
    iOperation := GetRandPoint(StimCov) ;
    case iOperation is
      when 1  =>    . . .  -- Nominal  70%
      when 3  =>    . . .  -- Parity   10%
      . . .
    end case ;
    Push(SB, (Data, Operation) ) ;
    Send(TxRec, Data, Operation) ;
    ICover(StimCov, iOperation ) ;
    wait for Idle * UART_BAUD_PERIOD_115200 ;
  end loop ;
```

Coverage Object

Constructor

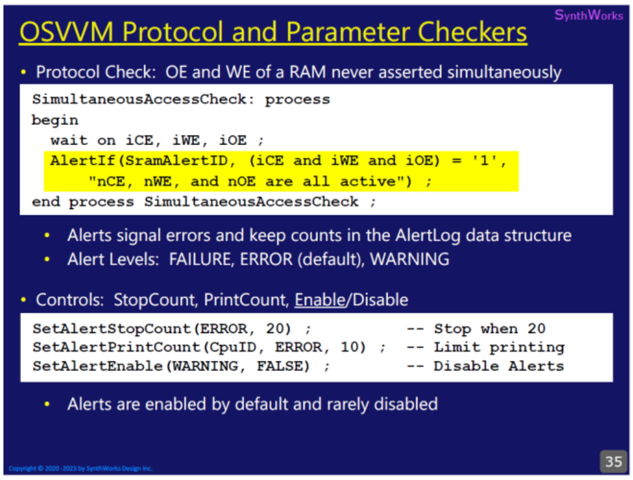
Coverage Goals = Randomization Weights

Randomize

Similar actions to constrained random

Do transaction & Collect coverage

Intelligent Coverage goes beyond what SV does



OSVVM Protocol and Parameter Checkers

SynthWorks

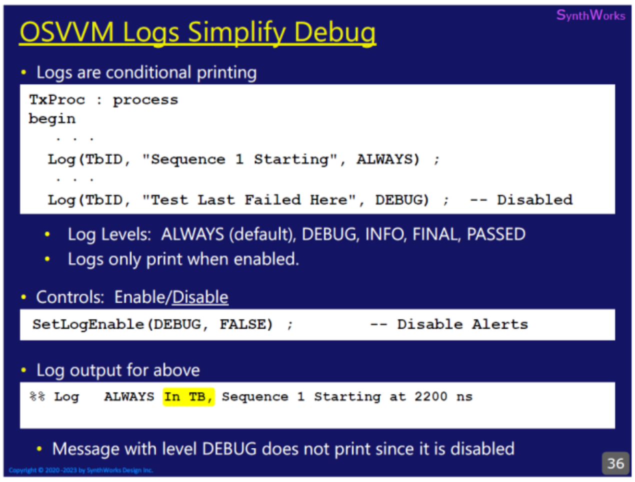
- Protocol Check: OE and WE of a RAM never asserted simultaneously

```
SimultaneousAccessCheck: process
begin
  wait on iCE, iWE, iOE ;
  AlertIf(SramAlertID, (iCE and iWE and iOE) = '1',
       "nCE, nWE, and nOE are all active") ;
end process SimultaneousAccessCheck ;
```

  - Alerts signal errors and keep counts in the AlertLog data structure
  - Alert Levels: FAILURE, ERROR (default), WARNING
- Controls: StopCount, PrintCount, Enable/Disable

```
SetAlertStopCount(ERROR, 20) ;            -- Stop when 20
SetAlertPrintCount(CpuID, ERROR, 10) ;    -- Limit printing
SetAlertEnable(WARNING, FALSE) ;          -- Disable Alerts
```

  - Alerts are enabled by default and rarely disabled



OSVVM Logs Simplify Debug

SynthWorks

- Logs are conditional printing

```
TxProc : process
begin
  . . .
  Log(TbID, "Sequence 1 Starting", ALWAYS) ;
  . . .
  Log(TbID, "Test Last Failed Here", DEBUG) ;   -- Disabled
```

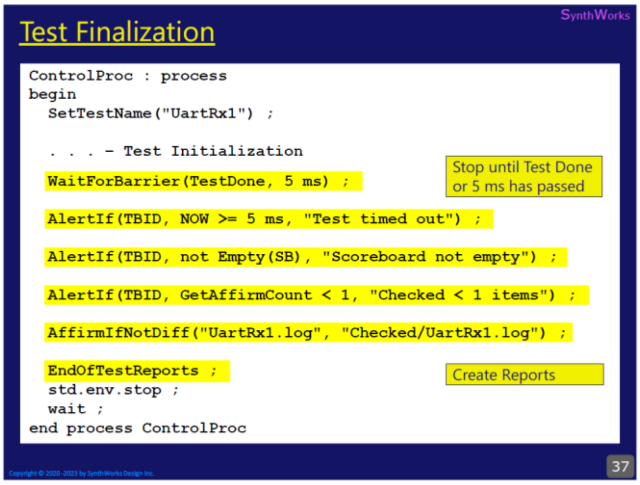
  - Log Levels: ALWAYS (default), DEBUG, INFO, FINAL, PASSED
  - Logs only print when enabled.
- Controls: Enable/Disable

```
SetLogEnable(DEBUG, FALSE) ;          -- Disable Alerts
```

- Log output for above

```
%% Log   ALWAYS In TB, Sequence 1 Starting at 2200 ns
```

  - Message with level DEBUG does not print since it is disabled

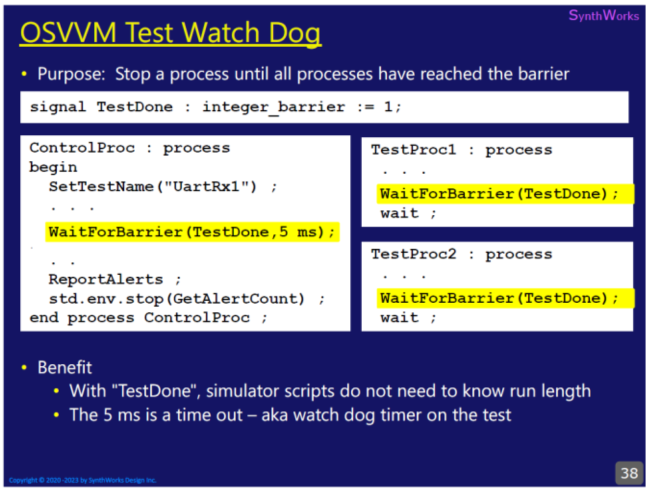


Test Finalization

```
ControlProc : process
begin
  SetTestName("UartRx1") ;

  . . . - Test Initialization
  WaitForBarrier(TestDone, 5 ms) ;

  AlertIf(TBID, NOW >= 5 ms, "Test timed out") ;

  AlertIf(TBID, not Empty(SB), "Scoreboard not empty") ;

  AlertIf(TBID, GetAffirmCount < 1, "Checked < 1 items") ;

  AffirmIfNotDiff("UartRx1.log", "Checked/UartRx1.log") ;

  EndOfTestReports ;
  std.env.stop ;
  wait ;
end process ControlProc
```

Stop until Test Done or 5 ms has passed

Create Reports



OSVVM Test Watch Dog

- Purpose: Stop a process until all processes have reached the barrier

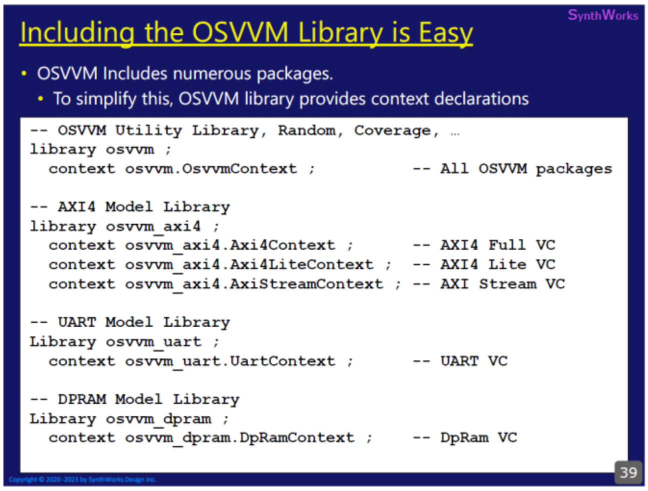
```
signal TestDone : integer_barrier := 1;
```

```
ControlProc : process
begin
  SetTestName("UartRx1") ;
  . . .
  WaitForBarrier(TestDone,5 ms);
  . .
  ReportAlerts ;
  std.env.stop(GetAlertCount) ;
end process ControlProc ;
```

```
TestProc1 : process
  . . .
  WaitForBarrier(TestDone);
  wait ;
```

```
TestProc2 : process
  . . .
  WaitForBarrier(TestDone);
  wait ;
```

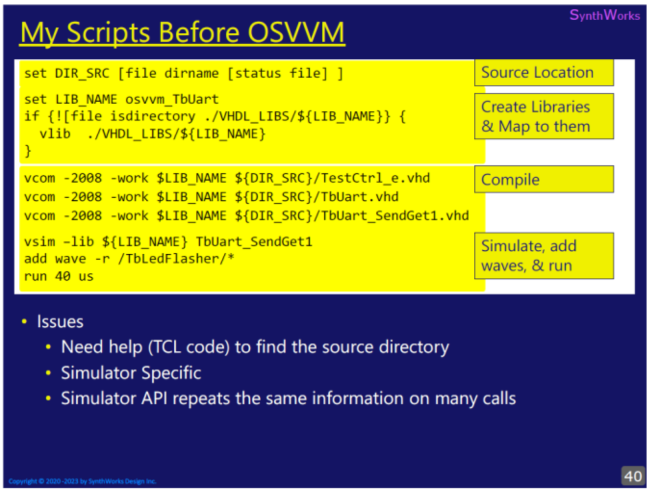
- Benefit
  - With "TestDone", simulator scripts do not need to know run length
  - The 5 ms is a time out – aka watch dog timer on the test



Including the OSVVM Library is Easy

- OSVVM Includes numerous packages.
  - To simplify this, OSVVM library provides context declarations

```
-- OSVVM Utility Library, Random, Coverage, …
library osvvm ;
  context osvvm.OsvvmContext ;           -- All OSVVM packages

-- AXI4 Model Library
library osvvm_axi4 ;
  context osvvm_axi4.Axi4Context ;       -- AXI4 Full VC
  context osvvm_axi4.Axi4LiteContext ;   -- AXI4 Lite VC
  context osvvm_axi4.AxiStreamContext ;  -- AXI Stream VC

-- UART Model Library
Library osvvm_uart ;
  context osvvm_uart.UartContext ;       -- UART VC

-- DPRAM Model Library
Library osvvm_dpram ;
  context osvvm_dpram.DpRamContext ;     -- DpRam VC
```



My Scripts Before OSVVM

```
set DIR_SRC [file dirname [status file] ]

set LIB_NAME osvvm_TbUart
if {![file isdirectory ./VHDL_LIBS/${LIB_NAME}} {
  vlib  ./VHDL_LIBS/${LIB_NAME}
}

vcom -2008 -work $LIB_NAME ${DIR_SRC}/TestCtrl_e.vhd
vcom -2008 -work $LIB_NAME ${DIR_SRC}/TbUart.vhd
vcom -2008 -work $LIB_NAME ${DIR_SRC}/TbUart_SendGet1.vhd

vsim -lib ${LIB_NAME} TbUart_SendGet1
add wave -r /TbLedFlasher/*
run 40 us
```

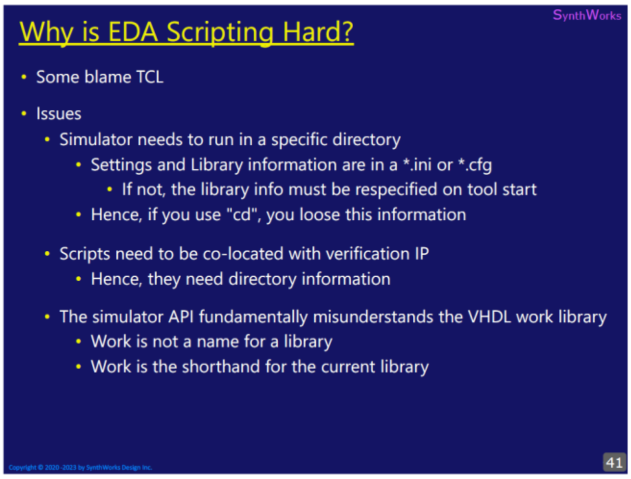
Source Location

Create Libraries & Map to them

Compile

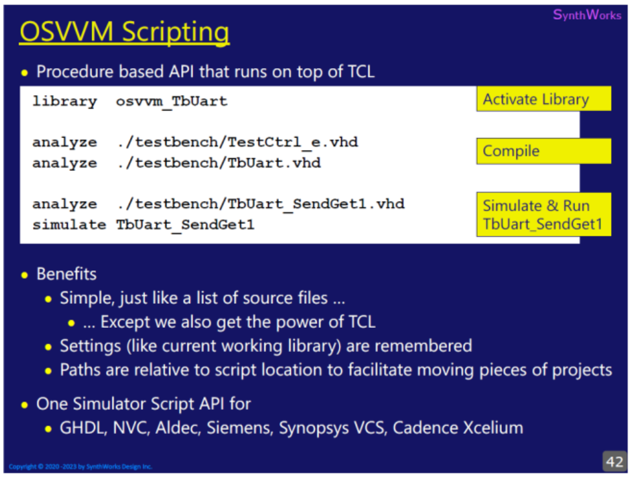
Simulate, add waves, & run

- Issues
  - Need help (TCL code) to find the source directory
  - Simulator Specific
  - Simulator API repeats the same information on many calls



Why is EDA Scripting Hard?

- Some blame TCL
- Issues
  - Simulator needs to run in a specific directory
    - Settings and Library information are in a *.ini or *.cfg
      - If not, the library info must be respecified on tool start
    - Hence, if you use "cd", you loose this information
  - Scripts need to be co-located with verification IP
    - Hence, they need directory information
  - The simulator API fundamentally misunderstands the VHDL work library
    - Work is not a name for a library
    - Work is the shorthand for the current library



OSVVM Scripting

- Procedure based API that runs on top of TCL

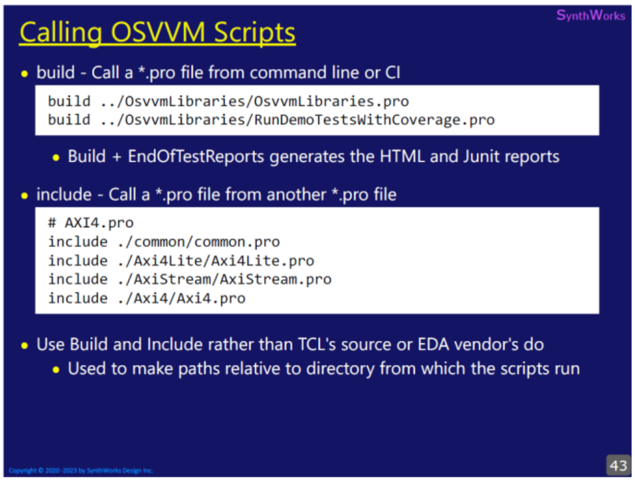
```
library  osvvm_TbUart

analyze  ./testbench/TestCtrl_e.vhd
analyze  ./testbench/TbUart.vhd

analyze  ./testbench/TbUart_SendGet1.vhd
simulate TbUart_SendGet1
```

Activate Library

Compile

Simulate & Run TbUart_SendGet1

- Benefits
  - Simple, just like a list of source files ...
    - ... Except we also get the power of TCL
  - Settings (like current working library) are remembered
  - Paths are relative to script location to facilitate moving pieces of projects
- One Simulator Script API for
  - GHDL, NVC, Aldec, Siemens, Synopsys VCS, Cadence Xcelium



Calling OSVVM Scripts

- build - Call a *.pro file from command line or CI

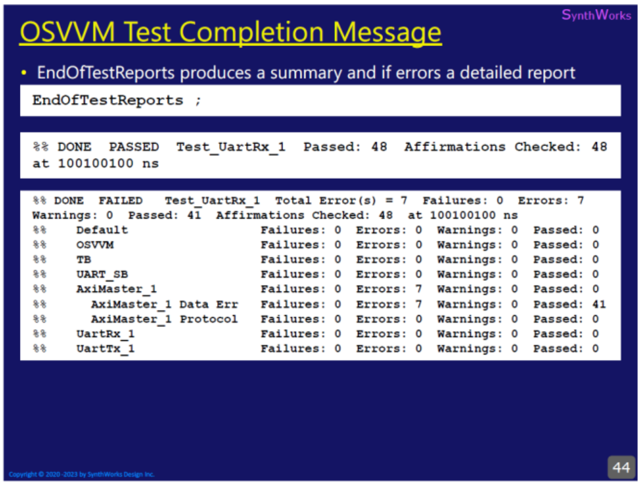
```
build ../OsvvmLibraries/OsvvmLibraries.pro
build ../OsvvmLibraries/RunDemoTestsWithCoverage.pro
```

  - Build + EndOfTestReports generates the HTML and Junit reports

- include - Call a *.pro file from another *.pro file

```
# AXI4.pro
include ./common/common.pro
include ./Axi4Lite/Axi4Lite.pro
include ./AxiStream/AxiStream.pro
include ./Axi4/Axi4.pro
```

- Use Build and Include rather than TCL's source or EDA vendor's do
  - Used to make paths relative to directory from which the scripts run



OSVVM Test Completion Message

- EndOfTestReports produces a summary and if errors a detailed report

```
EndOfTestReports ;
```

```
%% DONE  PASSED  Test_UartRx_1  Passed: 48  Affirmations Checked: 48
at 100100100 ns
```

```
%% DONE  FAILED  Test_UartRx_1  Total Error(s) = 7  Failures: 0  Errors: 7
Warnings: 0  Passed: 41  Affirmations Checked: 48  at 100100100 ns
%%    Default                 Failures: 0  Errors: 0  Warnings: 0  Passed: 0
%%    OSVVM                   Failures: 0  Errors: 0  Warnings: 0  Passed: 0
%%    TB                      Failures: 0  Errors: 0  Warnings: 0  Passed: 0
%%    UART_SB                 Failures: 0  Errors: 0  Warnings: 0  Passed: 0
%%    AxiMaster_1             Failures: 0  Errors: 7  Warnings: 0  Passed: 0
%%      AxiMaster_1 Data Err  Failures: 0  Errors: 7  Warnings: 0  Passed: 41
%%      AxiMaster_1 Protocol  Failures: 0  Errors: 0  Warnings: 0  Passed: 0
%%    UartRx_1                Failures: 0  Errors: 0  Warnings: 0  Passed: 0
%%    UartTx_1                Failures: 0  Errors: 0  Warnings: 0  Passed: 0
```

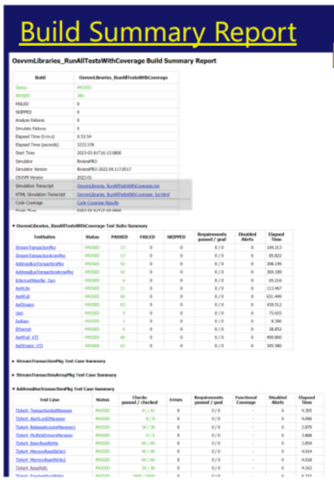
Build Summary Mini-Report

- When a build finishes, a single line, mini report is produced

```
BuildError: Sim_Demo FAILED,  Passed: 149,  Failed: 3,
Skipped: 0,  Analyze Errors: 0,  Simulate Errors: 0,  Build
Error Code: 0
```

  - If there are errors, this is the first place we see an indication

Build Summary Report



OsvvmLibraries_RunAllTestsWithCoverage Build Summary Report

Build	OsvvmLibraries_RunAllTestsWithCoverage
Status	PASSED
PASSED	389
FAILED	0
SKIPPED	0
Analyze Failures	0
Simulate Failures	0
Elapsed Time (h:m:s)	0:53:54
Elapsed Time (seconds)	3233.578
Start Time	2023-03-01T16:15-0800
Simulator	RivieraPRO
Simulator Version	RivieraPRO-2022.04.117.8517
OSVVM Version	2023.01
Simulation Transcript	OsvvmLibraries_RunAllTestsWithCoverage.log
HTML Simulation Transcript	OsvvmLibraries_RunAllTestsWithCoverage_log.html
Code Coverage	Code Coverage Results
Finish Time	[illegible]

▾ OsvvmLibraries_RunAllTestsWithCoverage Test Suite Summary

TestSuites	Status	PASSED	FAILED	SKIPPED	Requirements passed / goal	Disabled Alerts	Elapsed Time
StreamTransactionPkg	PASSED	13	0	0	0 / 0	0	104.213
StreamTransactionArrayPkg	PASSED	13	0	0	0 / 0	0	85.822
AddressBusTransactionPkg	PASSED	42	0	0	0 / 0	0	308.199
AddressBusTransactionArrayPkg	PASSED	42	0	0	0 / 0	0	369.189
InterruptHandler_Gen	PASSED	6	0	0	0 / 0	0	65.216
Axi4Lite	PASSED	11	0	0	0 / 0	0	113.467
Axi4Full	PASSED	60	0	0	0 / 0	0	651.499
AxiStream	PASSED	63	0	0	0 / 0	0	439.512
Uart	PASSED	9	0	0	0 / 0	0	75.655
DpRam	PASSED	1	0	0	0 / 0	0	8.386
Ethernet	PASSED	6	0	0	0 / 0	0	38.852
Axi4Full_VTI	PASSED	60	0	0	0 / 0	0	499.860
AxiStream_VTI	PASSED	63	0	0	0 / 0	0	395.580

▸ StreamTransactionPkg Test Case Summary

▸ StreamTransactionArrayPkg Test Case Summary

▾ AddressBusTransactionPkg Test Case Summary

Test Case	Status	Checks passed / checked	Errors	Requirements passed / goal	Functional Coverage	Disabled Alerts	Elapsed Time
TbAxi4_TransactionApiManager	PASSED	41 / 41	0	0 / 0	-	0	4.305
TbAxi4_AlertLogIDManager	PASSED	8 / 8	0	0 / 0	-	0	4.096
TbAxi4_ReleaseAcquireManager1	PASSED	38 / 38	0	0 / 0	-	0	3.879
TbAxi4_MultipleDriversManager	PASSED	0 / 0	0	0 / 0	-	0	3.868
TbAxi4_BasicReadWrite	PASSED	60 / 60	0	0 / 0	-	0	3.854
TbAxi4_MemoryReadWrite1	PASSED	40 / 40	0	0 / 0	-	0	4.014
TbAxi4_MemoryReadWrite2	PASSED	60 / 60	0	0 / 0	-	0	4.018
TbAxi4_ReadPoll1	PASSED	28 / 28	0	0 / 0	-	0	4.163
TbAxi4_RandomReadWrite	PASSED	3000 / 3000	0	0 / 0	-	0	[illegible]

Created by Scripts + EndOfTestReports

Build Status

Summarizes all tests run
Link to Simulation Transcript
Both text and html
Link to code coverage

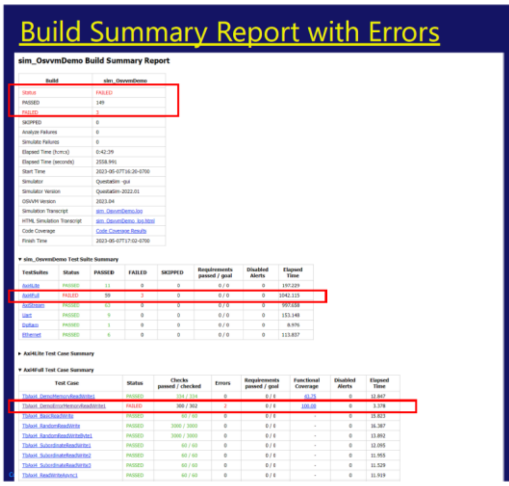
Test Suite Summaries

Test Suite = multiple test cases
Summarizes Pass/Fail+

Test Case Summaries

Test Case = Testbench
Identify Failing Test(s)
Links to Test Case Reports

Build Summary Report with Errors



sim_OsvvmDemo Build Summary Report

Build	sim_OsvvmDemo
Status	FAILED
PASSED	149
FAILED	3
SKIPPED	0
Analyze Failures	0
Simulate Failures	0
Elapsed Time (h:m:s)	0:42:39
Elapsed Time (seconds)	2558.991
Start Time	2023-05-07T16:20-0700
Simulator	QuestaSim -gui
Simulator Version	QuestaSim-2022.01
OSVVM Version	2023.04
Simulation Transcript	sim_OsvvmDemo.log
HTML Simulation Transcript	sim_OsvvmDemo_log.html
Code Coverage	Code Coverage Results
Finish Time	2023-05-07T17:02-0700

▾ sim_OsvvmDemo Test Suite Summary

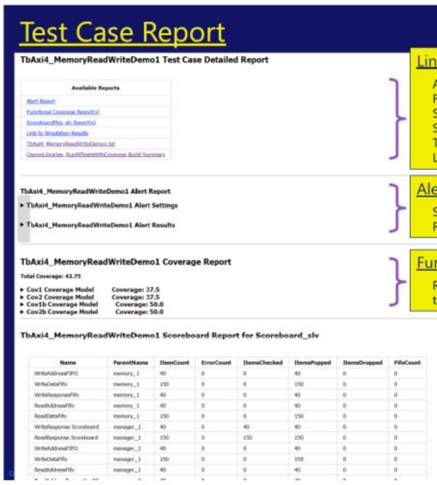
TestSuites	Status	PASSED	FAILED	SKIPPED	Requirements passed / goal	Disabled Alerts	Elapsed Time
Axi4Lite	PASSED	11	0	0	0 / 0	0	197.229
Axi4Full	FAILED	59	3	0	0 / 0	0	1042.115
AxiStream	PASSED	63	0	0	0 / 0	0	997.658
Uart	PASSED	9	0	0	0 / 0	0	153.148
DpRam	PASSED	1	0	0	0 / 0	0	8.976
Ethernet	PASSED	6	0	0	0 / 0	0	113.837

▸ Axi4Lite Test Case Summary

▾ Axi4Full Test Case Summary

Test Case	Status	Checks passed / checked	Errors	Requirements passed / goal	Functional Coverage	Disabled Alerts	Elapsed Time
TbAxi4_DemoMemoryReadWrite1	PASSED	334 / 334	0	0 / 0	43.75	0	12.847
TbAxi4_DemoErrorMemoryReadWrite1	FAILED	300 / 302	2	0 / 0	100.00	0	3.378
TbAxi4_BasicReadWrite	PASSED	60 / 60	0	0 / 0	-	0	15.823
TbAxi4_RandomReadWrite	PASSED	3000 / 3000	0	0 / 0	-	0	16.387
TbAxi4_RandomReadWriteByte1	PASSED	3000 / 3000	0	0 / 0	-	0	13.892
TbAxi4_SubordinateReadWrite1	PASSED	60 / 60	0	0 / 0	-	0	12.095
TbAxi4_SubordinateReadWrite2	PASSED	60 / 60	0	0 / 0	-	0	11.955
TbAxi4_SubordinateReadWrite3	PASSED	60 / 60	0	0 / 0	-	0	11.529
TbAxi4_ReadWriteAsync1	PASSED	60 / 60	0	0 / 0	-	0	11.919

Test Case Report



TbAxi4_MemoryReadWriteDemo1 Test Case Detailed Report

Available Reports
Alert Report
Functional Coverage Report(s)
ScoreboardPkg_slv Report(s)
Link to Simulation Results
TbAxi4_MemoryReadWriteDemo1.txt
OsvvmLibraries_RunAllTestsWithCoverage Build Summary

TbAxi4_MemoryReadWriteDemo1 Alert Report

▸ TbAxi4_MemoryReadWriteDemo1 Alert Settings

▸ TbAxi4_MemoryReadWriteDemo1 Alert Results

TbAxi4_MemoryReadWriteDemo1 Coverage Report

Total Coverage: 43.75

▸ Cov1 Coverage Model Coverage: 37.5
▸ Cov2 Coverage Model Coverage: 37.5
▸ Cov1b Coverage Model Coverage: 50.0
▸ Cov2b Coverage Model Coverage: 50.0

TbAxi4_MemoryReadWriteDemo1 Scoreboard Report for Scoreboard_slv

Name	ParentName	ItemCount	ErrorCount	ItemsChecked	ItemsPopped	ItemsDropped	FifoCount
WriteAddressFIFO	memory_1	40	0	0	40	0	0
WriteDataFifo	memory_1	150	0	0	150	0	0
WriteResponseFifo	memory_1	40	0	0	40	0	0
ReadAddressFifo	memory_1	40	0	0	40	0	0
ReadDataFifo	memory_1	150	0	0	150	0	0
WriteResponse Scoreboard	manager_1	40	0	40	40	0	0
ReadResponse Scoreboard	manager_1	150	0	150	150	0	0
WriteAddressFIFO	manager_1	40	0	0	40	0	0
WriteDataFifo	manager_1	150	0	0	150	0	0
ReadAddressFifo	manager_1	40	0	0	40	0	0

Links

Alert Report
Functional Coverage Report
Scoreboard Reports
Simulation Results
Test Case Transcript
Link to Build Summary

Alert Report

Settings (hidden)
Results (hidden)

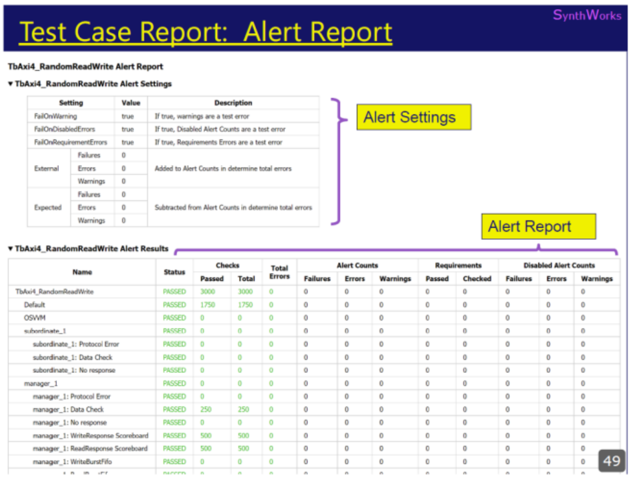
Functional Coverage Report

Report for each FC model in testbench (each hidden)

Scoreboard Report

One Table for each Scoreboard type.
One row in table for each scoreboard.

Test Case Report: Alert Report



TbAxi4_RandomReadWrite Alert Report

▾ TbAxi4_RandomReadWrite Alert Settings

Setting		Value	Description
FailOnWarning		true	If true, warnings are a test error
FailOnDisabledErrors		true	If true, Disabled Alert Counts are a test error
FailOnRequirementErrors		true	If true, Requirements Errors are a test error
External	Failures	0	Added to Alert Counts in determine total errors
	Errors	0	
	Warnings	0	
Expected	Failures	0	Subtracted from Alert Counts in determine total errors
	Errors	0	
	Warnings	0	

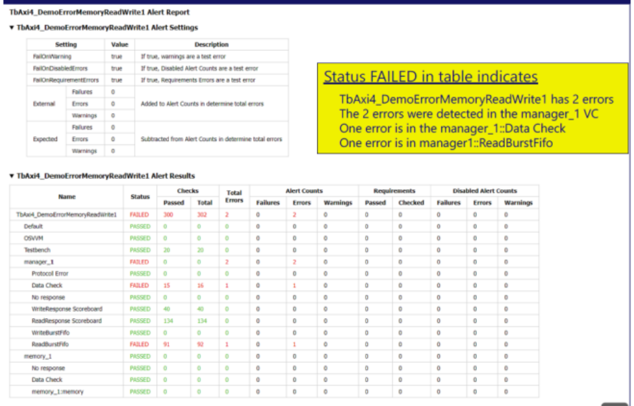
Alert Settings

Alert Report

▾ TbAxi4_RandomReadWrite Alert Results

Name	Status	Checks		Total Errors	Alert Counts			Requirements		Disabled Alert Counts		
		Passed	Total		Failures	Errors	Warnings	Passed	Checked	Failures	Errors	Warnings
TbAxi4_RandomReadWrite	PASSED	3000	3000	0	0	0	0	0	0	0	0	0
Default	PASSED	1750	1750	0	0	0	0	0	0	0	0	0
OSVVM	PASSED	0	0	0	0	0	0	0	0	0	0	0
subordinate_1	PASSED	0	0	0	0	0	0	0	0	0	0	0
subordinate_1: Protocol Error	PASSED	0	0	0	0	0	0	0	0	0	0	0
subordinate_1: Data Check	PASSED	0	0	0	0	0	0	0	0	0	0	0
subordinate_1: No response	PASSED	0	0	0	0	0	0	0	0	0	0	0
manager_1	PASSED	0	0	0	0	0	0	0	0	0	0	0
manager_1: Protocol Error	PASSED	0	0	0	0	0	0	0	0	0	0	0
manager_1: Data Check	PASSED	250	250	0	0	0	0	0	0	0	0	0
manager_1: No response	PASSED	0	0	0	0	0	0	0	0	0	0	0
manager_1: WriteResponse Scoreboard	PASSED	500	500	0	0	0	0	0	0	0	0	0
manager_1: ReadResponse Scoreboard	PASSED	500	500	0	0	0	0	0	0	0	0	0
manager_1: WriteBurstFifo	PASSED	0	0	0	0	0	0	0	0	0	0	0

Test Case Report: Alert Report with Errors



TbAxi4_DemoErrorMemoryReadWrite1 Alert Report

▾ TbAxi4_DemoErrorMemoryReadWrite1 Alert Settings

Setting		Value	Description
FailOnWarning		true	If true, warnings are a test error
FailOnDisabledErrors		true	If true, Disabled Alert Counts are a test error
FailOnRequirementErrors		true	If true, Requirements Errors are a test error
External	Failures	0	Added to Alert Counts in determine total errors
	Errors	0	
	Warnings	0	
Expected	Failures	0	Subtracted from Alert Counts in determine total errors
	Errors	0	
	Warnings	0	

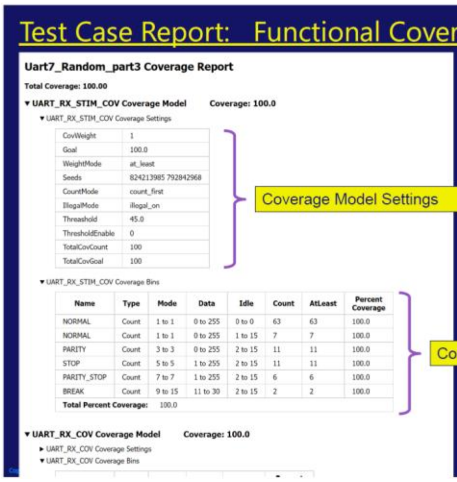
Status FAILED in table indicates

TbAxi4_DemoErrorMemoryReadWrite1 has 2 errors
The 2 errors were detected in the manager_1 VC
One error is in the manager_1::Data Check
One error is in manager1::ReadBurstFifo

▾ TbAxi4_DemoErrorMemoryReadWrite1 Alert Results

Name	Status	Checks		Total Errors	Alert Counts			Requirements		Disabled Alert Counts		
		Passed	Total		Failures	Errors	Warnings	Passed	Checked	Failures	Errors	Warnings
TbAxi4_DemoErrorMemoryReadWrite1	FAILED	300	302	2	0	2	0	0	0	0	0	0
Default	PASSED	0	0	0	0	0	0	0	0	0	0	0
OSVVM	PASSED	0	0	0	0	0	0	0	0	0	0	0
Testbench	PASSED	20	20	0	0	0	0	0	0	0	0	0
manager_1	FAILED	0	0	2	0	2	0	0	0	0	0	0
Protocol Error	PASSED	0	0	0	0	0	0	0	0	0	0	0
Data Check	FAILED	15	16	1	0	1	0	0	0	0	0	0
No response	PASSED	0	0	0	0	0	0	0	0	0	0	0
WriteResponse Scoreboard	PASSED	40	40	0	0	0	0	0	0	0	0	0
ReadResponse Scoreboard	PASSED	134	134	0	0	0	0	0	0	0	0	0
WriteBurstFifo	PASSED	0	0	0	0	0	0	0	0	0	0	0
ReadBurstFifo	FAILED	91	92	1	0	1	0	0	0	0	0	0
memory_1	PASSED	0	0	0	0	0	0	0	0	0	0	0
No response	PASSED	0	0	0	0	0	0	0	0	0	0	0
Data Check	PASSED	0	0	0	0	0	0	0	0	0	0	0
memory_1::memory	PASSED	0	0	0	0	0	0	0	0	0	0	0

Test Case Report: Functional Coverage



Uart7_Random_part3 Coverage Report

Total Coverage: 100.00

▾ UART_RX_STIM_COV Coverage Model   **Coverage: 100.0**

▾ UART_RX_STIM_COV Coverage Settings

CovWeight	1
Goal	100.0
WeightMode	at_least
Seeds	824213985 792842968
CountMode	count_first
IllegalMode	illegal_on
Threashold	45.0
ThresholdEnable	0
TotalCovCount	100
TotalCovGoal	100

Coverage Model Settings

▾ UART_RX_STIM_COV Coverage Bins

Name	Type	Mode	Data	Idle	Count	AtLeast	Percent Coverage
NORMAL	Count	1 to 1	0 to 255	0 to 0	63	63	100.0
NORMAL	Count	1 to 1	0 to 255	1 to 15	7	7	100.0
PARITY	Count	3 to 3	0 to 255	2 to 15	11	11	100.0
STOP	Count	5 to 5	1 to 255	2 to 15	11	11	100.0
PARITY_STOP	Count	7 to 7	1 to 255	2 to 15	6	6	100.0
BREAK	Count	9 to 15	11 to 30	2 to 15	2	2	100.0
Total Percent Coverage:	100.0						

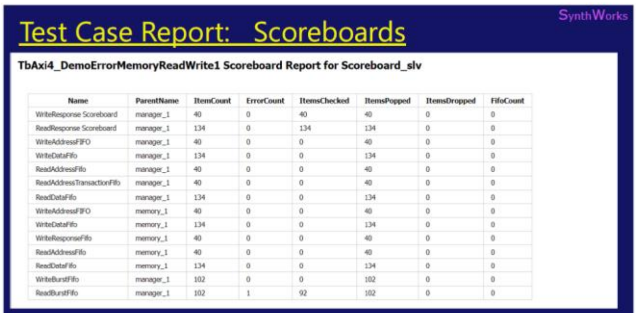
Coverage Results

▾ UART_RX_COV Coverage Model   **Coverage: 100.0**

▸ UART_RX_COV Coverage Settings

▾ UART_RX_COV Coverage Bins

Test Case Report: Scoreboards

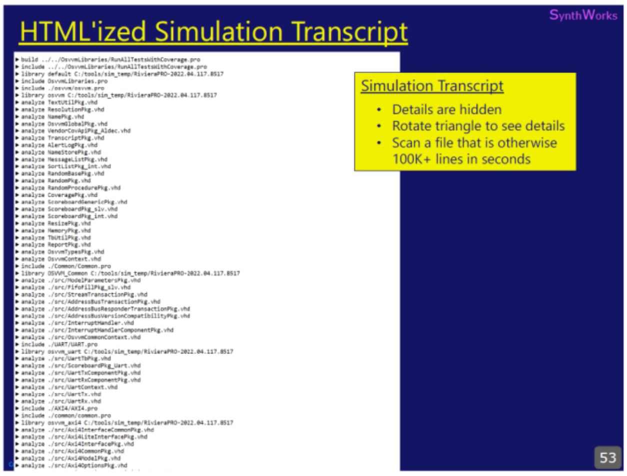


TbAxi4_DemoErrorMemoryReadWrite1 Scoreboard Report for Scoreboard_slv

Name	ParentName	ItemCount	ErrorCount	ItemsChecked	ItemsPopped	ItemsDropped	FifoCount
WriteResponse Scoreboard	manager_1	40	0	40	40	0	0
ReadResponse Scoreboard	manager_1	134	0	134	134	0	0
WriteAddressFIFO	manager_1	40	0	0	40	0	0
WriteDataFifo	manager_1	134	0	0	134	0	0
ReadAddressFifo	manager_1	40	0	0	40	0	0
ReadAddressTransactionFifo	manager_1	40	0	0	40	0	0
ReadDataFifo	manager_1	134	0	0	134	0	0
WriteAddressFIFO	memory_1	40	0	0	40	0	0
WriteDataFifo	memory_1	134	0	0	134	0	0
WriteResponseFifo	memory_1	40	0	0	40	0	0
ReadAddressFifo	memory_1	40	0	0	40	0	0
ReadDataFifo	memory_1	134	0	0	134	0	0
WriteBurstFifo	manager_1	102	0	0	102	0	0
ReadBurstFifo	manager_1	102	1	92	102	0	0

- A separate table is created for each scoreboard instance
- Each row in the table has statistics for a single scoreboard

- Tables for Scoreboard_slv and Scoreboard_int are automatically generated
- Use WriteScoreboardYaml to generate reports for user created scoreboards

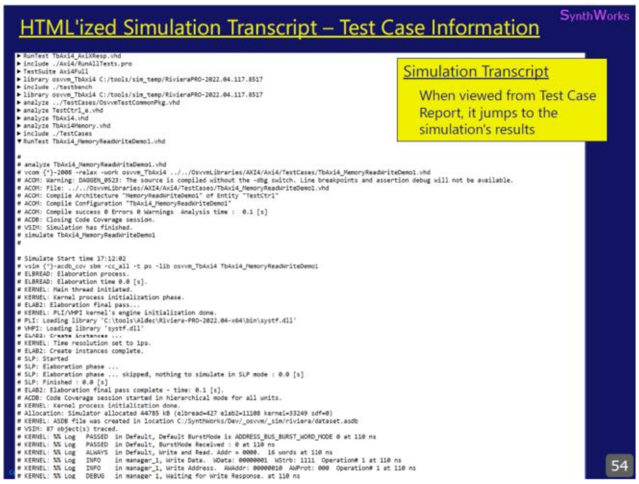


HTML'ized Simulation Transcript

```
▶ build ../../OsvvmLibraries/RunAllTestsWithCoverage.pro
▶ include ../../OsvvmLibraries/RunAllTestsWithCoverage.pro
▶ library default C:/tools/sim_temp/RivieraPRO-2022.04.117.8517
▶ include OsvvmLibraries.pro
▶ include ./osvvm/osvvm.pro
▶ library osvvm C:/tools/sim_temp/RivieraPRO-2022.04.117.8517
▶ analyze TextUtilPkg.vhd
▶ analyze ResolutionPkg.vhd
▶ analyze NamePkg.vhd
▶ analyze OsvvmGlobalPkg.vhd
▶ analyze VendorCovApiPkg_Aldec.vhd
▶ analyze TranscriptPkg.vhd
▶ analyze AlertLogPkg.vhd
▶ analyze NameStorePkg.vhd
▶ analyze MessageListPkg.vhd
▶ analyze SortListPkg_int.vhd
▶ analyze RandomBasePkg.vhd
▶ analyze RandomPkg.vhd
▶ analyze RandomProcedurePkg.vhd
▶ analyze CoveragePkg.vhd
▶ analyze ScoreboardGenericPkg.vhd
▶ analyze ScoreboardPkg_slv.vhd
▶ analyze ScoreboardPkg_int.vhd
▶ analyze ResizePkg.vhd
▶ analyze MemoryPkg.vhd
▶ analyze TbUtilPkg.vhd
▶ analyze ReportPkg.vhd
▶ analyze OsvvmTypesPkg.vhd
▶ analyze OsvvmContext.vhd
▶ include ./Common/Common.pro
▶ library OSVVM_Common C:/tools/sim_temp/RivieraPRO-2022.04.117.8517
▶ analyze ./src/ModelParametersPkg.vhd
▶ analyze ./src/FifoFillPkg_slv.vhd
▶ analyze ./src/StreamTransactionPkg.vhd
▶ analyze ./src/AddressBusTransactionPkg.vhd
▶ analyze ./src/AddressBusResponderTransactionPkg.vhd
▶ analyze ./src/AddressBusVersionCompatibilityPkg.vhd
▶ analyze ./src/InterruptHandler.vhd
▶ analyze ./src/InterruptHandlerComponentPkg.vhd
▶ analyze ./src/OsvvmCommonContext.vhd
▶ include ./UART/UART.pro
▶ library osvvm_uart C:/tools/sim_temp/RivieraPRO-2022.04.117.8517
▶ analyze ./src/UartTbPkg.vhd
▶ analyze ./src/ScoreboardPkg_Uart.vhd
▶ analyze ./src/UartTxComponentPkg.vhd
▶ analyze ./src/UartRxComponentPkg.vhd
▶ analyze ./src/UartContext.vhd
▶ analyze ./src/UartTx.vhd
▶ analyze ./src/UartRx.vhd
▶ include ./AXI4/AXI4.pro
▶ include ./common/common.pro
▶ library osvvm_axi4 C:/tools/sim_temp/RivieraPRO-2022.04.117.8517
▶ analyze ./src/Axi4InterfaceCommonPkg.vhd
▶ analyze ./src/Axi4LiteInterfacePkg.vhd
▶ analyze ./src/Axi4InterfacePkg.vhd
▶ analyze ./src/Axi4CommonPkg.vhd
▶ analyze ./src/Axi4ModelPkg.vhd
▶ analyze ./src/Axi4OptionsPkg.vhd
```

Simulation Transcript

- Details are hidden
- Rotate triangle to see details
- Scan a file that is otherwise 100K+ lines in seconds

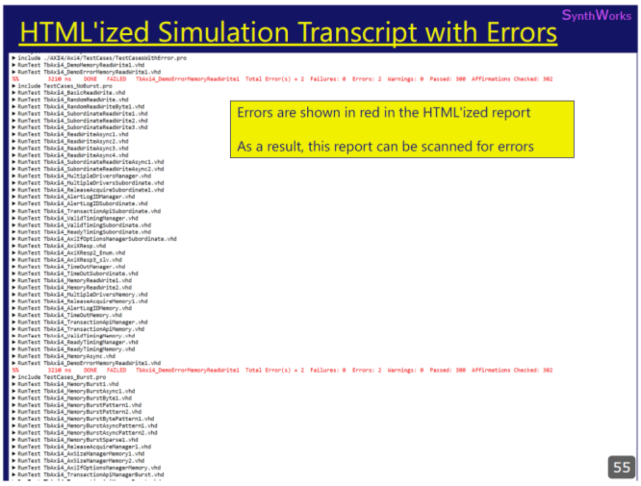


HTML'ized Simulation Transcript – Test Case Information

```
▶ RunTest TbAxi4_AxiXResp.vhd
▶ include ./Axi4/RunAllTests.pro
▶ TestSuite Axi4Full
▶ library osvvm_TbAxi4 C:/tools/sim_temp/RivieraPRO-2022.04.117.8517
▶ include ./testbench
▶ library osvvm_TbAxi4 C:/tools/sim_temp/RivieraPRO-2022.04.117.8517
▶ analyze ../TestCases/OsvvmTestCommonPkg.vhd
▶ analyze TestCtrl_e.vhd
▶ analyze TbAxi4.vhd
▶ analyze TbAxi4Memory.vhd
▶ include ./TestCases
▼ RunTest TbAxi4_MemoryReadWriteDemo1.vhd

#
# analyze TbAxi4_MemoryReadWriteDemo1.vhd
# vcom {*}-2008 -relax -work osvvm_TbAxi4 ../../OsvvmLibraries/AXI4/Axi4/TestCases/TbAxi4_MemoryReadWriteDemo1.vhd
# ACOM: Warning: DAGGEN_0523: The source is compiled without the -dbg switch. Line breakpoints and assertion debug will not be available.
# ACOM: File: ../../OsvvmLibraries/AXI4/Axi4/TestCases/TbAxi4_MemoryReadWriteDemo1.vhd
# ACOM: Compile Architecture "MemoryReadWriteDemo1" of Entity "TestCtrl"
# ACOM: Compile Configuration "TbAxi4_MemoryReadWriteDemo1"
# ACOM: Compile success 0 Errors 0 Warnings  Analysis time :  0.1 [s]
# ACDB: Closing Code Coverage session.
# VSIM: Simulation has finished.
# simulate TbAxi4_MemoryReadWriteDemo1
#

# Simulate Start time 17:12:02
# vsim {*}-acdb_cov sbm -cc_all -t ps -lib osvvm_TbAxi4 TbAxi4_MemoryReadWriteDemo1
# ELBREAD: Elaboration process.
# ELBREAD: Elaboration time 0.0 [s].
# KERNEL: Main thread initiated.
# KERNEL: Kernel process initialization phase.
# ELAB2: Elaboration final pass...
# KERNEL: PLI/VHPI kernel's engine initialization done.
# PLI: Loading library 'C:\tools\Aldec\Riviera-PRO-2022.04-x64\bin\systf.dll'
# VHPI: Loading library 'systf.dll'
# ELAB2: Create instances ...
# KERNEL: Time resolution set to 1ps.
# ELAB2: Create instances complete.
# SLP: Started
# SLP: Elaboration phase ...
# SLP: Elaboration phase ... skipped, nothing to simulate in SLP mode : 0.0 [s]
# SLP: Finished : 0.0 [s]
# ELAB2: Elaboration final pass complete - time: 0.1 [s].
# ACDB: Code Coverage session started in hierarchical mode for all units.
# KERNEL: Kernel process initialization done.
# Allocation: Simulator allocated 44785 kB (elbread=427 elab2=11108 kernel=33249 sdf=0)
# KERNEL: ASDB file was created in location C:/SynthWorks/Dev/_osvvm/_sim/riviera/dataset.asdb
# VSIM: 87 object(s) traced.
# KERNEL: %% Log   PASSED  in Default, Default BurstMode is ADDRESS_BUS_BURST_WORD_MODE 0 at 110 ns
# KERNEL: %% Log   PASSED  in Default, BurstMode Received : 0 at 110 ns
# KERNEL: %% Log   ALWAYS  in Default, Write and Read. Addr = 0000.  16 words at 110 ns
# KERNEL: %% Log   INFO    in manager_1, Write Data.  WData: 00000001  WStrb: 1111  Operation# 1 at 110 ns
# KERNEL: %% Log   INFO    in manager_1, Write Address.  AWAddr: 00000010  AWProt: 000  Operation# 1 at 110 ns
# KERNEL: %% Log   DEBUG   in manager_1, Waiting for Write Response. at 110 ns
```

Simulation Transcript

When viewed from Test Case Report, it jumps to the simulation's results

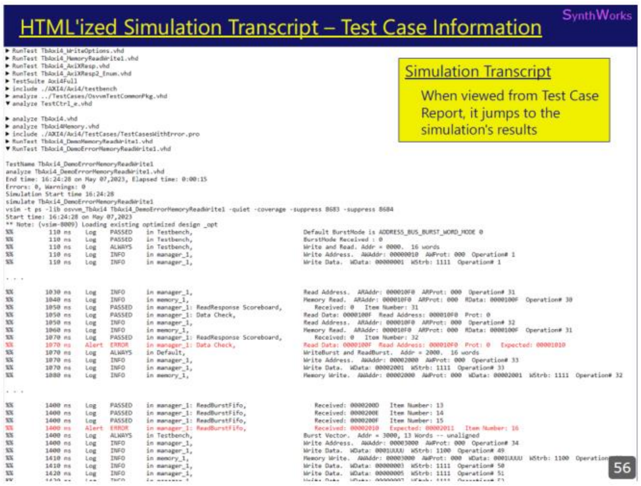


HTML'ized Simulation Transcript with Errors

```
▶ include ./AXI4/Axi4/TestCases/TestCasesWithError.pro
▶ RunTest TbAxi4_DemoMemoryReadWrite1.vhd
▶ RunTest TbAxi4_DemoErrorMemoryReadWrite1.vhd
%%        3210 ns    DONE   FAILED   TbAxi4_DemoErrorMemoryReadWrite1  Total Error(s) = 2  Failures: 0  Errors: 2  Warnings: 0  Passed: 300  Affirmations Checked: 302
▶ include TestCases_NoBurst.pro
▶ RunTest TbAxi4_BasicReadWrite.vhd
▶ RunTest TbAxi4_RandomReadWrite.vhd
▶ RunTest TbAxi4_RandomReadWriteByte1.vhd
▶ RunTest TbAxi4_SubordinateReadWrite1.vhd
▶ RunTest TbAxi4_SubordinateReadWrite2.vhd
▶ RunTest TbAxi4_SubordinateReadWrite3.vhd
▶ RunTest TbAxi4_ReadWriteAsync1.vhd
▶ RunTest TbAxi4_ReadWriteAsync2.vhd
▶ RunTest TbAxi4_ReadWriteAsync3.vhd
▶ RunTest TbAxi4_ReadWriteAsync4.vhd
▶ RunTest TbAxi4_SubordinateReadWriteAsync1.vhd
▶ RunTest TbAxi4_SubordinateReadWriteAsync2.vhd
▶ RunTest TbAxi4_MultipleDriversManager.vhd
▶ RunTest TbAxi4_MultipleDriversSubordinate.vhd
▶ RunTest TbAxi4_ReleaseAcquireSubordinate1.vhd
▶ RunTest TbAxi4_AlertLogIDManager.vhd
▶ RunTest TbAxi4_AlertLogIDSubordinate.vhd
▶ RunTest TbAxi4_TransactionApiSubordinate.vhd
▶ RunTest TbAxi4_ValidTimingManager.vhd
▶ RunTest TbAxi4_ValidTimingSubordinate.vhd
▶ RunTest TbAxi4_ReadyTimingSubordinate.vhd
▶ RunTest TbAxi4_AxiIfOptionsManagerSubordinate.vhd
▶ RunTest TbAxi4_AxiXResp.vhd
▶ RunTest TbAxi4_AxiXResp2_Enum.vhd
▶ RunTest TbAxi4_AxiXResp3_slv.vhd
▶ RunTest TbAxi4_TimeOutManager.vhd
▶ RunTest TbAxi4_TimeOutSubordinate.vhd
▶ RunTest TbAxi4_MemoryReadWrite1.vhd
▶ RunTest TbAxi4_MemoryReadWrite2.vhd
▶ RunTest TbAxi4_MultipleDriversMemory.vhd
▶ RunTest TbAxi4_ReleaseAcquireMemory1.vhd
▶ RunTest TbAxi4_AlertLogIDMemory.vhd
▶ RunTest TbAxi4_TimeOutMemory.vhd
▶ RunTest TbAxi4_TransactionApiManager.vhd
▶ RunTest TbAxi4_TransactionApiMemory.vhd
▶ RunTest TbAxi4_ValidTimingMemory.vhd
▶ RunTest TbAxi4_ReadyTimingManager.vhd
▶ RunTest TbAxi4_ReadyTimingMemory.vhd
▶ RunTest TbAxi4_MemoryAsync.vhd
▶ RunTest TbAxi4_DemoErrorMemoryReadWrite1.vhd
%%        3210 ns    DONE   FAILED   TbAxi4_DemoErrorMemoryReadWrite1  Total Error(s) = 2  Failures: 0  Errors: 2  Warnings: 0  Passed: 300  Affirmations Checked: 302
▶ include TestCases_Burst.pro
▶ RunTest TbAxi4_MemoryBurst1.vhd
▶ RunTest TbAxi4_MemoryBurstAsync1.vhd
▶ RunTest TbAxi4_MemoryBurstByte1.vhd
▶ RunTest TbAxi4_MemoryBurstPattern1.vhd
▶ RunTest TbAxi4_MemoryBurstPattern2.vhd
▶ RunTest TbAxi4_MemoryBurstBytePattern1.vhd
▶ RunTest TbAxi4_MemoryBurstAsyncPattern1.vhd
▶ RunTest TbAxi4_MemoryBurstAsyncPattern2.vhd
▶ RunTest TbAxi4_MemoryBurstSparse1.vhd
▶ RunTest TbAxi4_ReleaseAcquireManager1.vhd
▶ RunTest TbAxi4_AxSizeManagerMemory1.vhd
▶ RunTest TbAxi4_AxSizeManagerMemory2.vhd
▶ RunTest TbAxi4_AxiIfOptionsManagerMemory.vhd
▶ RunTest TbAxi4_TransactionApiManagerBurst.vhd
```

Errors are shown in red in the HTML'ized report

As a result, this report can be scanned for errors



HTML'ized Simulation Transcript – Test Case Information

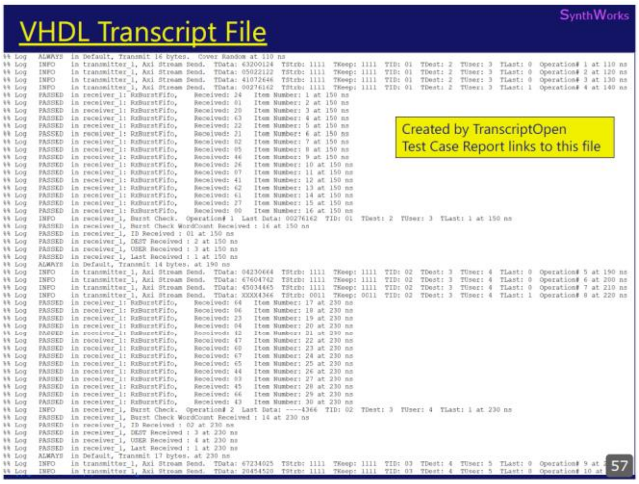
```
▶ RunTest TbAxi4_WriteOptions.vhd
▶ RunTest TbAxi4_MemoryReadWrite1.vhd
▶ RunTest TbAxi4_AxiXResp.vhd
▶ RunTest TbAxi4_AxiXResp2_Enum.vhd
▶ TestSuite Axi4Full
▶ include ./AXI4/Axi4/testbench
▶ analyze ../TestCases/OsvvmTestCommonPkg.vhd
▼ analyze TestCtrl_e.vhd

▶ analyze TbAxi4.vhd
▶ analyze TbAxi4Memory.vhd
▶ include ./AXI4/Axi4/TestCases/TestCasesWithError.pro
▶ RunTest TbAxi4_DemoMemoryReadWrite1.vhd
▼ RunTest TbAxi4_DemoErrorMemoryReadWrite1.vhd

TestName TbAxi4_DemoErrorMemoryReadWrite1
analyze TbAxi4_DemoErrorMemoryReadWrite1.vhd
End time: 16:24:28 on May 07,2023, Elapsed time: 0:00:15
Errors: 0, Warnings: 0
Simulation Start time 16:24:28
simulate TbAxi4_DemoErrorMemoryReadWrite1
vsim -t ps -lib osvvm_TbAxi4 TbAxi4_DemoErrorMemoryReadWrite1 -quiet -coverage -suppress 8683 -suppress 8684
Start time: 16:24:28 on May 07,2023
** Note: (vsim-8009) Loading existing optimized design _opt
%%       110 ns   Log   PASSED   in Testbench,                                Default BurstMode is ADDRESS_BUS_BURST_WORD_MODE 0
%%       110 ns   Log   PASSED   in Testbench,                                BurstMode Received : 0
%%       110 ns   Log   ALWAYS   in Testbench,                                Write and Read. Addr = 0000.  16 words
%%       110 ns   Log   INFO     in manager_1,                                Write Address.  AWAddr: 00000010  AWProt: 000  Operation# 1
%%       110 ns   Log   INFO     in manager_1,                                Write Data.  WData: 00000001  WStrb: 1111  Operation# 1

. . .

%%      1030 ns   Log   INFO     in manager_1,                                Read Address.  ARAddr: 000010F0  ARProt: 000  Operation# 31
%%      1040 ns   Log   INFO     in memory_1,                                 Memory Read.  ARAddr: 000010F0  ARProt: 000  RData: 0000100F  Operation# 30
%%      1050 ns   Log   PASSED   in manager_1: ReadResponse Scoreboard,         Received: 0   Item Number: 31
%%      1050 ns   Log   PASSED   in manager_1: Data Check,                    Read Data: 0000100F  Read Address: 000010F0  Prot: 0
%%      1050 ns   Log   INFO     in manager_1,                                Read Address.  ARAddr: 000010F0  ARProt: 000  Operation# 32
%%      1060 ns   Log   INFO     in memory_1,                                 Memory Read.  ARAddr: 000010F0  ARProt: 000  RData: 0000100F  Operation# 31
%%      1070 ns   Log   PASSED   in manager_1: ReadResponse Scoreboard,         Received: 0   Item Number: 32
%%      1070 ns   Alert ERROR    in manager_1: Data Check,                    Read Data: 0000100F  Read Address: 000010F0  Prot: 0   Expected: 00001010
%%      1070 ns   Log   ALWAYS   in Default,                                  WriteBurst and ReadBurst.  Addr = 2000.  16 words
%%      1070 ns   Log   INFO     in manager_1,                                Write Address.  AWAddr: 00002000  AWProt: 000  Operation# 33
%%      1070 ns   Log   INFO     in manager_1,                                Write Data.  WData: 00002001  WStrb: 1111  Operation# 33
%%      1080 ns   Log   INFO     in memory_1,                                 Memory Write.  AWAddr: 00002000  AWProt: 000  WData: 00002001  WStrb: 1111  Operation# 32

. . .

%%      1400 ns   Log   PASSED   in manager_1: ReadBurstFifo,                   Received: 0000200D   Item Number: 13
%%      1400 ns   Log   PASSED   in manager_1: ReadBurstFifo,                   Received: 0000200E   Item Number: 14
%%      1400 ns   Log   PASSED   in manager_1: ReadBurstFifo,                   Received: 0000200F   Item Number: 15
%%      1400 ns   Alert ERROR    in manager_1: ReadBurstFifo,                   Received: 00002010   Expected: 00002011   Item Number: 16
%%      1400 ns   Log   ALWAYS   in Testbench,                                Burst Vector.  Addr = 3000, 13 Words -- unaligned
%%      1400 ns   Log   INFO     in manager_1,                                Write Address.  AWAddr: 00003000  AWProt: 000  Operation# 34
%%      1400 ns   Log   INFO     in manager_1,                                Write Data.  WData: 0001UUUU  WStrb: 1100  Operation# 49
%%      1410 ns   Log   INFO     in memory_1,                                 Memory Write.  AWAddr: 00003000  AWProt: 000  WData: 0001UUUU  WStrb: 1100  Operation
%%      1410 ns   Log   INFO     in manager_1,                                Write Data.  WData: 00000003  WStrb: 1111  Operation# 50
%%      1420 ns   Log   INFO     in manager_1,                                Write Data.  WData: 00000005  WStrb: 1111  Operation# 51
```

Simulation Transcript

When viewed from Test Case Report, it jumps to the simulation's results

VHDL Transcript File

SynthWorks



```
%% Log   ALWAYS  in Default, Transmit 16 bytes.  Cover Random at 110 ns
%% Log   INFO    in transmitter_1, Axi Stream Send.  TData: 63200124  TStrb: 1111  TKeep: 1111  TID: 01  TDest: 2  TUser: 3  TLast: 0  Operation# 1 at 110 ns
%% Log   INFO    in transmitter_1, Axi Stream Send.  TData: 05022122  TStrb: 1111  TKeep: 1111  TID: 01  TDest: 2  TUser: 3  TLast: 0  Operation# 2 at 120 ns
%% Log   INFO    in transmitter_1, Axi Stream Send.  TData: 41072646  TStrb: 1111  TKeep: 1111  TID: 01  TDest: 2  TUser: 3  TLast: 0  Operation# 3 at 130 ns
%% Log   INFO    in transmitter_1, Axi Stream Send.  TData: 00276162  TStrb: 1111  TKeep: 1111  TID: 01  TDest: 2  TUser: 3  TLast: 1  Operation# 4 at 140 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 24   Item Number: 1 at 150 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 01   Item Number: 2 at 150 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 20   Item Number: 3 at 150 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 63   Item Number: 4 at 150 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 22   Item Number: 5 at 150 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 21   Item Number: 6 at 150 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 02   Item Number: 7 at 150 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 05   Item Number: 8 at 150 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 46   Item Number: 9 at 150 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 26   Item Number: 10 at 150 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 07   Item Number: 11 at 150 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 41   Item Number: 12 at 150 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 62   Item Number: 13 at 150 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 61   Item Number: 14 at 150 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 27   Item Number: 15 at 150 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 00   Item Number: 16 at 150 ns
%% Log   INFO    in receiver_1, Burst Check.  Operation# 1  Last Data: 00276162  TID: 01  TDest: 2  TUser: 3  TLast: 1 at 150 ns
%% Log   PASSED  in receiver_1, Burst Check WordCount Received : 16 at 150 ns
%% Log   PASSED  in receiver_1, ID Received : 01 at 150 ns
%% Log   PASSED  in receiver_1, DEST Received : 2 at 150 ns
%% Log   PASSED  in receiver_1, USER Received : 3 at 150 ns
%% Log   PASSED  in receiver_1, Last Received : 1 at 150 ns
%% Log   ALWAYS  in Default, Transmit 14 bytes. at 190 ns
%% Log   INFO    in transmitter_1, Axi Stream Send.  TData: 04230664  TStrb: 1111  TKeep: 1111  TID: 02  TDest: 3  TUser: 4  TLast: 0  Operation# 5 at 190 ns
%% Log   INFO    in transmitter_1, Axi Stream Send.  TData: 67604742  TStrb: 1111  TKeep: 1111  TID: 02  TDest: 3  TUser: 4  TLast: 0  Operation# 6 at 200 ns
%% Log   INFO    in transmitter_1, Axi Stream Send.  TData: 45034465  TStrb: 1111  TKeep: 1111  TID: 02  TDest: 3  TUser: 4  TLast: 0  Operation# 7 at 210 ns
%% Log   INFO    in transmitter_1, Axi Stream Send.  TData: XXXX4366  TStrb: 0011  TKeep: 0011  TID: 02  TDest: 3  TUser: 4  TLast: 1  Operation# 8 at 220 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 64   Item Number: 17 at 230 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 06   Item Number: 18 at 230 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 23   Item Number: 19 at 230 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 04   Item Number: 20 at 230 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 42   Item Number: 21 at 230 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 47   Item Number: 22 at 230 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 60   Item Number: 23 at 230 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 67   Item Number: 24 at 230 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 65   Item Number: 25 at 230 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 44   Item Number: 26 at 230 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 03   Item Number: 27 at 230 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 45   Item Number: 28 at 230 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 66   Item Number: 29 at 230 ns
%% Log   PASSED  in receiver_1: RxBurstFifo,    Received: 43   Item Number: 30 at 230 ns
%% Log   INFO    in receiver_1, Burst Check.  Operation# 2  Last Data: ----4366  TID: 02  TDest: 3  TUser: 4  TLast: 1 at 230 ns
%% Log   PASSED  in receiver_1, Burst Check WordCount Received : 14 at 230 ns
%% Log   PASSED  in receiver_1, ID Received : 02 at 230 ns
%% Log   PASSED  in receiver_1, DEST Received : 3 at 230 ns
%% Log   PASSED  in receiver_1, USER Received : 4 at 230 ns
%% Log   PASSED  in receiver_1, Last Received : 1 at 230 ns
%% Log   ALWAYS  in Default, Transmit 17 bytes. at 230 ns
%% Log   INFO    in transmitter_1, Axi Stream Send.  TData: 67234025  TStrb: 1111  TKeep: 1111  TID: 03  TDest: 4  TUser: 5  TLast: 0  Operation# 9 at
%% Log   INFO    in transmitter_1, Axi Stream Send.  TData: 20454520  TStrb: 1111  TKeep: 1111  TID: 03  TDest: 4  TUser: 5  TLast: 0  Operation# 10 at
```

Created by TranscriptOpen
Test Case Report links to this file

Getting OSVVM & Running Scripts

SynthWorks

- Get the sources:

```
git clone --recursive https://github.com/osvvm/OsvvmLibraries
```

  - Alternately, a zip file is at: osvvm.org/downloads
- Initialize the simulator – see Documentation/Scripts_user_guide.pdf

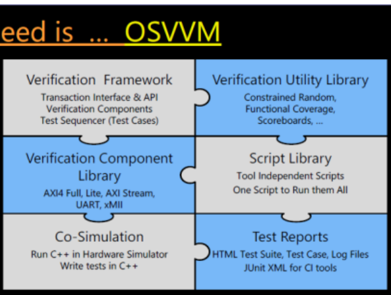
```
file mkdir sim ;    # In directory containing OsvvmLibraries
cd   sim
source ../OsvvmLibraries/Scripts/StartUp.tcl
```

- Build all OSVVM and Run All VC Tests

```
build $OsvvmLibraries/OsvvmLibraries.pro
build $OsvvmLibraries/RunAllTests.pro
```

  - Each VC has a RunAllTests and RunDemoTests

All you need is … OSVVM



Verification Framework
Transaction Interface & API
Verification Components
Test Sequencer (Test Cases)

Verification Utility Library
Constrained Random,
Functional Coverage,
Scoreboards, …

Verification Component Library
AXI4 Full, Lite, AXI Stream,
UART, xMII

Script Library
Tool Independent Scripts
One Script to Run them All

Co-Simulation
Run C++ in Hardware Simulator
Write tests in C++

Test Reports
HTML Test Suite, Test Case, Log Files
JUnit XML for CI tools

- Benefits
  - Powerful and Concise – rivals other verification languages
  - Unmatched reuse through the entire verification process
  - Unmatched report capability with HTML for humans and JUnit XML for CI
  - Tests are Readable and Reviewable by All
  - Adopt incrementally as needed
  - Tests and VC can be written by any VHDL Engineer

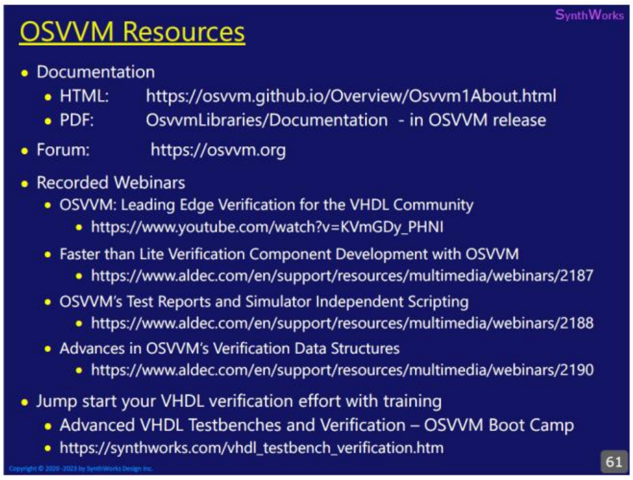
SynthWorks VHDL Classes

SynthWorks

Comprehensive VHDL Introduction   4 Days - beginners class
http://www.synthworks.com/comprehensive_vhdl_introduction.htm
A design and verification engineer's introduction to VHDL syntax, RTL coding, and testbenches. Students get VHDL hardware experience with our FPGA based lab board.

Advanced VHDL Testbenches and Verification - OSVVM Boot Camp - 5 days
http://www.synthworks.com/vhdl_testbench_verification.htm
Learn the latest VHDL verification techniques including transaction based modeling, self-checking, scoreboards, memory modeling, functional coverage, directed, algorithmic, constrained random, and intelligent testbench test generation.   Create a VHDL testbench environment that is competitive with other verification languages, such as SystemVerilog or 'e'.   Our techniques work on VHDL simulators without additional licenses and are accessible to RTL engineers.

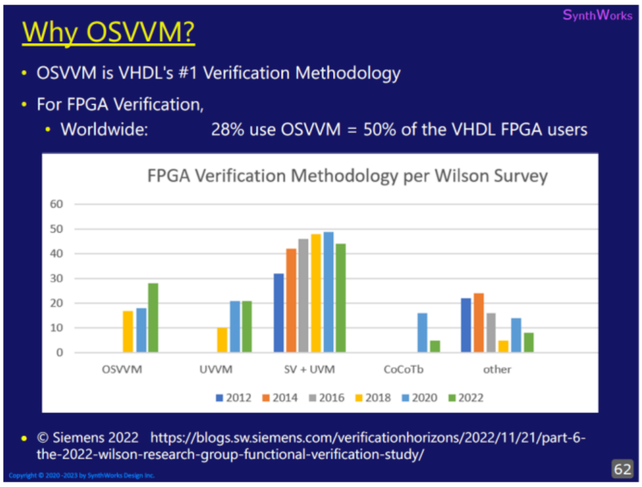
VHDL Coding for Synthesis 4 Days
http://www.synthworks.com/vhdl_rtl_synthesis.htm
Learn VHDL RTL (FPGA and ASIC) coding styles, methodologies, design techniques, problem solving techniques, and advanced language constructs to produce better, faster, and smaller logic.



SynthWorks

OSVVM Resources

- Documentation
  - HTML: https://osvvm.github.io/Overview/Osvvm1About.html
  - PDF: OsvvmLibraries/Documentation - in OSVVM release
- Forum: https://osvvm.org
- Recorded Webinars
  - OSVVM: Leading Edge Verification for the VHDL Community
    - https://www.youtube.com/watch?v=KVmGDy_PHNI
  - Faster than Lite Verification Component Development with OSVVM
    - https://www.aldec.com/en/support/resources/multimedia/webinars/2187
  - OSVVM's Test Reports and Simulator Independent Scripting
    - https://www.aldec.com/en/support/resources/multimedia/webinars/2188
  - Advances in OSVVM's Verification Data Structures
    - https://www.aldec.com/en/support/resources/multimedia/webinars/2190
- Jump start your VHDL verification effort with training
  - Advanced VHDL Testbenches and Verification – OSVVM Boot Camp
  - https://synthworks.com/vhdl_testbench_verification.htm



SynthWorks

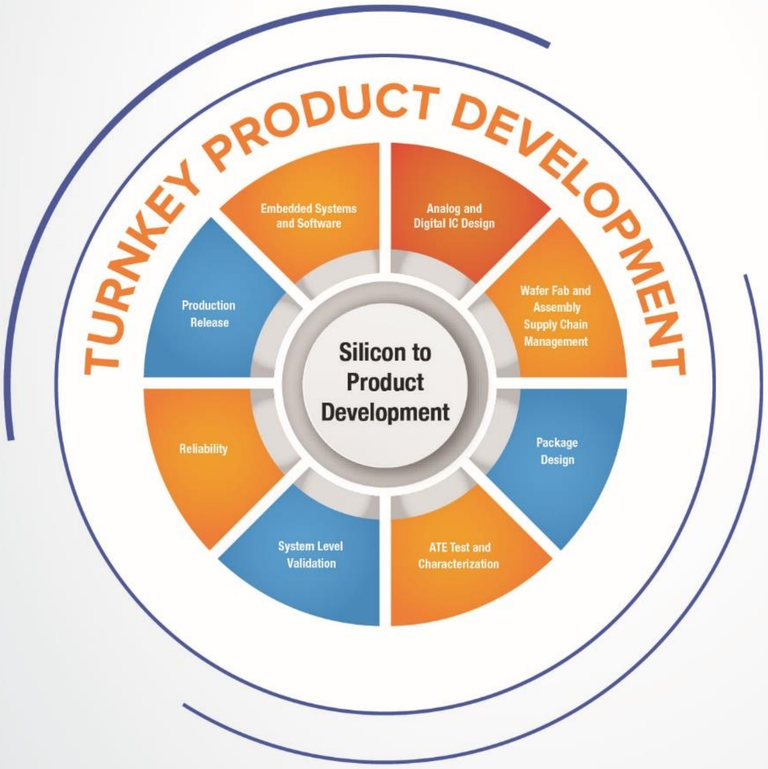
Why OSVVM?

- OSVVM is VHDL's #1 Verification Methodology
- For FPGA Verification,
  - Worldwide: 28% use OSVVM = 50% of the VHDL FPGA users

- © Siemens 2022 https://blogs.sw.siemens.com/verificationhorizons/2022/11/21/part-6-the-2022-wilson-research-group-functional-verification-study/

TESSOLVE

A HERO ELECTRONIX VENTURE


TURNKEY PRODUCT DEVELOPMENT
Silicon to Product Development
Embedded Systems and Software
Analog and Digital IC Design
Wafer Fab and Assembly Supply Chain Management
Package Design
ATE Test and Characterization
System Level Validation
Reliability
Production Release

Tessolve is the leading engineering service/ solution provider with 3000+ employees worldwide and a full breadth of pre-and post-silicon expertise. Tessolve provides a one-stop-shop solution with full-fledged hardware and software capabilities, including its advanced silicon and system testing labs. We have Test labs in India, the US, Malaysia, and Singapore.

Tessolve offers complete Turnkey ASIC Solutions, from design to packaged parts. We are actively investing in the R&D center of excellence initiatives such as 5G, mmWave, high power PMICs, HSIO, HBM/3D/Chiplets, system-level tests, advanced verification methodologies, and others.

Tessolve also offers end-to-end product design services in the embedded domain from concept to manufacturing under an ODM model with application expertise in Avionics, Automotive, Data Centre/ Enterprise, Industrial/IoT, and Wireless segments. We are ISO 9001:2015 certified and our Embedded team is ISO9001 & EN9100 Quality certified.

Visit www.tessolve.com for more information.

Chip Design

700+ Engs

180nm 3nm↕
Advanced Process node down to 7nm

VIPs
Verification IP (VIP) solution for different Protocols & IPs

Test/Product Engineering

950+ Engs

900+
Test programs released

30+
Automated Test Equipment supported

Hardware Development

225+ Engs

130+
PCB designs per month

60+
Experience with 60+ layers PCB design + SI + PI

Embedded Systems

250+ Engs

30+
Embedded Systems released

50%
Over 50% employees with 10+ years' experience

Mike Bartley
Senior Vice President
VLSI Design
+44 (0)779 630 7958
mike.bartley@tessolve.com

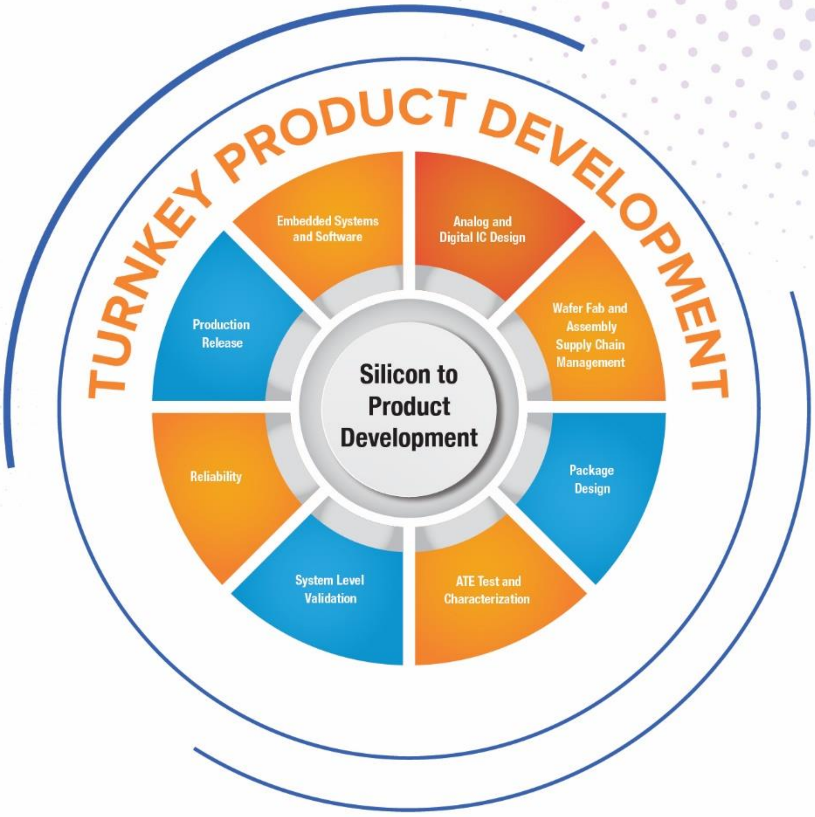
Marc Waugh
Director Sales &
Marketing
512-461-4017
marc.waugh@tessolve.com

Tessolve DTS Inc
4210 South Industrial Drive
STE 140
Austin, TX 78744, USA
Email: sales@tessolve.com

Notes

TESSOLVE

SILICON AND SYSTEMS SOLUTION PARTNER



TURNKEY PRODUCT DEVELOPMENT

Silicon to Product Development

Embedded Systems and Software

Analog and Digital IC Design

Wafer Fab and Assembly Supply Chain Management

Package Design

ATE Test and Characterization

System Level Validation

Reliability

Production Release

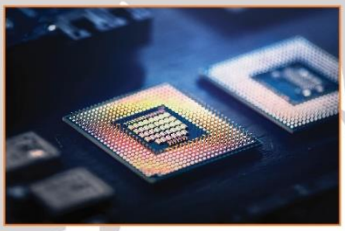
INDUSTRY FOCUS

Automotive

Data Centre/ Enterprise

Industrial/IoT

Avionics



Semiconductor

SCAN ME

https://www.tessolve.com/verification-futures/vf2022/

We hope you have found the conference interesting and informative

Slides and recordings will be available on the Tessolve Website

https://www.tessolve.com/verification-futures/vf2022/

VF2022

VERIFICATION FUTURES