Verification Futures Conference

Austin Marriot South

Thursday 12 September 2024

Sponsored by

TESSOLVE
A HERO ELECTRONIX VENTURE

BREKER™

DOULOS

cadence®

SYNOPSYS®

Participating Companies

arm

Bitstar

CONDOR

EmLogic

OPENHW GROUP
PROVEN PROCESSOR IP

PLANORAMA DESIGN

RENESAS

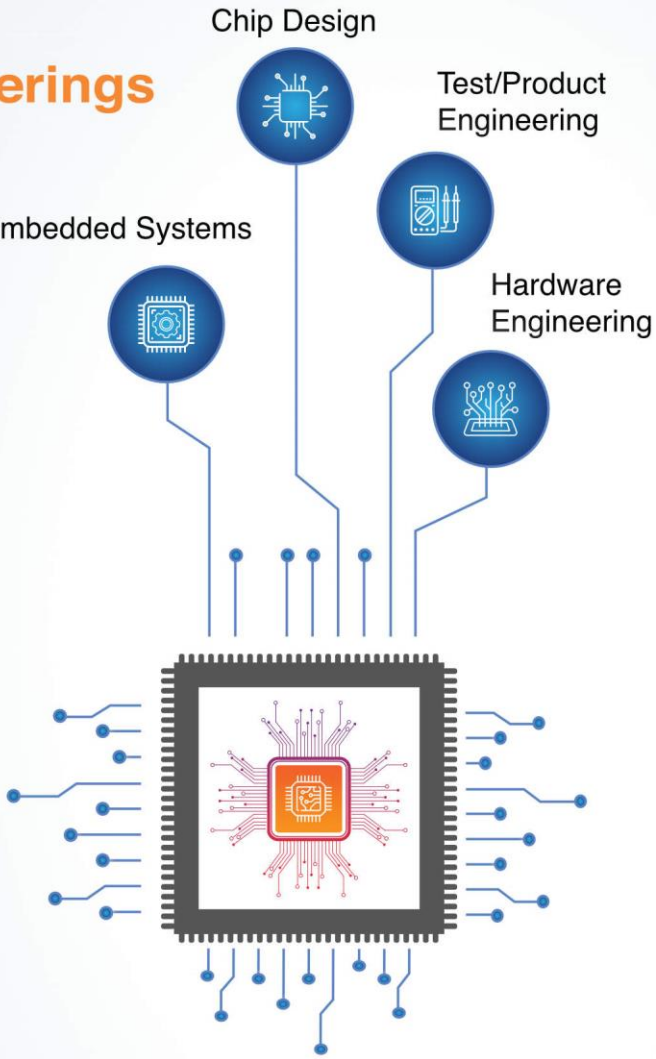
tenstorrent

TEXAS A&M UNIVERSITY

University of Houston Clear Lake

UNIVERSITY of NEBRASKA LINCOLN

Offerings



TESSOLVE
A HERO ELECTRONIX VENTURE

Great Place To Work. Certified OCT 2023 – OCT 2024 INDIA

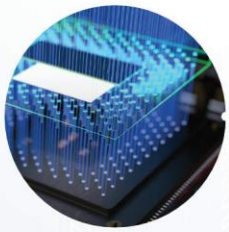
Silicon & Systems Solution Partner

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

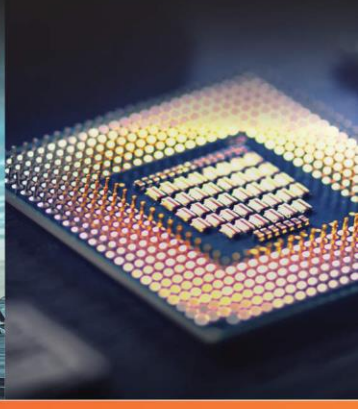
Automotive

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**
Hemendra Talesar (Startups (Bitstar Technologies, Planorama Design, others))

10:15 User Top Verification Challenges

Vijay Kanumuri (Renesas Electronics)

10:30 Paul Graykowski (Cadence Design Systems)

11:00 Refreshments and Networking

11.30 **Multi-Track Session**

Track 1 - CPU User Presentations ***[Lonestar Ballroom – Salon A+B]***

11:30 Mike Thompson (OpenHW Group)

11:50 Varun Koyyalagunta (Tenstorrent)

12:10 Vibarajan Viswanathan (Condor Computing)

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 Adnan Hamid (Breker Verification System)

14:00 Dilip Kumar (TessolveDTS Inc)

14:20 Larry Lapides (Synopsys)

14:40 Vikram Khosa (ARM)

15:00 Refreshments and Networking

15:30 **Multi-Track Session**

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

15:30 Nianhang Hu (University of Nebraska - Lincoln)

15:50 Rahul Kande (Texas A&M University)

16:10 Ishaq Unwala (University of Houston Clear Lake)

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

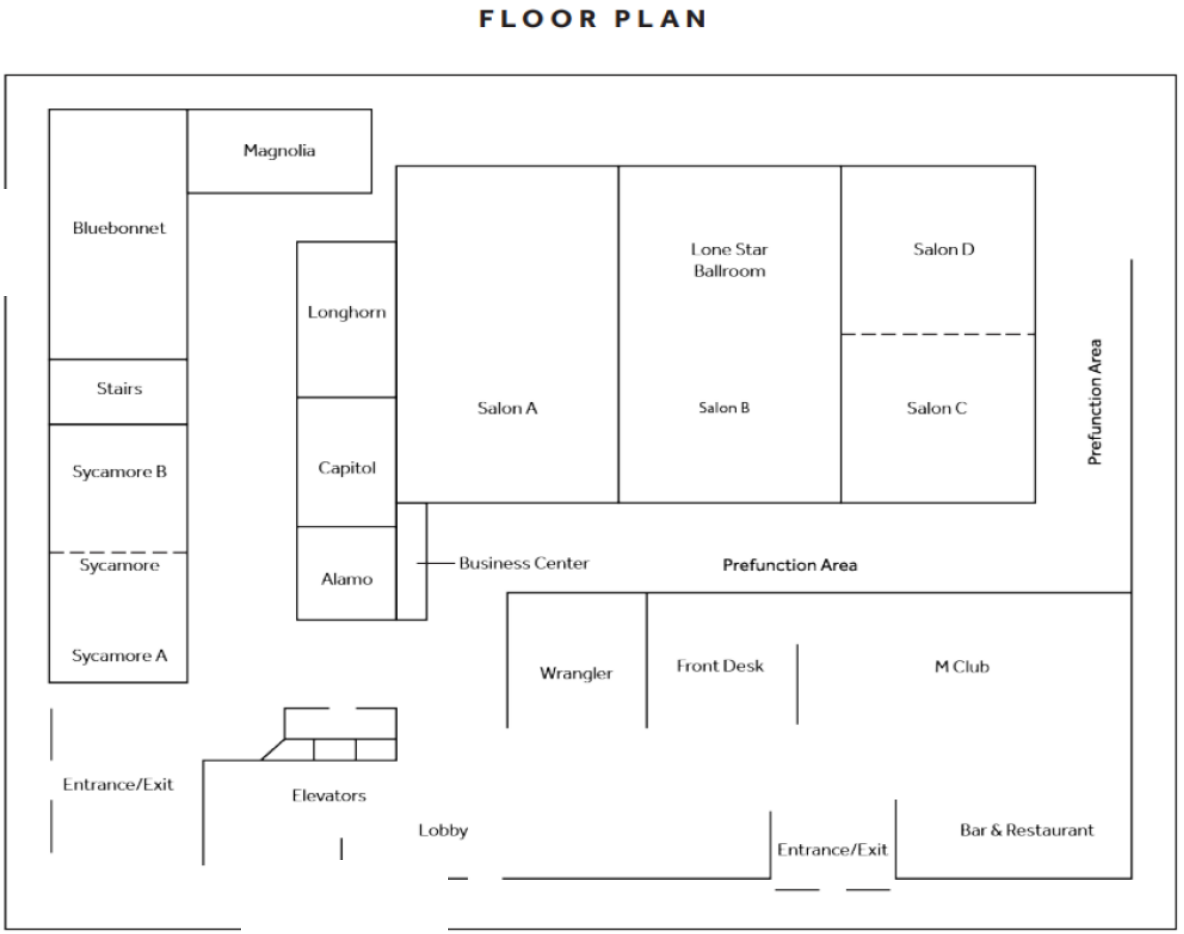
15:30 Doug Smith (Doulos)

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

15:30 Espen Tallaksen (EmLogic) *Remote Presentation*

16:30 Event Closes

Floor Plan


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

Notes

CALL FOR PAPERS

Whatever your specialty, Verification Futures provides an excellent opportunity to share your experiences and insights on the key technical and industry challenges we face in verification.

Submit an Abstract for VF2025

We are now seeking submissions for presentations and papers describing interesting and innovative experiences related to challenges faced in hardware verification. These submissions should include a brief description of the ASIC, SoC and FPGA verification challenges that need to be addresses and/or innovative solutions to improving verification. Abstracts should be targeted toward a technical audience of hardware verification engineers. Abstracts may also address the safety and security issues relating to verification.

Submission Dates

- Call for Papers Opens UK & USA 12 September 2024
- Final Presentations Required UK 31 March 2025
- Final Presentations Required USA 30 June 2025

Abstract Submissions

Abstract submissions should be no more than 2,500 characters and include a short biography of the speaker. All abstracts will be reviewed and notice of acceptance will be sent via email.

To submit your abstract please sent to – mike.bartley@tessolve.com and for guidance please visit https://www.tessolve.com/verification-futures/

About Verification Futures 2025

Verification Futures is a unique one-day conference, exhibition and industry networking event organized by Tessolve to discuss the challenges faced in hardware verification. The event gives the opportunity for end users to define their current and future verification challenges and collaborate with the vendors to create solutions. It's also an excellent opportunity to network and catch up with other verification engineers across Europe & USA.

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 is currently a Senior VP at Tessolve supporting VLSI globally with a focus on helping companies to incorporate the latest verification techniques and strategies into their verification flows and building verification teams to support these companies implement them on IP and SoC projects. He is also responsible for the Tessolve Centres of excellence running all R&D project with Tessolve, including building a new AI capability across all Tessolve products and services.

Mike has a PhD in Mathematics (Bristol University), and 9 MSc's in various subjects including management (MBA), software engineering, computer security robotics and AI, corporate finance, and blockchain and digital currency. He is currently studying part-time for an MSc in quantum computing with the University of Sussex and for the use of technology in healthcare with the University of Glasgow.

Tessolve would like to thank the sponsors and participants of the 2024 Verification Futures Conference

TESSOLVE
A HERO ELECTRONIX VENTURE

Notes

Hemendra Talesara

Startups (Bitstar Technologies, Planorama Design, others)
Advisor

Rethinking Verification Leadership: Ready or Not, Here Comes Generative AI

Keynote Speaker

Abstract
The scale and ever-increasing complexity of chip design in the semiconductor industry have always posed a challenge for verification. We are all too familiar with the escalating costs of missed defects and long verification cycles. Project timelines have made leadership stick to legacy methods and often risk aversive. Unfortunately, sticking to only tried and true methods can put you out of the game. Generative AI is a game-changer technology. Its a disruptive force that will transform chip verification practices. From automation to augmentation, it will complement and enhance our most advanced method available today. We will look at how everything can be accelerated, from brainstorming, test planning, design verification reviews, scenario generation, and improved coverage to finding corner cases. It does have challenges, but nothing unsurmountable. Its time to push boundaries. Let's explore and rethink verification.

Biography
Hemendra is a known name in verification with over 35 years of experience. He worked for IBM, Synopsys, AMD, and many others and led many projects through the entire verification life cycle, from initial architecture exploration to design implementation to tapeout. His projects included CPUs, GPUs, Wireless, Networking, Telecom, Solid State Drive, and various peripheral IPs. He is a Certified Corporate Director, Advisor, and Mentor to startups. He is deeply into disruptive and transformative technology's technical and societal impact. He will explore the implications of next-generation AI on Verification Processes.

PLANORAMA DESIGN Bitstar

Notes

RETHINKING VERIFICATION LEADERSHIP

READY OR NOT, HERE COMES GEN-AI

Hemendra Talesara
Advisor, Mentor
Verification Startups

PLANORAMA DESIGN Bitstar Technologies

WHY GEN-AI IS A GAME CHANGER? /1

- WAY BEYOND AI/ML ANALYSIS
- CREATES NEW CONTENT WITHOUT PROGRAMMING AND INSTRUCTION
- OUT OF THE BOX THINKING/SOLUTIONS
- FEEDS ON DIGITAL DATA (IT IS PLENTIFUL NOW), RECOGNIZES PATTERNS
- FOR EXAMPLE, CHATGPT HAS DIGESTED
  - ALL PUBLIC DOMAIN INFORMATION ON WEB AVAILBALE ON THE DATE OF CREATION
  - MILLIONS OF BOOKS/PUBLICATIONS IN DIGITAL
- PLENTY OF PROOF POINTS
  - PRODUCTIVITY BOOSTER
  - ROUTINE AUTOMATION TO CREATIVE NEW IDEA
  - A COLLABORATOR/PARTNER
  - TOP 4 ROI INCLUDES
    - (1) **SW ENGINEERING (2) R AND D** (3) CUSTOMER OPERATIONS (3) MARKETING AND SALES

WHY GEN-AI IS A GAME CHANGER? /2

- AI IS EVERYWHERE
  - GENERAL PURPOSE TECHNOLOGY
  - MULTITUDE OF APPLICATIONS
- SPEED AND SCALE ARE UNPRECEDENTED
  - GET ON BOARD
    - YOUR COMPETITION HAS
    - DONT BE LEFT BEHIND
  - FASTEN YOUR SEAT BELT
    - GUARD RAILS ARE A MUST
    - MANAGE RISKS (THERE IS PLENTY TO WORRY ALSO)

WHAT DO WE CARE ABOUT?

GOALS

- A FIRST PASS SILICON
- NO SHOWSTOPPER BUG
- WITHIN SHRINKING SCHEDULE
- WITH RESOURCES AVAILABLE
- WITHIN BUDGET ?

MEANS

- TESTS, REGRESSIONS AND DEBUG
- COVERAGE CLOSURE
- COMPLIANCE TO REQUIREMENTS'
- LEGACY AND REUSE
- DOCUMENTATION, REVIEWS AND SIGNOFF

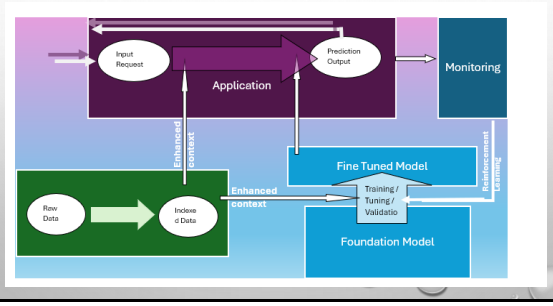
HOW CAN GEN-AI / LLM HELP?

LLMS – LIMITATIONS AND WORKAROUND

- LINGUISTICALLY COHERENT
  - WORLD IS AMAZED AT ITS OUTPUT
  - TRAINED ON BIG DATA;
  - TRILLIONS OF TOKENS DO IT WELL AND QUICKLY GENERATE COHERENT TEXT.
- PERFORM MEMORIZATION
  - SOMETIMES COHERENT RESULTS COULD APPEAR AS REASONED, BECAUSE DATA PATTERNS MADE IT POSSIBLE
  - PROBABILSTIC PREDICTION COULD BE ABSURD (HALLUCINATION)
  - BIASED
- NOT REASON CONSISTENTLY
  - THEY CAN NOT IDENTIFY STRUCTURES
  - THEY DON'T HAVE ANY SEMANTIC UNDERSTANDING
  - THEY DONT THINK SLOW
  - THEY DONT DO MATH
  - THEY DON'T REASON DEDUCTIVELY

- LLMS CAN BE TAUGHT TO REASON TO A DGREE OR IMPROVE ACCURACY
  - DECOMPOSE AND STEPWISE PLANNING, BACKTRACKING
  - ASK IT TO REFLECT ON ITS OUTPUT
  - HUMAN FEEDBACK, CLARIFYING QUESTIONS
  - GROUNDING (WITH CONTEXT ABOUT EXTERNAL WORLD)
  - BETTER TRAINING WITH HIGH QUALITY DATA
  - RAG – IMPROVES ACCURACY, REDUCE HALLUCINATIONS FOR DOMAIN SPECIFIC TASKS (EVEN THOUGH, FUNDAMENTAL REASONING ABILITY REMAINS LIMITED FOR LLM)
  - OTHER ENRICHMENT METHODS

GEN-AI WORKFLOW



A FEW GEN-AI TERMS

- **PROMPT ENGINEERING** – TEACH LLM TO RESPOND TO A QUERY (WITH GUIDANCE)
- **RAG (RETRIEVAL AUGMENTED GENERATION)** – ADDING KNOWLEDGE FROM EXTERNAL SOURCE (OUR PDF, SPECS, PLANS ETC.)
- **REINFORCEMENT LEARNING WITH HUMAN FEEDBACK – (RLHF) –** IT'S A WAY TO ALIGN AN AI AGENT TO HUMAN PREFERENCE BY MONITORING THE OUTPUT AND PROVIDING THE FEEDBACK.
- **FINE TUNING** – IMPROVING A PRE-TRAINED FOUNDATION MODEL WITH NEW EXAMPLES FOR SPECIFIC TASKS OR USE CASE
- **TRAINING** – (OR PRE-TRAINING) RETRAIN OR CREATE A NEW MODEL TO MEET YOUR NEEDS (EXPENSIVE)
- **LARGE (OR SMALL?) LANGUAGE MODEL (LLM) –** A TEXT GENERATOR (LIKE AUTO-COMPLETE ON YOUR PHONE) MODEL THAT LEARNS PROBAILISTIC RELATIONSHIP OF WORDS DURING A VERY COMPUTE INTENSIVE TRAINING WITH MASSIVE AMMOUNT OF DATA. IT PREDICTS THE NEXT WORD (AKA TOKEN) STARTING FROM YOUR INPUT TEXT (PROMPT) EG. CHATGPT, LLAMA, GEMINI ETC.
- **FOUNDATION MODEL –** SAME AS LLM, DIFFERENCE IS IN SCOPE AND USE. LLMS ARE CONSIDERED A SUBSET OF FOUNDATION MODEL. WHILE LLM PROCESS LANGUAGE, FOUNDATION MODEL ALSO INCLUDES MULTIMODAL DATA SUCH AS VOICE, IMAGES AND SO ON.

PROMPT ENGINEERING

- PRIMING
- STYLE AND TONE INSTRUCTIONS
- HANDLING ERRORS AND EDGE CASES
- DYNAMIC CONTENT
- OUTPUT FORMATTING

FINE TUNING

- FURTHER TRAINING FOUNDATION MODEL WITH NEW EXAMPLES
  - TEACH INTUITUION WHEN WORD FALLS SHORT
  - BAKE IN STYLE , TONE, AND FORMATTING TO OUTPUTS
  - REDUCING PROMPT LENGTH
  - TRAIN SMALLER MODEL TO PERFORM BETTER ON SPECIALIZED TASKS
  - BIGGER IS SLOWER AND MORE EXPENSIVE
  - PICK RIGHT SIZE MODEL FOR THE TASK
  - NARROW RANGE OF OUTPUTS TO MOST RELEVANT

RETRIEVAL AUGMENTED GENERATION (RAG)

- KNOWLEDGE FORM EXTERNAL (PROPRIETARY) SOURCE
- REMEMBER
  - LLMS DO NOT STORE FACTS. THEY STORE PROBABILITIES
  - THEY PARAPHRASE WHAT IT HAS READ IN PROBABILISTIC MANNER
  - ADDING RIGHT INFORMATION CAN BE POWERFUL

RETRIEVAL AUGMENTED GENERATION (RAG)

- PREP
  - DATA – PDFS, WEB PAGE, (SPECS, PLANS, AND SO ON)
  - SPLIT- CREATE SMALL CHUNKS FROM TEXTS
  - EMBED – CREATE A VECTOR REPRESENTATION OF CHUNKS
  - STORE – SAVE IT

RETRIEVAL
QUESTION
Pre---PROCESS
Go TO EMBEDDINGS
SEARCH RELEVANT
FILTER
BUILD PROMPT
SEND to LLM FOR GENERATION
SELF-REFLECT

FINE TUNING

- DOES NOT TEACH FACTS
- CAN DO WITH SMALLER DATA SETS
- DOES NOT HAVE TO BE TOO EXPENSIVE
- DOES NOT HAVE TO BE COMPLICATED
- IT CAN WORK ALONG WITH RAG

AGENTIC REASONING

BY ANDREW NG

- ZERO SHOT – WRITE A PROMPT, GET AN ANSWER; WRITE A CODE IN ONE ATTEMPT, NO BODY WRITES LIKE THAT. (3.4 48%, 4.0 67%,
- PROVIDE A STEP WISE PLAN, AN ITERATIVE APPROACH -> REMARKABLY BETTER RESULTS
- MULITPLE AGENTS WITH DIFFERENT ROLES WORKING TOGETHER COLLABORATIVELY – DON'T DO IT ONE TAKE.
- YOU DO NOT WRITE A CODE IN FIRST ATTEMPT. YOU WRITE AND REVISE, DEBUG AND ITERATE. HELP MODEL DO THE SAME
- CHOSING BENCHMARK

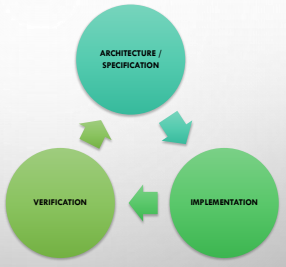
- GROQ – 800 TOKENS PER SECOND – LOT OF TIME IN INFERENCE RUNNING
- HYPER INFERENCE SPEED READING EACH OTHERS VERY FAST
- GET EMBEDDING PART IMPROVE

AGENT PATTERNS

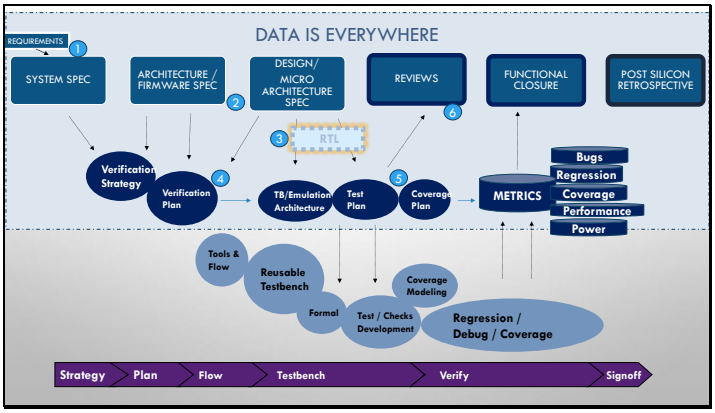
BY ANDREW NG

- REFLECTION – REFLECT ON OUTPUT, FIND A WAY TO IMPROVE IT AND ITERATIVE REFINEMENT YOU CAN EVEN AUTOMATE YOURSELF.
  - (REFLEXION: LANGUAGE AGENTS WITH VERBAL REINFORCEMENT LEARNING – SHINN 2023
  - SELF-REFINE: ITERATIVE REFINEMENT WITH SELF FEEDBACK – MADAAN 2023
  - 2ND AGENT CAN DO UNIT TEST REVIEWER
- TOOL USE – FUNCTION CALL – MATH LIBRARY (LLM MAY NOT BE GOOD AT IT), COPILOT
  - HARD CODED, NOT LLM, API CALLS, LIBRARIES – EXISTING TOOLS
  - LLM CONNECTED WITH MASSIVE API
- PLANNING – CHAIN OF THOUGHTS PROMPTING ELICITS REASONING IN LLM – THINK SLOWLY – ASK THEM TO EXPLAIN THEIR STEPS
  - HUGGINGGPT PAPER
  - RESEARCH AGENTS
  - USUALLY CAN RECOVER FROM EARLIER ERROR
- MULTI-AGENT COLLABORATION – MULTIAGENT DEBATE – CHATDEV – ASSIGN ROLES – COLLABORATE AND EXTENSIVE CONVERSATION
  - COMMUNICATIVE AGENTS FOR SW DEVELOPMENT, AUTO GEN
  - CHATGPT VS. GEMINI FINE TUNED MODEL CAN DEBATE

VERIFICATION: WHERE IS THE DATA?



DATA IS EVERYWHERE



VERIFICATION GEN-AI PROOF POINTS

- **PLANORAMA DESIGN (BASED IN AUSTIN, TEXAS) CEO, MATT GENOVESE**
  - BEST DEMONSTRATES USE OF RAG, FINETUNING, PROMPT ENGINEERING TO ADVANCE VERIFICATION METHODOLOGY
  - RISC-5 DEMO ON YOU TUBE (**LINK**)
    - ABILITY TO ACCESS DISPERSED DATA ACROSS THE DEPARTMENT
    - RISC 5 INSTRUCTION TEST GENERATION
    - STRUCTURED TEST PLAN REVIEW
      - FINDING GAPS BETWEEN SPEC AND THE PLAN
    - FINDING BUG IN RTL
    - FAST INFORMATION RETRIEVAL RELATED TO BUG AND SPECS
    - VERILOG CODE GENERATION
    - A BRAINSTORMING PARTNER
  - EDA AUTOMATION RETHINK
  - WORKING WITH SEMICONDUCTOR ORGS
    - TO ENABLE CUSTOMIZED GEN-AI FLOW FOR DESIGN AND VERIFICATION

APPLICATION OF THE GEN-AI
ADVANCED CONCPETS TO VERIFICATION

- TBD

VERIFICATION GEN-AI PROOF POINTS

1. QUALCOMM'S AI-DRIVEN VERIFICATION:
   - QUALCOMM USES AI TO ACCELERATE COVERAGE CLOSURE IN CHIP DESIGN VERIFICATION
   - AUTOMATING THE GENERATION OF TESTBENCH STIMULI
   - PROVIDING ANALYTICS TO ENHANCE TESTBENCH QUALITY
2. AWS GENERATIVE AI FOR SEMICONDUCTOR DESIGN
   - AUTOMATING TASKS LIKE CODING, DEBUGGING
3. NVIDIA
   - USING AI IN VERIFICATION AND DESIGN
4. GOOGLE
   - USING AI TO DESIGN NEW CHIPS

VERIFICATION GEN-AI PROOF POINTS

- **UVM AND SYSTEM VERILOG TESTBENCH**
  - TRIALS OF USING GENERATIVE AI FOR **APB UVM TESTBENCH GENERATION** BY DIANA DRANGA , ROMANIAN JOURNAL OF INFORMATION TECHNOLOGY AND AUTOMATIC CONTROL, VOL. 34, NO. 2, 75-84, 2024
- **SYSTEM VERILOG GPT**

RETHINK VERIFICATION EADERSHIP
AI CHALLENGES

- **AI FEEDS ON DATA**
  - GET HANDLE/ORGANIZE YOUR DATA
    - DATA COULD BE ANYWHERE INCLUDING OUTSIDE YOUR ORGANIZATION
  - QUALITY MATTERS
  - CLEAN YOUR DATA/DOCUMENTATION
    - ITS ABOUT TIME
    - SPECS/PLANS WHERE MANY BUGS HIDE
  - REMEMBER - GARBAGE IN, GARBAGE OUT
  - PROTECT YOUR DATA

- **AI SKILLS UPGRADE**
  - START AT THE TOP (Responsibility will include)
    - DEFINE NEW VERIFICATION ARCHITECTURE
    - FRAMEWORK FOR MANAGING DATA
    - DECISION AUGMENTED BY GEN-AI ADVISE
    - LEGAL RAMIFICATION OWNERSHIP
      - ANY INADVERTANT MISUSE
      - MISINFORMATION
      - ACCURACY, TRUSTWORTHINESS
      - JUSTIFICATION
      - (NOT THE MODEL OR ITS IMPLEMENTER)
  - HIRE DATA ENGINEERS/SCIENTISTS TO THE TEAM
  - RESKILL – EVERY ENGINEER SHOULD LEARN AI

RETHINK VERIFICATION EADERSHIP
AI CHALLENGES

- **PILOTS**
  - DENTIFY STRONG CANDIDATES
  - STRATEGIZE to EXPAND FROM PILOTS
  - Don't HESITATE TO PARTNER WITH EXPERTS
- **GOVERNANCE FRAMEWORK**
  - PROTECTION, SECURITY, COPYRIGHT, ABUSE
  - Transparency, TRACEBILITY, EXPLAINABILITY

- **A NEW VERIFICATION ARCHITECTURE – EMBRACE IT**
  - WORK WILL CHANGE
  - TOOLS WILL CHANGE
  - PROCESS WILL CHANGE
  - FLOW WILL CHANGE
  - PARTNER WITH AI EXPERTS, IF NEED BE

- **CHANGE (CULTURE) MANAGEMENT**
  - AI FLOW COULD BE DIFFERENT
    - ITERATION AND EXPERIMENTATION
    - AGILE IS A MUST
  - **READY THE TEAM (AND MANAGEMENT) TO EMBRACE NEW WAYS**
    - ALL JOBS/RESPONSIBILITY WILL CHANGE (INCLUDING YOURS)
    - TEAM ORGANIZATION WILL CHANGE
    - COLLABORATION MORE THEN EVER BEFORE

TAKEAWAYS

- ✓ BIG ACCELERATION OF
  - ✓ SPEC/PLAN GAPS REVIEW
  - ✓ TEST GENERATION
  - ✓ COVERAGE CLOSURE
- ✓ IMPROVED PROCESS AUTOMATION
- ✓ FUTURE PROOF VERIFICATION ARCHITECTURE

LEADERSHIP MANTRA: RETHINK, RESKILL AND REORGANIZE

CONTACT US

WWW.PLANORMA.DESIGN.

HEMENDRA.TALESARA@PLANORAMA.DESIGN

WWW.BITSTAR-TECH.COM.

HEMENDRA.TALESARA@BITSTAR-TECH.COM

HEMENDRA.TALESARA@GMAIL.COM, FOUNDER @ 2ND VOICE

MOBILE: (512) 657-7520

Vijay Kanumuri

Renesas Electronics

Senior Principal

Challenges in Verification for Automotive Applications

Challenge Paper

Abstract

Design verification targeted for Automotive SoCs provides several challenges spanning technical, compute, safety, reliability, advanced technologies and complexity of modern vehicles. We will discuss these aspects as part of presentation.

Biography

Vijay has a rich 16 years of experience in leading Design Verification teams across SOC Design Verification, Platform IP Design Verification, HSIO/Networking/Infrastructure IPs domains. In various capacity as Design Verification lead, he has led verification across many breakthrough products and platform definition in Automotive and Industrial space. He joined Texas Instruments after his graduation in master's in computer engineering from UT Austin. In his role starting in IP development and leading SOC DV, he has been instrumental in developing a platform methodology for verification across multiple product lines as well as leading end-to-end SOC DV execution for successful 16nm Automotive ADAS SOCs including post-silicon Test development. He led vendor Technical Review Meetings, evaluating, onboarding new tools, and driving architecture, test plan reviews and execution for first pass silicon success. In his previous role as Design Manager responsible for HSIO/Networking/Infrastructure IPs, he was leading the architecture for low-cost platform and networking IPs, IP procurement and Management Review meetings with external vendors to drive roadmap decision and alignment.

Notes

AUTOMOTIVE VERIFICATION CHALLENGES

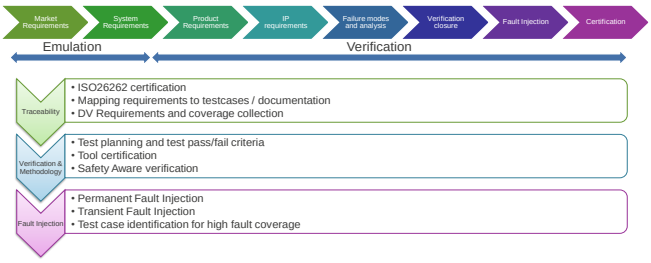
VIJAY KANUMURI
Senior Principal - Renesas

RENESAS

AUTOMOTIVE VERIFICATION HOTSPOTS

RENESAS

SAFETY & COMPLIANCE



RENESAS

PERFORMANCE & LATENCY

Infotainment ADAS

- AI / Video / Display / APU performance – High bandwidth
- DDR Performance and efficiency
- Structural Latency calculation across multiple NoCs
- In built performance monitors in VIPs or EDA tools that gather
- Long simulation times for usecase traffic

Gateway Domain Controller MCU

- Real Time deadlines and latency measurement
- Mix of real time and non-real time traffic scenarios
- DMA performance with peripherals (UART/CAN..)
- Built-in monitor to flag error for failing deadlines
- Accuracy of measurements and modeling of variability

RENESAS

LOW POWER

Functional Verification

- Understanding System requirements and breaking it down to Unit and SoC verification
- Power domain dependencies and validation of the spec
- Power measurement using EDA tools from waveform and annotation to gates
- Custom VIP development and checkers for state transitions. Need scalability across hierarchies
- Clock gating complexity for dynamic power reduction

UPF Power Aware

- Late in the design cycle
- Isolation values and spec content need to be done ahead of UPF sims
- Slower run times and tool dependencies with x-prop and UPF enabled
- Power domain definition (including dependencies) and functional intent
- Library support for Power Aware RTL

Paul Graykowski

Cadence Design Systems

Product Marketing Director

AI and GenAI for Verification Productivity

Platinum Sponsor

Abstract

The chip and system design industry faces a growing productivity challenge, with complexity on pace to increase 100X over the next decade. The complexity is outpacing the industry's ability to train new engineers, presenting companies with substantial risk. AI is being actively applied to the productivity challenge, showing great promise in improving efficiency by ten times or more. Cadence is further addressing this challenge through a series of advancements, which include accelerated compute, application of high-speed engines, and optimization with AI and Generative AI. EDA can benefit greatly by applying AI with the engineer-in-the-loop approach. This presentation will explore Cadence's AI accelerators for debug and formal proof analysis. In addition, we will also examine how a Retrieval-Augmented Generation (RAG) integrated with large Language Models (LLMs) can significantly reduce LLM hallucinations. Finally, we will delve into the application of copilots for assisting verification engineers with documentation, specifications, and planning.

Biography

Paul has over 25 years of expertise in the design and verification of SoCs. His diverse career encompasses technical roles in corporate and field application engineering as well as technical and product marketing. Currently, Paul is focused on product marketing and management for Cadence's Xcelium Simulator. Previously, he drove the application of IPXACT and Network on Chip at Arteris. During his 20-year tenure at Synopsys, Paul specialized in Verification IP and methodologies, evolving from Vera and VMM to SystemVerilog and UVM. His professional journey also includes positions at Compaq (HP), Intel, and Cirrus Logic. Paul earned his BSEE from Texas A&M.

cadence®

cadence®

Verisium AI-Driven Verification

Leverage big data and AI to optimize verification productivity and efficiency

AI and GenAI for Verification Productivity

Paul Graykowski, Product Marketing Director, System and Verification Group
September 2024

cadence

Productivity Challenge in Chip and System Design

Complexity Is Increasing
100X
Over the Next Decade

Source: Cadence analysis

More Annual Design Starts
4X
Over the Next Decade
(<=10nm)

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

Not Enough Engineers

Semiconductor Engineering

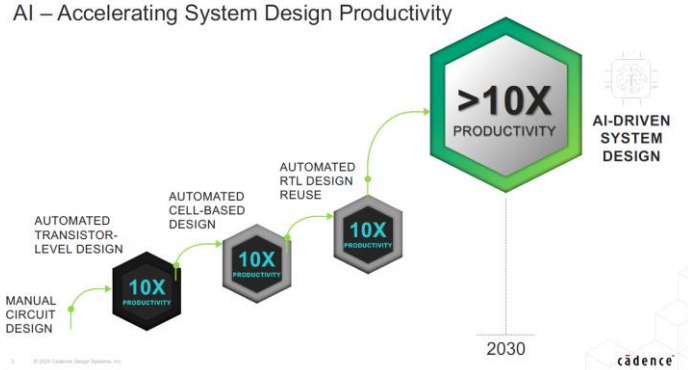
Engineering Talent Shortage Now Top Risk Factor

FEBRUARY 25TH, 2019 - BY: MARK LAPEDUS

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

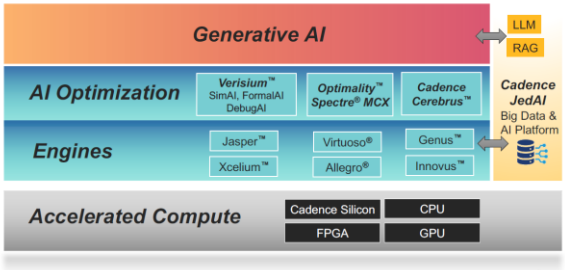
cadence

AI – Accelerating System Design Productivity



cadence

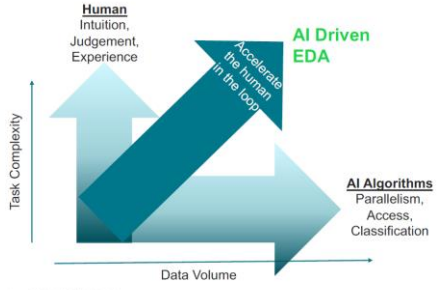
Cadence AI Strategy



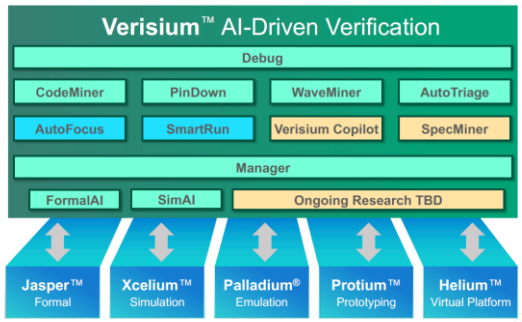
cadence

Power of AI-Driven *Solutions*

Catalyze the engineer

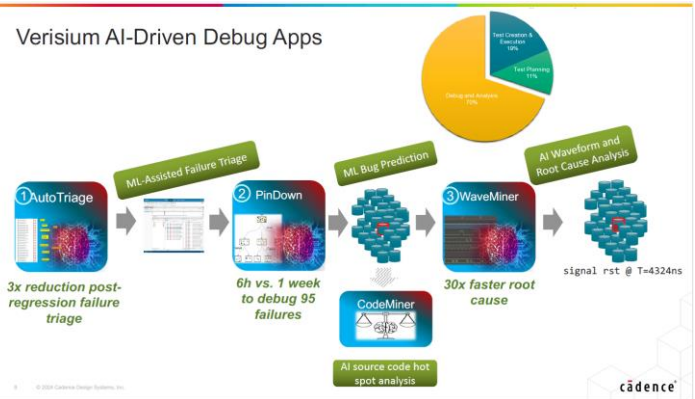


Verisium AI-Driven Verification Platform



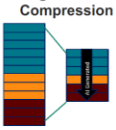
Major AI Flows for Verification

Verisium AI-Driven Debug Apps



Verisium SimAI

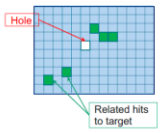
Regression Compression



2-6x speed up

Train ML model; Generate stimulus

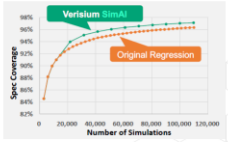
Coverage Maximization



New bins covered 2x fewer sims

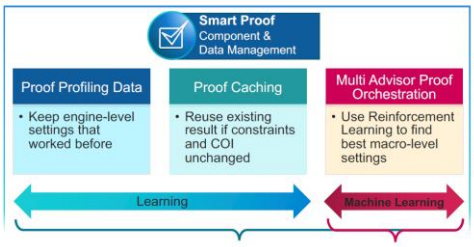
Simulation CPU hours	Baseline	Verisium SimAI	Speed-up
IP 1	19	1	19x
IP 2	3,588	597	6.0x
IP 3	12,807	4,396	2.9x
IP 4	26	2	13x
IP 5	3,810	536	7.1x

Qualcomm – "We observe as much as 20x reduction in CPU cycles and find SimAI very useful for hard to hit cross coverage bins."
CadenceLive Silicon Valley 2024



cadence

FormalAI: Smart Proof Automation Framework



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

Optimizes subsequent runs/regressions **Optimizes out-of-the-box proofs**

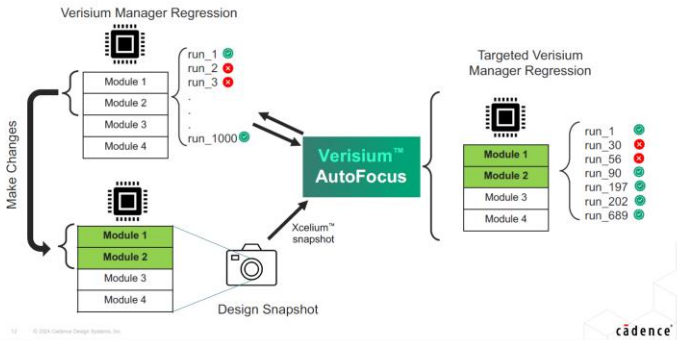
Find more bugs Better convergence Faster proofs

cadence

Resource Optimization

cadence

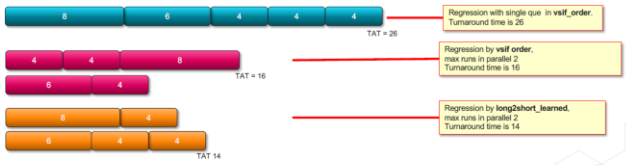
An enhanced flow with AutoFocus



cadence

SmartRun

- The goal of **Smart Run** is to provide the fastest turn around time for a regression and improving verification throughput
- Flow
  - **Learning** the typical run durations for tests
  - **Reports** that allow the user to decide what queuing policy and farm resources to apply to optimize the regression throughput.
  - **Reshuffle** and launch new regressions using the new queuing policies.



cadence

Generative AI for Verification

cadence

LLM Expectations and Reality

Bot: How can I help you today?

User: Write formal properties for my 3 channel round robin arbiter

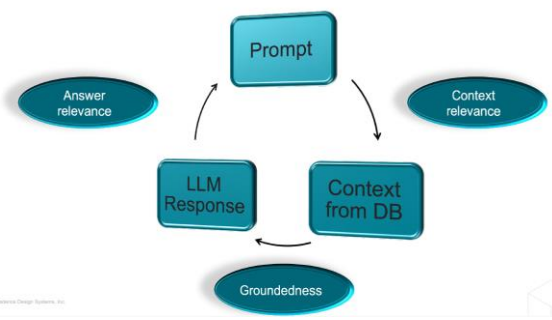
Bot: Sure, here's your formal property:

```
// Assert that the arbiter
always grants access to the next
requester in round robin order
assert property (@(posedge clk)
disable iff (!rstn)
(req0.granted &&
 req1.granted &&
 req2.granted) |=>
req0.granted ##1
req1.granted ##1
req2.granted);
```


cadence

Mitigating Hallucinations with RAG and LLM as Judge

Retrieval Augmented Generation



cadence

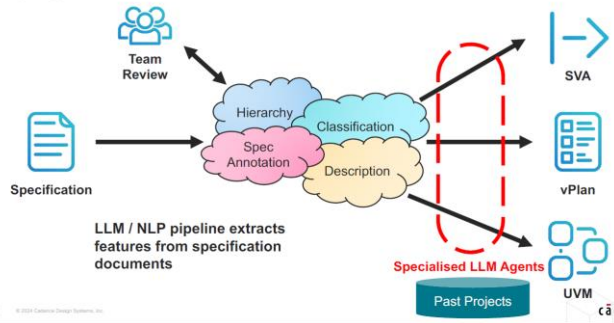
Generative AI Flow for SVA

- Syntax issues
- Equivalence
- Intersection
- Implication
- Conflicts
- Ranking
- Clustering

cadence

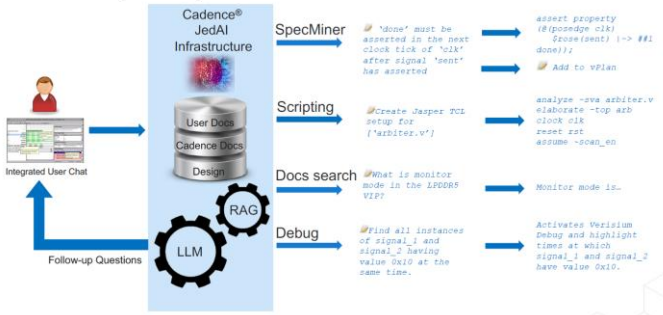
Verisium SpecMiner: AI Driven Verification Generation

Ongoing Research



cadence

Verisium Copilot Capabilities

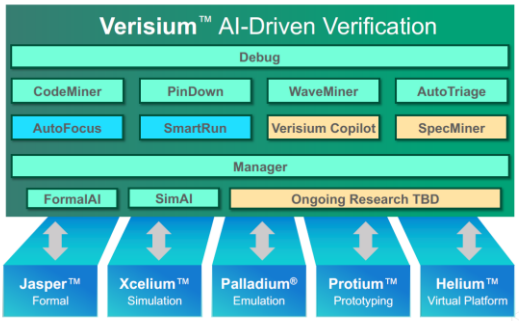


cadence

Summary

cadence

Verisium AI-Driven Verification Platform



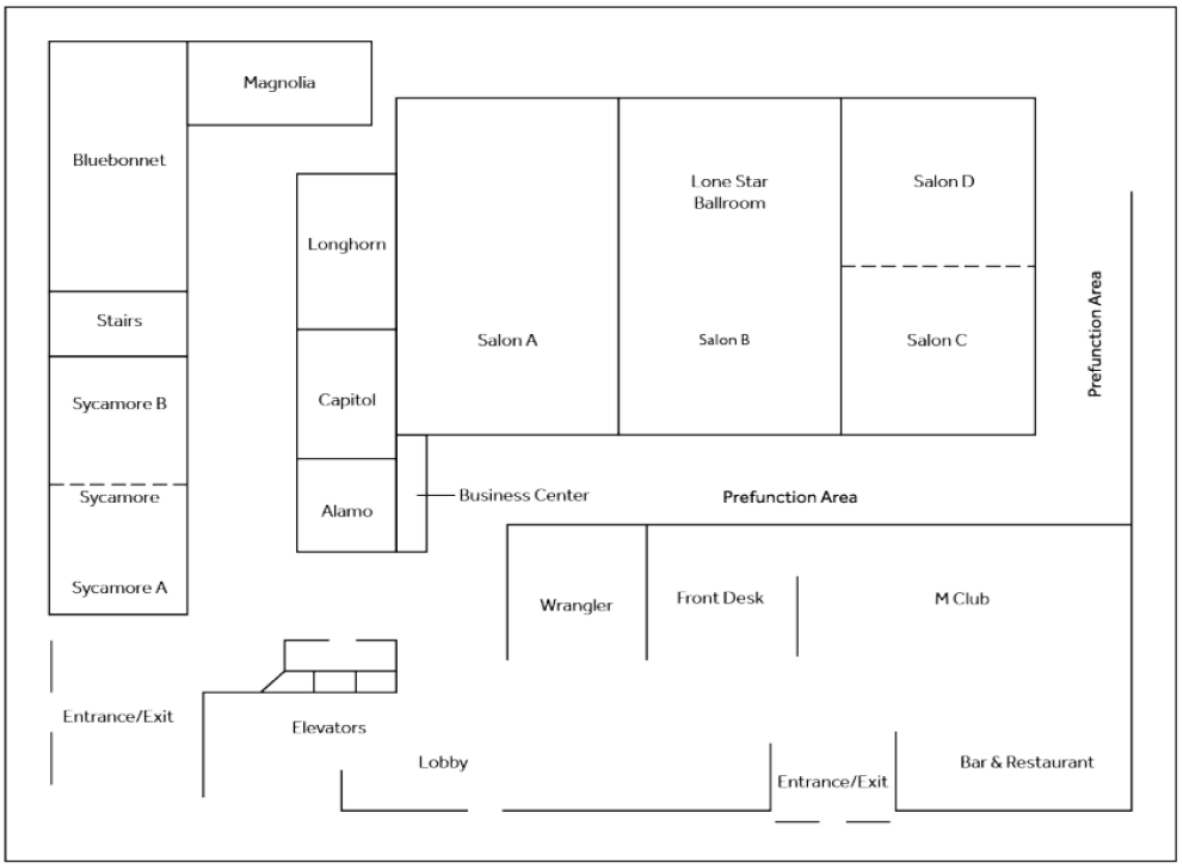
cadence

cadence®

Track Session

CPU 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 Thompson

OpenHW Group

Director of Engineering, Verification Task Group

Accelerate your adoption of RISC-V with CORE-V-VERIF

User Paper

Abstract

CORE-V-VERIF is an open-source project supported by the OpenHW Group. Its goal is to provide an open-source environment and work-flow that can be deployed onto any RISC-V processor core. Since December of 2020, OpenHW Group members have successfully used CORE-V-VERIF for end-to-end verification of more than six RISC-V cores.

Biography

Mike is a functional verification engineer and manager who has been involved in all aspects of the discipline: simulation, emulation, prototyping and formal verification. He is strong proponent of coverage driven processes in the pursuit of first-time-right silicon.

Notes

OpenHW Group

Proven Processor IP

Accelerate your adoption of RISC-V with CORE-V-VERIF

Mike Thompson
mike@openhwgroup.org

 September, 2024

Objectives

- Introduce the OpenHW Group, who we are and what we do, with specific emphasis on functional verification of RTL models of RISC-V cores.
- Introduce "CORE-V":
  - CORE-V (pronounced "core five") is the family of RISC-V processor cores curated by the OpenHW Group.
  - Written in SystemVerilog.
  - Permissively licensed open-source code and documentation.
  - Quantitative verification metrics.
- Take-aways:
  - The OpenHW Group applies industry leading tools and methodologies to fully verify its open-source RTL IP.
  - CORE-V-VERIF is an open-source SV/UVM verification environment for RISC-V cores.
  - CORE-V-VERIF can be used as-is, or by using select components such an Instruction Stream Generator, UVM Bus Agents, Assertion Libraries and Functional Coverage.

OPENHW GROUP and CORE-V

- The OpenHW Group is a not-for-profit, global organization.
- The OpenHW ecosystem is driven by members (corporate & academic) bringing together HW and SW designers collaborate in developing open-source cores, related IP and SW.
- Primary focus is the CORE-V Family of open-source RISC-V processors.
- International footprint with developers in North America, Europe and Asia.
- Strong support from industry, academia and individual contributors worldwide.

Is Open Source for You?

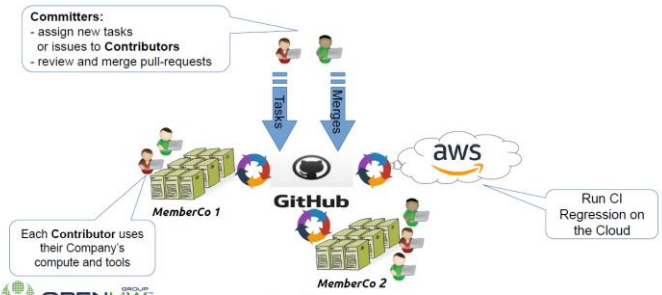
- Permissively licensed open-source IP provides the "freedom to innovate":
  - Reduces the barrier to adoption.
  - Enables proprietary extensions of open-source reference implementations.
  - Promotes "competitive collaboration" through sharing of development resources.
  - Reduces productization costs.
- Open-Source does not mean "free lunch":
  - You will need commercial tools to run CORE-V-VERIF.
  - Engineers do not work for free.
  - Closing coverage requires tens-to-hundreds of thousands of compute-hours.
- CORE-V benefits those organizations that:
  - Do not view processor IP as a differentiator for their business.
  - Can take advantage of potential cost savings of using open-source IP.
  - Wish to develop or demonstrate technical capabilities.

OPENHW GROUP Members

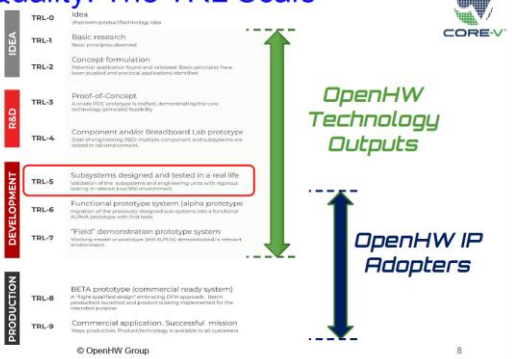
A Different Kind of Open Source Company

- The OpenHW Group is all about Industrial Grade practises for Industrial Grade Products:
  - Use of commercial EDA tools and methodologies.
  - Comprehensive and usable documentation.
  - Well implemented RTL.
  - Quantitative verification metrics.
- You get out what you put in:
  - The OpenHW Group does not employ a large staff of design, verification and implementation engineers.
  - 95% of the engineering effort, EDA tooling and compute infrastructure used to develop CORE-V IP comes from our Members.

Task Assignment, Execution and CI



Qualitative Quality: The TRL Scale

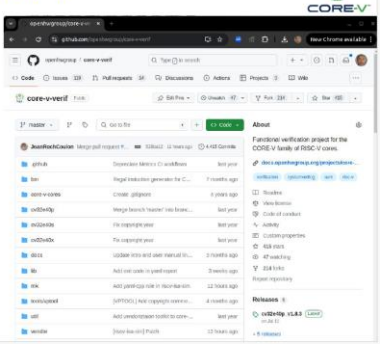


Quantitative Quality: TRL-5

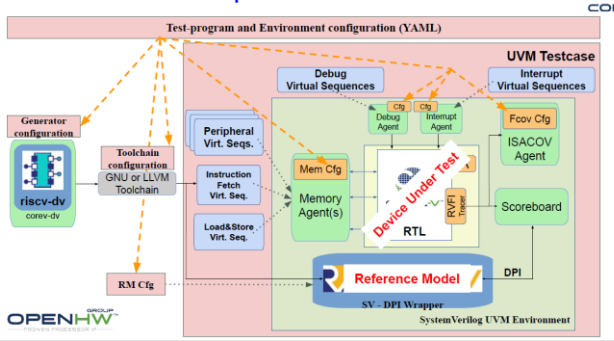
- An OpenHW development project has achieved TRL-5 when it has completed all the deliverables defined in the OpenHW Group TRL5 for COREV RTL IP Template checklist:
  - Find it on GitHub at https://github.com/openhwgroup/programs (link)
- This is a spreadsheet with worksheets for:
  - Documentation
  - IP Licensing
  - RTL Design
  - Simulation Verification
  - Formal Verification
  - Software Requirements

What is CORE-V-VERIF?

- A GitHub repository!
  - Directly supports verification of most CORE-V cores.
- Three usage models:
  - **Direct** - instantiate your RTL directly into CORE-V-VERIF.
  - **Indirect** - clone CORE-V-VERIF structure into your environment.
  - **Library** - re-use select components as desired.



CORE-V-VERIF Implementation

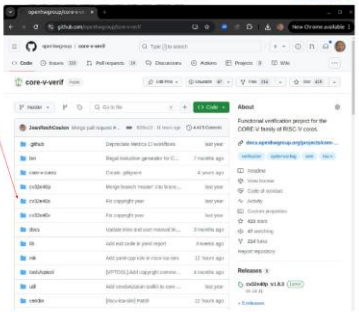


Important Points

- Everything represented in the previous slide is permissively licensed open-source:
  - RTL code for the core.
  - User Manual for the core.
  - Testbench code.
  - Agents
  - Testcases.
  - Assertions.
  - Functional coverage.
- The Reference Model is the exception:
  - **Commercial Option:** ImperasDV from Synopsys.
  - **Open-Source Option:** custom Spike extension.

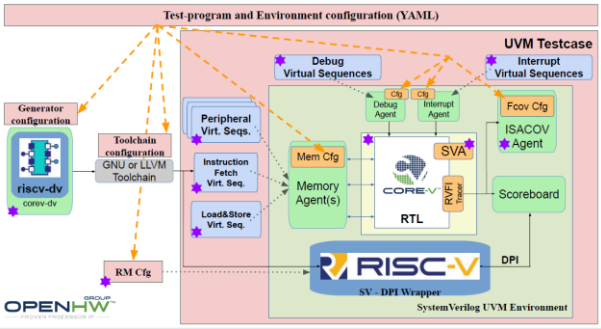
Usage Models: **Direct**

- Add a new "core-specific" directory in CORE-V-VERIF for your specific core.
- Create a core-specific testbench layer.
- Add core-specific extensions.
- Pros:
  - Gets you up and running fast.
  - Reuse extensively library of scriptware for running simulations, regressions, etc.
  - 80% of the way there for 20% of the effort.
- Cons:
  - Verification environment is not in the same repo as RTL.
  - That 20% is a lot of work!



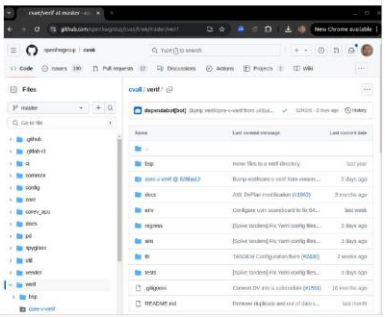
Direct use of CORE-V-VERIF

…everything marked with a "✱" can be a core-specific extension.



Usage Models: **Indirect**

- Use CORE-V-VERIF as part of your core's repository.
- Allows for more customization.
- Currently only used by the CVA6, which is a highly parameterizable, application-class core.



Use CORE-V-VERIF as a **Library**

Summary / Wrap-up

- CORE-V is a family of permissively licensed open-source RISC-V cores curated by the OpenHW Group.
- CORE-V IP is Industrial Grade.
- **"There is no *they*"**: benefitting from open-source means getting involved.
- CORE-V-VERIF is the SV/UVM verification environment for CORE-V IP.

Thank You

Notes

Varun Koyyalagunta

Tenstorrent

Design Verification Engineer

Accelerating RISC-V testbench development with open source RISC-V RTL and emulation

User Paper

Abstract

Today's shorter product time to market makes silicon verification runway shorter. Tenstorrent is working on CPUs based on RISC-V architecture for many AI applications. Since this is an emerging processor environment having RTL ready is not an easy task. Once RTL is available the testbench should be ready for both simulation and emulation workloads. Also, we should have all test collaterals ready to go, which involves firmware, drivers, applications etc.

Biography

Today's shorter product time to market makes silicon verification runway shorter. Tenstorrent is working on CPUs based on RISC-V architecture for many AI applications. Since this is an emerging processor environment having RTL ready is not an easy task. Once RTL is available the testbench should be ready for both simulation and emulation workloads. Also, we should have all test collaterals ready to go, which involves firmware, drivers, applications etc.

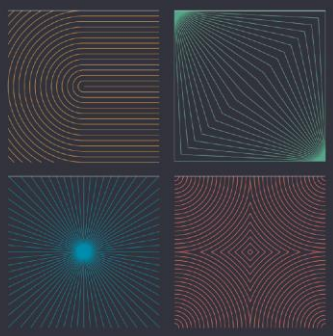
At Tenstorrent we solved this problem by adopting RTL from RISC-V open source. This enabled us to shift left the emulation and simulation testbench creation. We use a standard memory interface, AXI, standard instruction interface, RISC-V Formal Interface (RVFI), and the open source CVA-6 RISC-V cpu to develop testbench architecture and collateral in advance with full architectural instruction-by-instruction checking.

This helped us complete the testbench development and test infrastructure ready without our custom CPU RTL. When the inhouse RTL is ready, we could be able to replace our custom CPU RTL with open source CVA-6 processor.

This methodology helped us significantly shift left the testbench and test infrastructure readiness. Due to this, we could able to innovate in the area of test collateral creation, making emulation ready infrastructure and were confident to run application level tests the minute RTL was available. We used ZeBu for emulation work on this accelerated testbench creation with open source RTL.

Accelerating RISC-V testbench development with open source RTL and emulation

Varun Koyyalagunta, Mansoor Anees, Jiahan Zhang



tenstorrent

Introduction

- Tenstorrent seeks to develop a high performance RISC-V core and bring it market ASAP
- Emulation is key to that effort
  - Emulation is a must for software development and function verification
- How do we keep DV out of the critical path?
  - Tenstorrent is starting infrastructure, DV, and RTL from scratch
  - No prior in-house emulator setup

Agenda

- Typical testbench bringup cycle on emulation
- What can we leverage from previous work to accelerate bringup.
  - Common protocols (AXI, RVFI)
  - Open source code (RISC-V soft cores)
- Overview of testbench developed based on those components

Goal: Boot Linux on a Custom RISC-V Core

Typical Order of Events

Shift left verification using open source

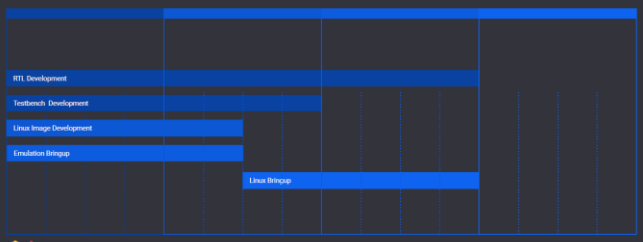
Accelerate testbench development

- RISC-V has a rich library of off-the-shelf free IP and VIP
- A testbench can reuse free VIP and be built around free IP
  - Allows the testbench and test development to be done in parallel with RTL development
  - A testbench with free IP can achieve goals like booting Linux without waiting on any custom RTL
- When the custom RTL IP is ready, it can be dropped into the existing testbench
  - Testbench functionality and bugs have already been flushed out on the free IP

tenstorrent

Shift left using Open Source RISC-V Core

Accelerate testbench development



tenstorrent

RISC-V Open Source IP and VIP

- IP
  - CVA6 core with AXI to memory
- VIP
  - Instruction Set Simulators
    - swerv-iss/whisper
  - CVA6 harness has RVFI for checking
- All the components needed to develop and test a RISC-V CPU testbench
- Custom RISC-V core can be dropped-in later

tenstorrent

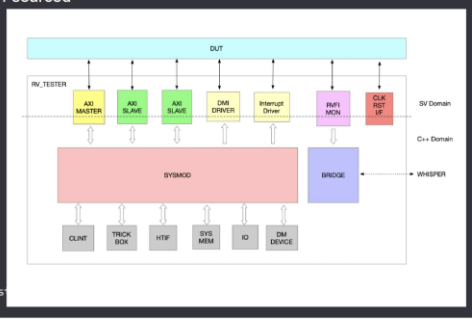
CVA6 RISC-V Open Source IP

- From the openhwgroup
- https://github.com/openhwgroup/cva6
- 6 stage in-order single-issue CPU
- Compiles into 67k LUTs: Fraction of 1 FPGA
- Enables fast compile iterations to debug testbench

tenstorrent

rv_tester Testbench

To be open-sourced



rv_tester Testbench

- Multi-threaded, asynchronous, message-passing based testbench built for configurability and maximum performance on emulation and simulation
- RVFI and AXI packets from CVA6 are transferred over DPI to C++ code
- These packets are asynchronously dispatched to components that subscribe to them
- Components (bridge, clint, etc) are configurable through a registry
  - Enabled or disabled based on DUT's configuration
- Test chaining on emulation is supported by reconstructing the registry between tests
  - "Warm reset" of the testbench
  - Saves ~2 minutes of overhead between tests

Connecting HW and SW in the Testbench

Example YAML code

```
m_rvfi:
    domain: 1
    num: ${max(TOP.PLATFORM.COSIM.RVFI.NRETS)}
    fields:
        trap:
            width: 1
        intr:
            width: 1
        cause:
            width: 64
        mode:
            width: 4
        rd_addr:
            width: 6
        rd_wdata:
            width: ${TOP.PLATFORM.XLEN}
```


Optimizing Transactions for Performance

Example C++ code that can subscribe to that transaction

- Called asynchronously on a separate thread, so that the emulator thread is not waiting

```
REGISTRY_register(rvfi, COSIM, cvm::registry::all);

rvfi::rvfi(cvm::topology::loc_t loc, unsigned id) {
    connect<
        rv_tester_transactions::cosim::m_rvfi<>
    >();


void rvfi::process(const rv_tester_transactions::cosim::m_rvfi<>& m_rvfi) {
    // do work
}
```

RISC-V Emulation Results

- Testbench iteration time drastically reduced
  - Emulator builds on the Open-SourceCVA6 are much faster than our custom core
  - ~1 hour vs ~12 hours
  - 67k LUTs vs 15M LUTs
  - Single issue in-order core vs 8 wide out-of-order custom RISC-V core
- Bugs found and fixed in emulation while custom core is brought up
  - Tools issues - SV constructs not fully supported in emulator
  - Testbench issues - various checker bugs missed in simulation
  - Verilog issues
    - 1 testbench tracing bug in cva6
    - 1 RTL bug in cva6
  - Linux image issues - throwing out unneeded features to speed-up boot
- Performance for custom RISC-V core on Linux boot: <5 minutes to boot
  - Full instruction checking: 990 kHz (working to improve)
  - No instruction checking: 1.5MHz (working to improve)

tenstorrent

Conclusion

- Accelerated testbench bring-up on Open-Source CVA6 prior to custom core
- 4 months of saved DV development time after RTL is ready!
- Testbench is ready to run billions of cycles every day to run regressions

tenstorrent

Vibarajan Viswanathan

Condor Computing

Sr.Principal DV Lead

Memory Patterns – Reusable Stimulus for RISC V Memory Subsystem Verification

User Paper

Abstract

Verifying Memory Subsystem in CPU is hard. The design blocks span from Load store to caches, NOCs and to the memory. In RISC, the max access size is 8 bytes at the Load Store Unit. The caches operate on whole 64bytes (32byte) cache line size. DV challenges span across a) Coherency: The verification of Hit or Miss, Snoops on L1, L2 and L3. b) Memory ordering between LD, ST c) Data Forwarding from ST to LD, d) Aligned vs unaligned transfers e) NOC Latency modelling f) Complex address patterns. This presentation proposes a reusable Memory Pattern generator that can stress the memory subsystem from Unit level to full core level.

Biography

Vibarajan (Viba) Viswanathan has over 25 Years of experience in Semiconductor and EDA industry, mostly in Design Verification. Viba holds a bachelor's degree in Electronics and Communication Engg., and a PG Diploma from UT Austin on AIML. His work experience spans across companies that include 0-In Design Automation, Synopsys, Mentor Graphics, Marvell Semiconductor, Samsung, Centaur and Microsoft. He was part of the pioneering team on Assertions and Assertion IPs, Formal Verification and Verification IPs. Viba's domain expertise includes RISCV CPU/GPU Memory Subsystem, Load Store, Cache Coherency, Memory Ordering, Shader Core, Coherent Interconnects, CHI, DDR/LPDDR Memory Controllers. Viba is an avid follower of conferences and the various Verification Methodologies that include Constrained Random, Formal Property Verification, Assembly level instruction generation and AIML DV Use cases.

Notes

CONDOR

Memory Patterns – Reusable Stimulus for RISC V Memory Subsystem Verification

Viba Viswanathan,
Sr. Principal DV Lead, Condor Computing
August 2024

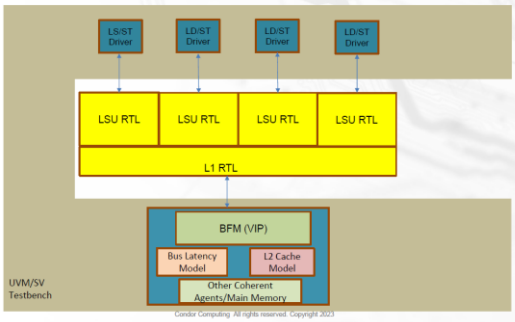
Acknowledgements

1) Kiran Puttaswamy, Sr. Principal DV Lead, Condor Computing
2) Doug Good, DV Director, Condor Computing
3) Ty Garibay, CEO, Condor Computing

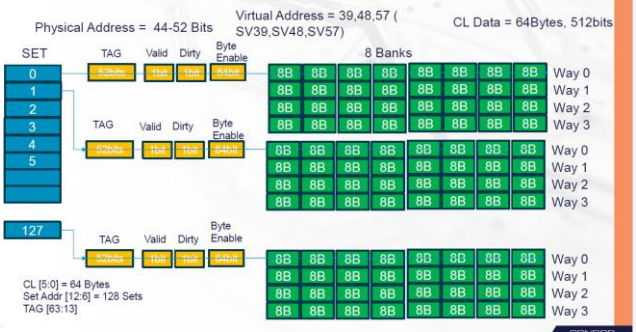
CONDOR

MSS Verification Scope

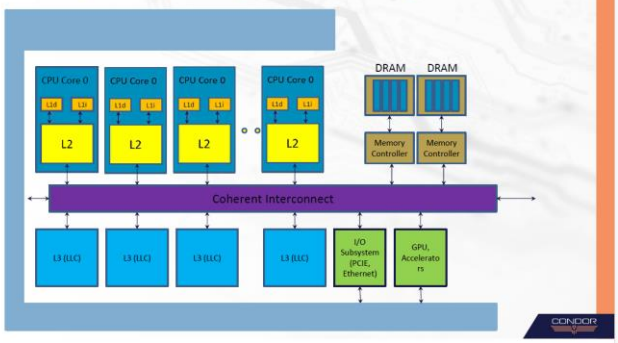
Load Store Unit Level (Single Core, Multi Slice)



L1 Cache



CPU Memory Subsystem (Full Core Level)



MSS Verification Scope

- Load Store
  - Loads, Stores, Atomics (Near/Far), IO Access
  - CL Aligned , Unaligned (Cross Cacheline)
  - Bank Aligned, Unaligned (Cross Data Bank)
  - MMU: Page Tables, TLBs, Exceptions, Hypervisor, Virtualization
  - Risc V Size : 2, 4, 8 Bytes (Max)
  - ROB Replay:
    - Speculative Execution (LD Pushed ahead when ST Address Unknown)
    - Branch Misprediction
  - Data Forwarding, Memory Ordering, Coherency
    - How will you handle when multiple STs, LDs to same cacheline outstanding?
  - OOO Responses from BUS Interface (DDR)
    - How to model OOO Responses
- L1
  - Cache Hit, Miss, Snoops, Evictions, LRU Schemes,
  - Hit: 2,4,8 Bytes
  - Miss: Full CacheLine (64Bytes)
  - Coherency
- L2/L3
  - Cache Hit, Miss, Snoops, Snoop Filter, Evictions, LRU Schemes
  - Access Size : Full CacheLine (64Bytes)
  - Coherency
  - Far atomics

Multicore Coherency

Two cores writing to same cache line

C0	C1
+1	+10

Incorrect Sequence
1
2
12
3
13

Correct Sequence:
1
2
12
13
23
33
34
35
36
37
38
39
40
41
51
52

Multiple modified copies

C0	C1
L1 (M)	ST to same CL (Snoop)
L2 (M)	
L3 (M)	

Which is most recent modified data?

Memory Ordering – Within a Core

- Same Cache Line Stores (ST0, ST0, ST0,..ST0)
  - Order needs to be maintained
- Same Cache Line Loads (LD0, LD0, LD0,..LD0)
  - Follow on Loads could reuse the return data of the first load
- Store to load forward : ST0,LD0
- Different Cache Line Address Loads/Stores: (ST0,LD1,ST2,ST3,LD4)
  - No ordering restrictions
- Same Cache line, Mixed Loads and Stores, Non-Overlapping bytes
  - No ordering restrictions
- Same Cache line, Mixed Loads and Stores, Same/Overlapping bytes (ST0,LD0,ST0,ST0,LD0)
  - All LDs, STs have to be in order
  - STs will be merged in order before LD data is written to Physical Register

Memory Ordering – Multicore

- Core0-ST0,Core1-Ld1, Core2-Ld2,Core3-Ld3 (Different Cacheline)
- At L3: LD1,LD2,ST0,LD3,LD4,LD5
- Intel (TSO): Total Store Order
  - LD1,LD2,ST0,LD3,LD4,LD5
  - LD2,LD1,ST0,LD4,LD3,LD5
  - LD1,LD2,ST0,LD3,LD5,LD4
  - LD2,LD1,ST0,LD5,LD4,LD3
  - .....
- ARM, RISC V: No Ordering
  - ST0,LD3,LD1,LD2
  - LD3,LD2,ST0,LD1
  - LD3,LD2,LD1,ST0
  - LD1,ST0,LD2,LD3
  - LD1,LD3,ST0,LD2
  - .....

CONDOR

Memory Patterns

Load / Store Instructions (RV64,I,M,A,C,F,D,ZICSR) – S?

- LB,LBU,LH,LHU,LW,LWU,LD,~~LDU~~ (RV128I inst)
- SB,SH,SW,SD
- Sizes
  - B – Byte (8 Bits)
  - H – Half Word (16 Bits)
  - W – Word (32 Bits)
  - D – Double Word (64 Bits)

Load Register/Address level Dependency

- Address dependency
- Sample Scenario

```
addi rd0, rs1, 0x5a;
sb r1, offset(rd0);
sb r2, offset(rd0);
....
....
addi rd1, rs1, 0xdead;
addi rd2, rs1, 0xdead;
sb r4, offset(rd1);
sb r5, offset(rd2);
```

- Register dependencies
- RAW, WAR, WAW
- Single

```
addi rd0, rs1, 0x5a;
lb r0, offset(rd0);
lb r1, offset(r0);
lb r2, offset(r1);
lb r3, offset(r2);
```

- Multiple

```
lb rb, offset(ra+8);
lb rd, offset(ra+64);
sh rd, offset(rb);
```

Same type Load, Continuous Address

Case 1

```
addi rd, rs1, 0x5a
//immediate value;
lb r1, 0(rd);
lb r1, 1(rd);
lb r1, 2(rd);
lb r1, 3(rd);
lb r1, 4(rd);
lb r1, 5(rd);
.....;
```

// Up to 5 or 10 Cacheline size and put them in loop

Case 2

```
addi rd, rs1, 0xa5
    //immediate value;
lb r1, offset(rd);
lb r1, offset(rd+1);
lb r1, offset(rd+2);
lb r1, offset(rd+3);
lb r1, offset(rd+4);
lb r1, offset(rd+5);
.....;
```

// Up to 5 or 10 Cacheline size and put them in loop

Case 3 (Decrement)

```
addi rd, rs1, 0xdeadbeef
  //immediate value;
lb r1, offset(rd);
lb r1, offset(rd-1);
lb r1, offset(rd-2);
lb r1, offset(rd-3);
lb r1, offset(rd-4);
lb r1, offset(rd-5);
.....;
```

// Up to 5 or 10 Cacheline size and put them in loop

Mixed Loads or Stores: Continuous Address

Mixed Loads

```
addi rd, rs1, imm value;
 lb r1, offset(rd);
 lh r1, offset(rd+1);
 lbu r1, offset(rd+1+2);
 lhu r1, offset(rd+1+3);
 lw r1, offset(rd+2+4);
 lw offset(rd+4+6);
 .....;
```

// Up to 5 or 10 Cacheline
//size and put them in loop

Mixed Stores

```
addi ra, rs1, imm value;
 sb rd, offset(ra);
 sh rd, offset(ra+1);
 sw rd, offset(ra+2+4);
 sw rd, offset(ra+4+6);
 .....;
```

// Up to 5 or 10 Cacheline size
// and put them in loop

Mixed Loads and Stores, Continuous Address

```
addi ra, rs1, imm address;
addi rd, rs1, imm data;
sb rd, offset(ra);
 lb rd, offset(ra);
sh rd, offset(ra+1);
lh rd, offset(ra+1);
sbu rd, offset(ra+1+2);
lh rd, offset(ra+1+2);
shu rd, offset(ra+1+3);
 lhu rd, offset(ra+1+3);
sw rd, offset(ra+2+4);
 lw rd, offset(ra+2+4);
sw rd, offset(ra+4+6);
 lw rd, offset(ra+4+6);
      .....;
```

// Up to 5 or 10 Cacheline size and put them in loop

Loads & Stores, Continuous Address, two level dependencies

Two Loads (Addr and Data) & One Store

```
addi ra0, rs1, imm addr;
addi rd0, rs1, imm data;
 lb ra, offset(ra0);
 lb rd, offset(rd0);
sb rd, offset(ra);
lb rb, offset(ra+1);
 lb rd, offset(ra+1);
sh rd, offset(rb);
 lb rb, offset(ra+1+2);
 lh rd, offset(ra+1+2);
sbu rd, offset(rb);
 lb rb, offset(ra+1+3);
 lh rd, offset(ra+1+3);
shu rd, offset(rb);
 lb rb, offset(ra+2+4);
 lh rd, offset(ra+2+4);
sw rd, offset(rb);
lb rb, offset(ra+4+6);
 lh rd, offset(ra+4+6);
sw rd, offset(ra+4+6);
.....;
// Up to 5 or 10 Cacheline size
and put them in loop
```

Base Patterns – Overlapping Addresses

```
addi rd, rs1, immediate value;
   lh r1, offset(rd);
   lh r1, offset(rd+2 - 2/2);
   lw r1, offset(rd+4 – 4/2);
   lh r1, offset(rd+8 – 2/2);
   lw r1, offset(rd+10 – 4/2);
   lw r1, offset(rd+14 – 4/2);
   .....;
// Up to 5 or 10 Cacheline size and put them in loop
```

Read / Write Collision : Bank Size is 8 Bytes.

- No overlapping bytes , Overlapping
  - Byte level overlapping
  - Word level overlapping
  - Dword level overlapping
- Same Cacheline different Bytes (No Collision)
- Same Cacheline same bytes (Collision)

CONDOR

Test Compose Strategy

Test Compose Strategy 1: "Base Patterns"

- Instruction Types:
  - Loads, Stores, Mixed Loads/Stores, Compressed Loads, Stores
    - LB,LBU,LH,LHU,LW,LWU,LD
    - SB,SH,SW,SD
- Patterns:
  - Address, Register Dependency
  - Single level, Multi level,
  - Increment, Decrement, Overlapping, Repeat (Same Address)

Test Compose Strategy 2 : "Address Pattern, Step Size"

- Address Patterns :
  - Random or Selective patterns
  - Selective Patterns
    - Increment, Decrement, Repeat
    - Continuous, Overlapping
  - Step Size
    - Same step size (Repeat / Same Address)
    - Variable Step Size
    - Fixed Step Sizes : Byte(8Bits), Half Word(16), Word(32), Dword(64)
    - **CL, SET, TAG Sizes**
  - Overlapping Variants:
    - Byte level Overlapping Addresses
    - Word level , dword level overlapping

Test Compose Strategy 3: "Test Options"

- Aligned/unaligned:
  - How to create unaligned : Using offsets or addresses.
- Test length (No of Transactions)
- One time run or Repeat Count

How to compose a test ?

- Final Test = Pick a "Base Pattern" + Pick a "Type of Address Pattern" + Pick a "Test Option"

CONDOR

Results/Conclusion

Conclusion

- Scalable Stimulus:
  - Generated Configurable, Reusable Assembly code
  - Same patterns could be generated for Unit level and Subsystem Level: LSU, L2, L3
- Results (LSU)
  - 200+ Generated Tests
  - 100+ Explicit tests
  - 60% Fails on Generated, 40% fails on Explicit tests

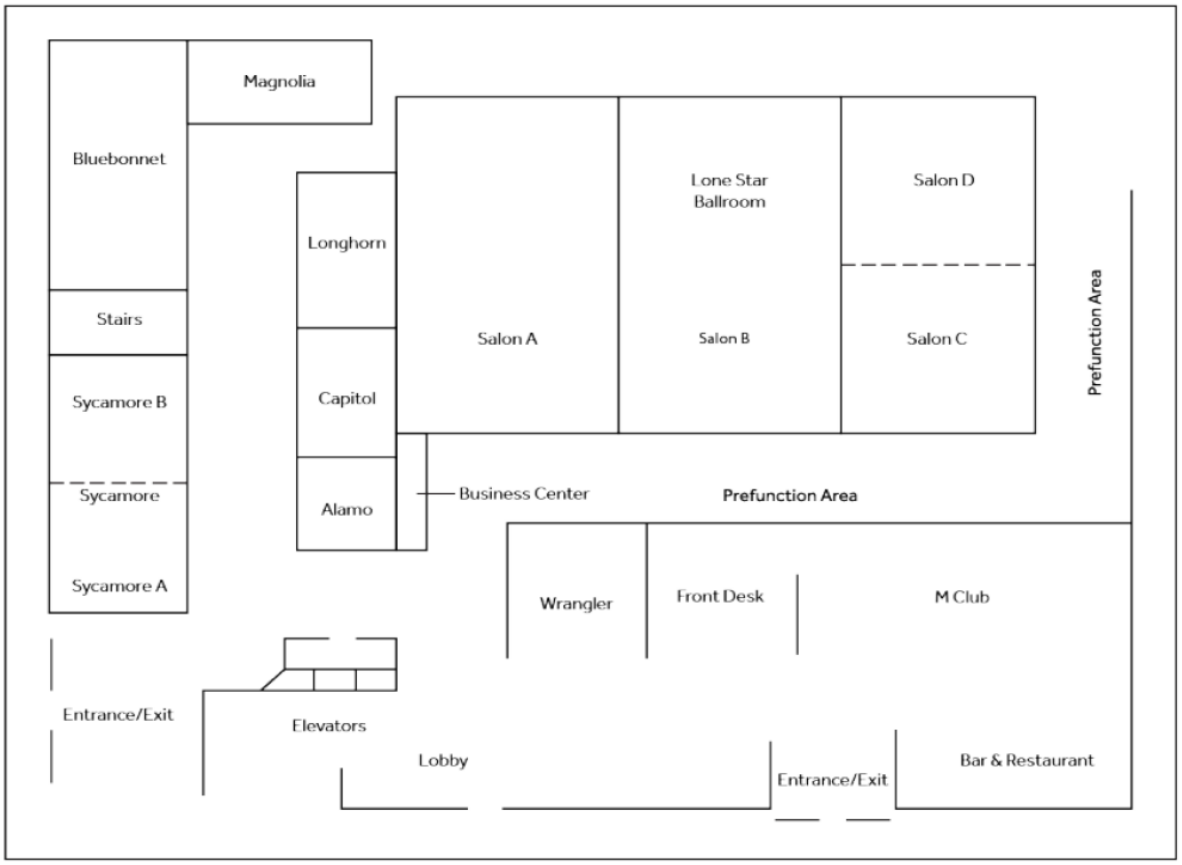
CONDOR

Notes

Track Session

Training Session 1
Signing Off with Formal
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

Signing Off with Formal

Gold Sponsor

Abstract

If you have only used formal apps or verified a few properties using modeling checking, then how do you sign-off on a project using formal just like a simulation-based verification environment? The answer is a good formal methodology and objective formal coverage. In this tutorial, we will have a look at both and discuss different types of formal coverage and how to interpret them.

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.

DOULOS KnowHow WEBINARS

Delivering KnowHow www.doulos.com

Signing-Off with Formal

Presenter: Doug Smith
Engineer / Instructor

Signing-Off with Formal

- Strategy and methodology
- Formal coverage
- Assertion density
- Reachability
- Mutation coverage
- Interpreting coverage
- Sign-off summary

Formal Approaches

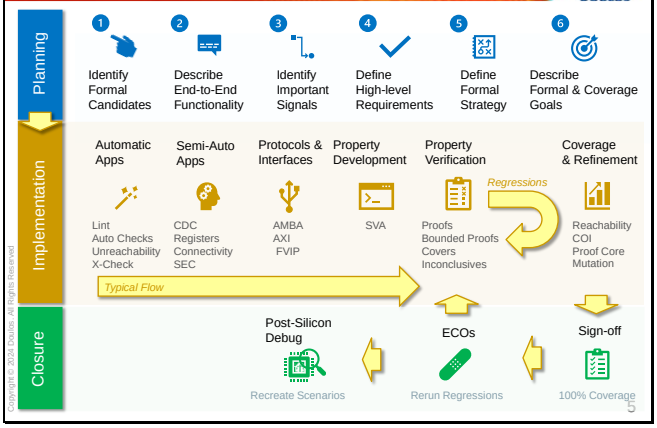
Bug Hunting	Sign-off
Formal apps	End-to-end checks
Formal VIP	Reduce complexity with abstraction
Property verification	Replace simulation with formal sign-off
Goal – CEXs	Find deep corner-cases
Coverage secondary	Full coverage

Sign-off Methodology

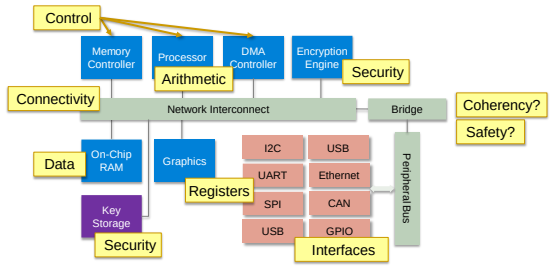
Plan	Implement	Closure
Project		
Resources	Allocate resources	Find computes Budget licenses
Block		
Identify candidates Divide and conquer	Verify block interfaces	Validate constraints
Functionality		
Describe functionality Prioritize goals	End-to-end checking	Prove and bug hunt
Complexity		
Complexity strategy Design reduction Case-split	Abstract	100% formal coverage

See paper, "Raising the level of Formal Signoff with End-to-End Checking Methodology" by Ping Yeung, et all, DVCon'22, San Jose, March 2022.

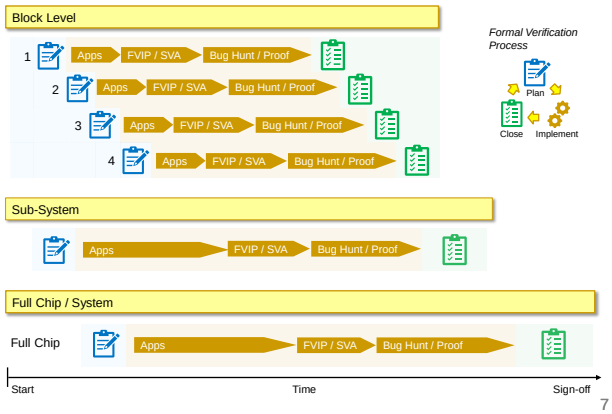
Formal Verification Process



Formal Candidates



Formal Throughout a Project



Signing-Off with Formal

- Strategy and methodology
- Formal coverage
- Assertion density
- Reachability
- Mutation coverage
- Interpreting coverage
- Sign-off summary

Formal Coverage Goals

Assertion precondition coverage – 100%

Functional coverage – 100% cover properties / covergroups

Assertion coverage – 100% at required proof depth

Code coverage – 100% line / condition / etc.

Types of Formal Coverage

- Assertion density (cone-of-influence)
- Assertion coverage (proof core)
- Functional coverage – cover properties / covergroups
- Unreachability
  - Identifies dead code
  - Reveals overconstraining
  - Generate coverage exclusions for simulation
- Reachability
  - Code coverage – statement / branch / FSM / toggle / etc.
  - Bounded reachability – proof bound / target for abstraction
- Mutation coverage – quality of assertions and constraints

Coverage – Known By Many Names

Aliases	Cone-of-Influence	Proof Core	Mutation	Reachability
Assertion or property density	X			
Stimulus coverage				X
Checker coverage	X	X	X	
Observability coverage		X		
Bounded coverage		X	X	X
Over-constraint coverage			X	X
Sign-off coverage	X (some tools)	X	X (some tools)	X

Signing-Off with Formal

Strategy and methodology

Formal coverage

➡ Assertion density

Reachability

Mutation coverage

Interpreting coverage

Sign-off summary

Assertion Density – Cone of Influence

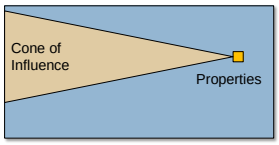
```
always @(posedge clock)
  if (reset)
    q <= 0;
  else if (ena)
    q <= a;
  else if (enb)
    q <= b;

a_enb: assert property (@(posedge clock) disable iff (reset)
                        ena |=> q == $past(a) ) ;

always @(posedge clock)
begin
  sig <= in1 & in2;
  out <= sig | in3;
end
```

Depends on

Does not require formal proof to run

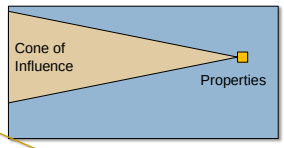
Assertion Density – All Assertions

Property Density (Structural)
Design: foo
Total Asserts - 1
Total Covers - 0
Total Registers - 3
Total Primary Inputs - 9
Total Primary Outputs - 2
Uncovered Registers (Asserts) - 2
Assertion Density (Registers) - 33.33%
Uncovered Primary Inputs (Asserts) - 3
Assertion Density (PI) - 66.67%
Uncovered Primary Outputs (Asserts) - 1
Assertion Density (PO) - 50.00%

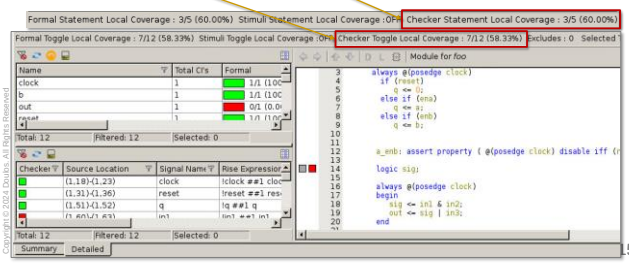


Registers – 33%
Primary Inputs – 66%
Primary Outputs – 50%

Assertion Density – By Coverage Items



Line Coverage 3/5
Toggle Coverage 7/12

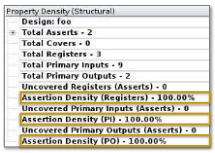


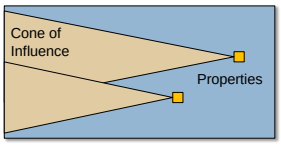
Add Another Assertion

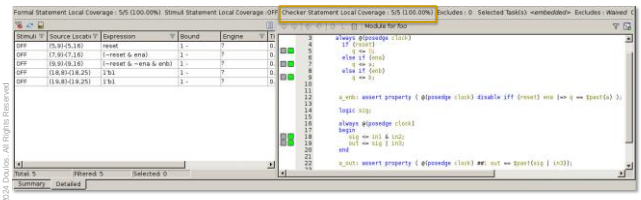
```
always @(posedge clock)
  if (reset)
    q <= 0;
  else if (ena)
    q <= a;
  else if (enb)
    q <= b;

a_enb: assert property (@(posedge clock) disable iff (reset)
                        ena |=> q == $past(a) ) ;

always @(posedge clock)
begin
  sig <= in1 & in2;
  out <= sig | in3;
end
```

```
a_out: assert property (@(posedge clock) ##1 out == $past(sig | in3) );
```

Revised Assertion Density Report







Everything is in the Cone of Influence of some assertion

Enough Assertions Really?

```
always @(posedge clock)
  if (reset)
    q <= 0;
  else if (ena)
    q <= a;
  else if (enb)
    q <= b;

a_enb: assert property (@(posedge clock) disable iff (reset)
                        ena |=> q == $past(a) ) ;

always @(posedge clock)
begin
  sig <= in1 & in2;
  out <= sig | in3;
end

a_out: assert property (@(posedge clock) ##1 out == $past(sig | in3) );
```

?

?

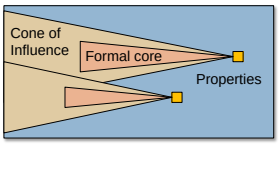
Instead, compute the formal core

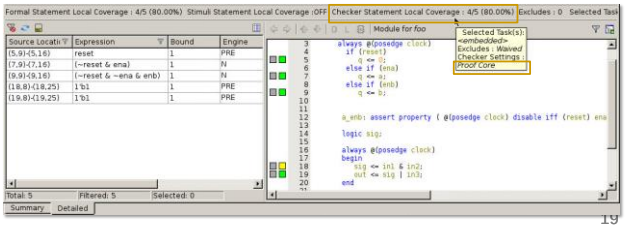
Formal Core

Run proof before computing formal core

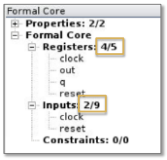
Computing formal core is CPU intensive

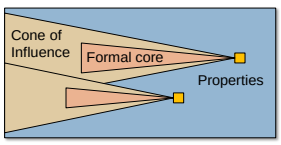
Works for proofs and bounded proofs





More Formal Core Results





Coverage	Total	Unobservable	Observable (Proofs)
Branch	4	0	4 (100.0%)
Condition	0	0	0 (N/A)
Expression	4	3	1 (25.0%)
FSM State	0	0	0 (N/A)
FSM Transition	0	0	0 (N/A)
Statement	5	1	4 (80.0%)
Toggle	24	6	18 (75.0%)
Covergroup Bin	0	0	0 (N/A)
Total	37	10	27 (73.0%)

Signing-Off with Formal

- Strategy and methodology
- Formal coverage
- Assertion density
- Reachability
- Mutation coverage
- Interpreting coverage
- Sign-off summary

Reachable Items

```
1   module simpleuart ( input clk, input resetn,
2                       output ser_tx, input  ser_rx,
3                       ... );
... ...
109 always @(posedge clk) begin
110   if (reg_div_we)
111     send_dummy <= 1;
112   send_divcnt <= send_divcnt + 1;
113   if (!resetn) begin
114         send_pattern <= ~0;
115         send_bitcnt <= 0;
116         send_divcnt <= 0;
117         send_dummy <= 1;
118   end else begin
119         if (send_dummy && !send_bitcnt) begin
120                 send_pattern <= ~0;
121                 send_bitcnt <= 15;
122                 send_divcnt <= 0;
123                 send_dummy <= 0;
124         end else ...
```

- Line or statement
- Branch
- Expression / Condition
- FSM States
- FSM Transitions
- Toggle

Conceptually ...

```
1   module simpleuart ( input clk, input resetn,
2                       output ser_tx, input  ser_rx,
3                       ... );
... ...
109 always @(posedge clk) begin
110   if (reg_div_we)
111     send_dummy <= 1;
112   send_divcnt <= send_divcnt + 1;
113   if (!resetn) begin
114         send_pattern <= ~0;
115         send_bitcnt <= 0;
116         send_divcnt <= 0;
117         send_dummy <= 1;
118   end else begin
119         if (send_dummy && !send_bitcnt) begin
120                 send_pattern <= ~0;
121                 send_bitcnt <= 15;
122                 send_divcnt <= 0;
123                 send_dummy <= 0;
124         end else ...
```

```
lineno = 110;
lineno = 111;
lineno = 112;
...
```

Line/statement reachable

```
line_cov_110: cover property ( lineno == 110 );
```


Unreachability

```
logic [2:0] state;

always @(posedge clock)
begin
  if (reset)
    state <= 0;
  else
    case (state)
    0: if (state == 0)
         state <= 1;
       else
         state <= 2;
    1: if (a && !prev_a)
         state <= 2;
       else if (prev_a && b)
         state <= 3;
    2: state <= 0;
    endcase
  prev_a <= a;
end

as_prev_a: assume property ( @(posedge clock) prev_a != b );
```

- state = 3 unreachable (FSM State Coverage)
- state 0->2, 1->0 unreachable (FSM Transition Coverage)
- state[2] 0->1, 1->0 unreachable (Toggle Coverage)
- Line unreachable (Line Coverage)
- Condition unreachable (Condition Coverage) w/ constraint
- Line unreachable (Line Coverage) w/ constraint

Use Models for Reachability Analysis

Run without constraints – identify dead code

Generate coverage waivers for simulation

Run with constraints – identify impact of constraints

```
else if (prev_a && b)
  state <= 3;

as_prev_a: assume property ( @(posedge clock) prev_a != b );
```

Line unreachable with constraint

Bounded coverage – identify areas not reached by bounded proof

Reachability Analysis

```
always @(posedge clock)
  if (reset)
    q <= 0;
  else if (ena)
    q <= a;
  else if (enb)
    q <= b;

a_enb: assert property (@(posedge clock) disable iff (reset)
                        ena |=> q == $past(a) ) ;

always @(posedge clock)
begin
  sig <= in1 & in2;
  out <= sig | in3;
end
```

Line Coverage 3/5
Toggle Coverage 7/12

But are there enough assertions?

Un/Reachability Coverage

Inputs

```
...
if (send_dummy && !send_bitcnt) begin
   send_pattern <= ~0;
   send_bitcnt <= 15;
   send_divcnt <= 0;
   send_dummy <= 0;
end else ...
```

Any items unreachable due to input stimulus?

Reveals over-constraining

Cover Type	Total	Unreachable	Reachable
Branch	35	35	35 (100.0%)
Condition	4	4	4 (100.0%)
Expression	2	2	2 (100.0%)
FSM State	0	0	0 (0.0%)
FSM Transition	0	0	0 (0.0%)
Statement	37	37	37 (100.0%)
Toggle	542	460	457 (84.9%)
Covergroup Bin	0	0	25 (0.0%)

Also known as: *Stimulus coverage* *Over-constraint coverage*

Coverage – The Story So Far ...

Simulation coverage – measure the quality of the simulation vectors [Test bench]

Cover property – prove that a specific property holds at least once [RTL/constraints]

Cone of Influence – measure assertion density (quick-and-dirty) [Assert/cover]

Formal core – measure property density (more realistic) [Assert/cover]

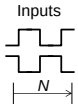
Un/Reachability analysis (unconstrained) – identify dead code [RTL]

Un/Reachability analysis (constrained) – identify over-constraint [Constraints]

Bounded unreachability analysis .....

Bounded Un/Reachability Coverage

What can be reached within a bounded analysis?




```
...
if (send_dummy && !send_bitcnt) begin
   send_pattern <= ~0;
   send_bitcnt <= 15;
   send_divcnt <= 0;
   send_dummy <= 0;
end else ...
```

Line#	Coverage Type	Result	Details
123	Statement	W 1	send_dummy <= 0;
125	Branch	? 28	(#1) if (reg_dat_we && !send_bitcnt) begin
129	Statement	? 28	send_pattern <= {1'b1, reg_dat_di[7:0], 1'b0};
134	Statement	? 28	send_bitcnt <= 10;
136	Statement	? 28	send_divcnt <= 0;
138	Condition	W 3	(#1 - ((((send_divcnt > cfg_divider)) && (	...
138	Branch	W 3	(#1) if (send_divcnt > cfg_divider && send_bitcnt) begin
139	Statement	W 3	send_pattern <= {1'b1, send_pattern[9:1]};
140	Statement	W 3	send_bitcnt <= send_bitcnt - 1;
141	Statement	W 3	send_divcnt <= 0;

Depth

Signing-Off with Formal

Strategy and methodology

Formal coverage

Assertion density

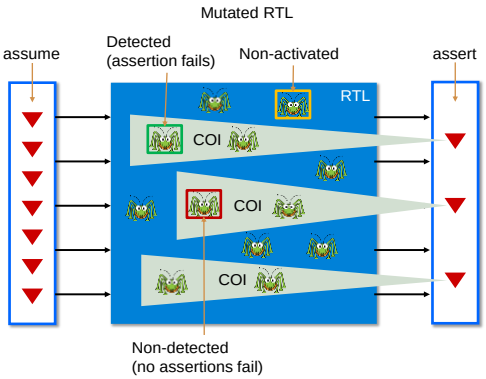
Reachability

Mutation coverage

Interpreting coverage

Sign-off summary

RTL Fault Injection



RTL Mutations

Inject RTL faults by:

Breaking connectivity (creates a free variable)

Forcing variable to a constant (stuck-at 0/1)

Forcing variable to inverted value (stuck-at negative)

Forcing execution of a conditional branch (stuck-at 0/1)

Changing OR to AND (functional fault)

Swapping operands in an expression (functional fault)

Running Mutation Coverage

1. First achieve 100% assertion, COI, and reachability coverage
2. Run unmutated RTL with formal testbench as sanity check
3. Run mutation coverage (exclude faults detected by simulation)

Fault outside COI of any assert => write more assertions or waive

Fault undetected => write more assertions or waive

Fault detected by causing assertion to fail => okay

Signing-Off with Formal

Strategy and methodology

Formal coverage

Assertion density

Reachability

Mutation coverage

Interpreting coverage

Sign-off summary

Interpreting Coverage – Property Coverage

Cover Properties

Assert Properties	Failing	Passing
Failing	Covers - overconstrained Asserts - underconstrained	Underconstrained Bugs
Passing	Overconstrained Missing asserts	Goal! ✓

Interpreting Coverage – Stimulus vs Checker

Un/Reachability (Stimulus Coverage)

COI / Proof Core (Checker Coverage)	Low	High
Low	Need more assertions Overconstrained Dead code	Need more assertions
High	Overconstrained	Goal! ✓

Signing-Off with Formal

Strategy and methodology

Formal coverage

Assertion density

Reachability

Mutation coverage

Interpreting coverage

Sign-off summary

Formal Sign-off Process in Review

Plan!

- Identify formal candidates
- Capture requirements

Implement

- Apps -> Interfaces -> SVA -> Formal Check -> Bug Hunt -> Regressions

Close

- 100% properties complete (or bounded)
- 100% functional coverage (covers and covergroups)
- 100% assertion density
- 100% reachability (with review)
- 100% code coverage (COI/proof core) (with review)
- Measure checker quality with mutation coverage

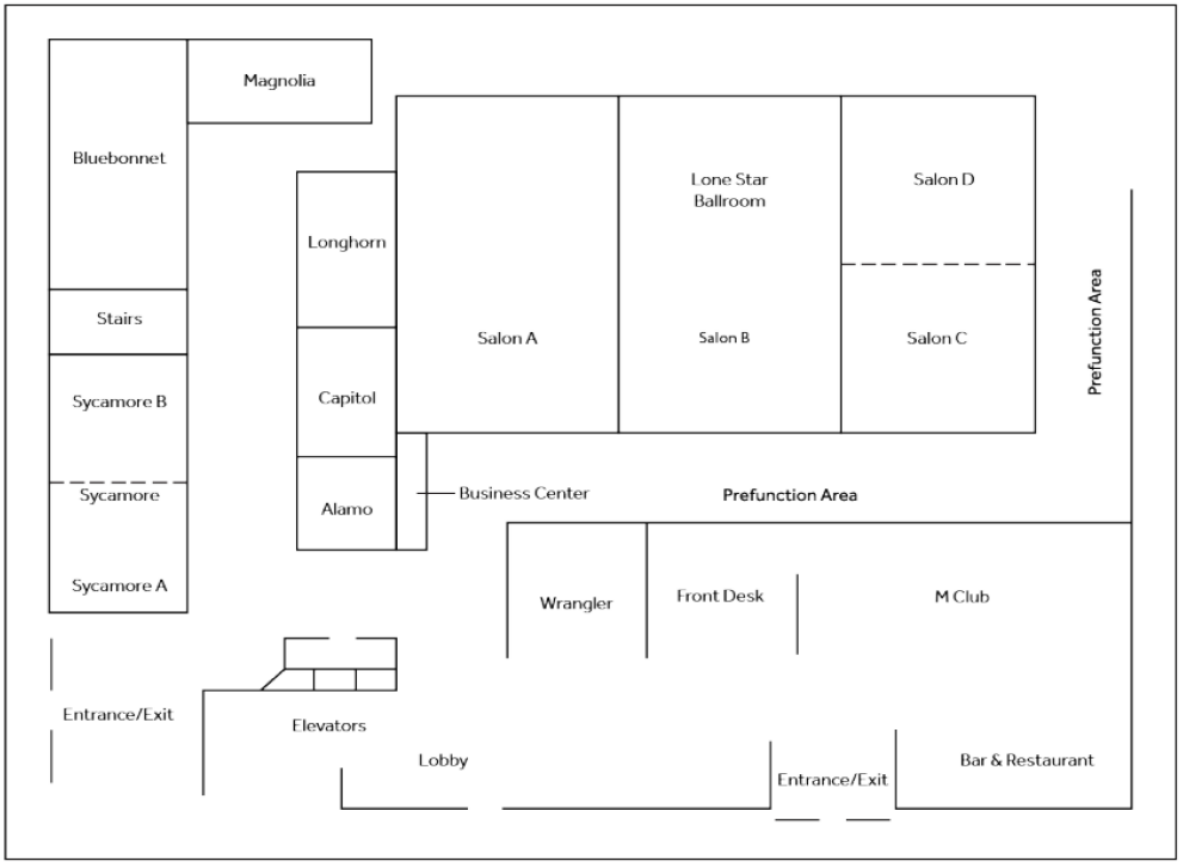
Q & A

Any questions?

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

Senior Member of Technical Staff

Mixed-Signal Randomisation - Stimulus and Checkers

User Paper (Remote presentation)

Abstract

Constrained-random verification (CRV) is a well-established and successful methodology for digital designs and UVM has become the primary means to achieve this.

The functional complexity in mixed-signal designs is increasing exponentially. Even a simple Power Management Unit can have hundreds or thousands of control bits and many modes of operation. To achieve good functional coverage, it is important to adopt a CRV approach for mixed-signal designs. This approach can help us manage a large verification space, including: checking analogue performance under a large set of programmable configurations; digital control system interaction with analogue circuits; covering unexpected corner cases in A/D interaction. To lower the bar for verification engineers to switch Device-Under-Test (DUT) abstractions between DMS and AMS a generic monitor will be presented. This monitor is agnostic to the DUT abstraction so the Verification engineer can work in the environment they are most comfortable in.

This presentation shows some of the ways in which Renesas has applied UVM to mixed-signal designs and some of the benefits we gained by doing so. The work presented was the driving force in defining the UVM-MS standard.

Biography

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'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 many 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.

Steve has 24 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. 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

MIXED-SIGNAL RANDOMISATION STIMULUS AND CHECKERS

METRIC DRIVEN MIXED SIGNAL VERIFICATION (DMS/AMS)

STEVE HOLLOWAY, RENESAS, SEPTEMBER 2024
PETER GROVE, RENESAS, SEPTEMBER 2024

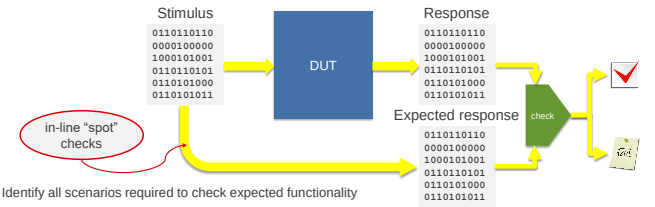
KEYWORDS: AMS, DMS, DV, VERIFICATION, UVM, UVM-MS

AGENDA

- Directed and Constrained Random Verification
- A Verification Example
- A Metric Driven Mindset
- Generic V / I Monitor for Mixed Signal Designs
- UVM for Mixed Signal (UVM-MS)
- Conclusion

PG1

DIRECTED TESTING



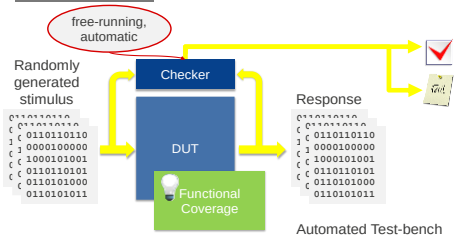
Identify all scenarios required to check expected functionality

Create (large) suite of procedural tests

Completeness from pass/fail status and code coverage PG0

By definition *unexpected* scenarios are **not** covered – a large hole in coverage

CONSTRAINED RANDOM TESTING



Constrained-random stimulus vectors

Automatically checked DUT response

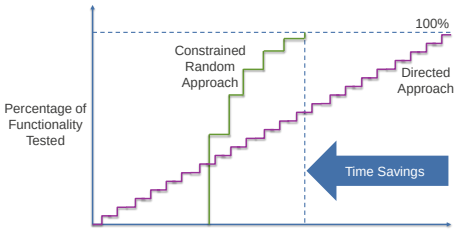
Functional Coverage is the measure of verification completeness

Stimulus, Checking, Coverage are *separate* PG0

UVM is not a prerequisite

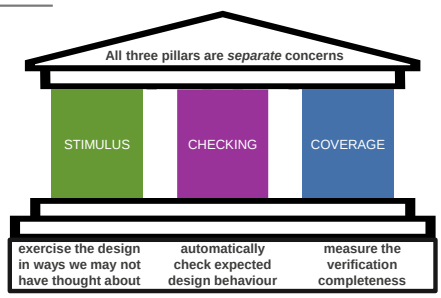
Automated Test-bench

DIRECTED AND CONSTRAINED RANDOM VERIFICATION



Source: SystemVerilog reference verification methodology

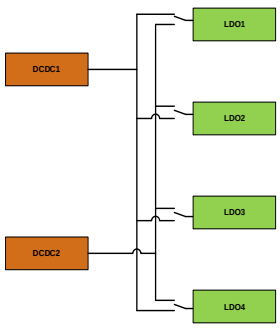
THREE PILLARS OF METRIC-DRIVEN VERIFICATION



A VERIFICATION EXAMPLE

MIXED SIGNAL EXAMPLE

- Our example system has 2 x DCDC and 4 x LDO's
  - LDO range 0.9V…2.5V in steps of 50mV. (32 steps)
  - DCDC range 1.2V…4.4V in steps of 100mV. (32 steps)
- All DCDC's and LDO's will be started during boot process.
  - Any DCDC can supply 1 or more LDO's.
  - DCDC will always be at least 200mV above the highest LDO VSEL.
  - Startup order requires DCDC to be started before LDO's it supplies (16 Slots).
  - LDO Voltage could be anything within their range.



VERIFICATION SPACE

Block	Configuration Bits
4 x LDO	24
2 x DCDC	10
Slot Sequencer	48
Total	**82**

This represents 2^{82} combinations

Even checking each setting at 100MHz takes 1.5×10^{9} years !!!

We need to forget about exhaustive (directed) testing

CONSTRAINED RANDOM

Use a "spray and pray" approach to bug finding and hope you get lucky:

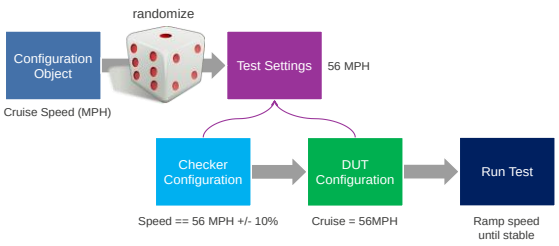
Use CPU horsepower to generate random test scenarios

Guide the automation based on your knowledge of the design to target areas of risk

Measure the scenarios that are hit by creating a functional coverage model

PG0

TEST RANDOMIZATION FLOW



PG0

MODELLING RAIL CONFIGURATION

```
class rail_cfg extends uvm_object;
...
       int unsigned id;      // rail id
  rand int unsigned supply; // rail supplying this one
  rand int unsigned vsel;   // voltage selection
  rand int unsigned slot;   // slot selection

  constraint c_vsel_range { vsel inside {[0:31]}; }
  constraint c_slot_range { slot inside {[0:15]}; }
...
endclass: rail_cfg
```

Composition

always create right array size

```
class sys_cfg extends uvm_object;
...
  rand int unsigned num_rails; // rails in the system
  rand rail_cfg     rails[];   // array of rail objects
  constraint c_rail_array_size { rails.size() == num_rails; }
...
endclass: sys_cfg
```

HOW TO GUIDE RANDOMISATION?

```
[SYS] DCDC 1 Vsel = 2.40v Slot = 6
[SYS] DCDC 2 Vsel = 2.60v Slot = 3
[SYS] Rail 1 Supply DCDC2 Vsel = 0.95v Headroom = 1.65v Slot = 15
[SYS] Rail 2 Supply DCDC1 Vsel = 1.45v Headroom = 0.95v Slot = 7
[SYS] Rail 3 Supply DCDC2 Vsel = 1.65v Headroom = 0.95v Slot = 6
[SYS] Rail 4 Supply DCDC2 Vsel = 1.15v Headroom = 1.45v Slot = 10
```

```
function void post_randomize;
    foreach (rails[i]) begin
      rails[i] = new();
      rails[i].id = i;
      assert (rails[i].randomize() with {
        // Supply must be set before solving vsel
        solve rails[i].supply before rails[i].vsel;
        // Constraints for LDO rails[2:] - supply from DCDC[1:2]
        rails[i].id > 1 -> rails[i].supply inside {[0:1]};
        // vsel < supply.vsel + 2 (supply-200mV)
        rails[i].id > 1 -> rails[i].vsel < rails[rails[i].supply].vsel + 2;
        // schedule after slot from which it is supplied
        rails[i].id > 1 -> rails[i].slot > rails[rails[i].supply].slot; });
    end
  endfunction
```


WHAT ABOUT REAL NUMBERS?

IEEE 1800-2023:

"***The solver can randomize singular variables of any integral or real type***"

Random real values are uniformly distributed over their range. For example:

```
rand real v;
constraint c {v > 0.0 && v < 2.0;}
```

The probability of choosing a value in the range 0.0 to 1.0 is the same as the probability of choosing a real value in the range 1.0 to 2.0.

A real is a 64-bit number so some *sensible clamping* on significant figure is needed

```
(int' (real_var/1e-6)*1e-6) //Clamp to 1u significant figure
```


PG0

REAL CONSTRAINTS

```
class real_config_c;
  const int  ZSTATE    = -100;
  const real VALUE_LOW = 0.70;
  const real VALUE_MIN = 1.43;
  const real VALUE_NOM = 3.30;
  const real VALUE_MAX = 3.65;
  rand real a;
  rand real b;
  . . .
endclass
```

```
constraint a_constraint {
  a dist { ZSTATE                          := 5,
           [VALUE_LOW:VALUE_MIN]           :/ 1,
           [VALUE_NOM-0.03:VALUE_NOM+0.03]:/ 13,
           [VALUE_MIN:VALUE_MAX]           :/ 1
         };
}
```

range — weight

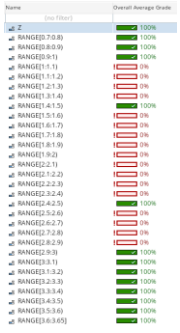
```
constraint b_constraint {
   (a inside [VALUE_LOW:VALUE_MIN])-> b == ZSTATE;
    b dist { ZSTATE                 := 1,
             [VALUE_MIN:VALUE_MAX] :/ 20
           };
    solve a before b;
}
```


PG0

REAL COVERAGE

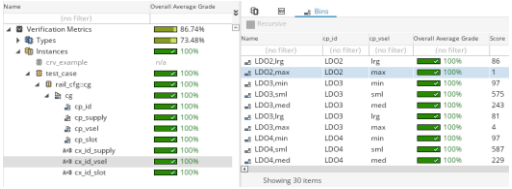
- A range of real values is different from a range of integral values.
- It allows all possible values within the specified range
  - e.g. **VALUE_LOW > a > VALUE_MAX**
- The covergroup option ***real_interval*** defines partitioning for a range of bins
- A single-value real bin can fail to cover numbers very close to the bin value, because a real number representation has limited precision

```
covergroup cg;
  type_option.real_interval = 0.1;
  cp_a:coverpoint a {
    bins Z = {real'(ZSTATE)};
    bins RANGE[] = {[VALUE_LOW:VALUE_MAX]};
  }
endgroup
```



VERIFICATION METRICS

- Constrained-random can cover more verification space by running many seeds
- Create a metrics model based on spec features, design knowledge, etc.
- Pairwise crosses between features are a useful strategy



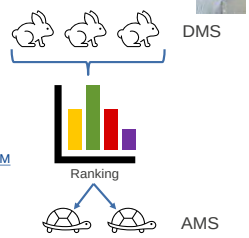
BUT AMS SIMULATIONS CAN BE VERY SLOW!

Run regression suite with **DMS** models and checkers

Rank tests by contribution to functional coverage

Run the best seeds on AMS

Works well with a ***Unifed TestBench for DMS/AMS***!
https://www.youtube.com/watch?v=IwnRXwR4whM



GENERIC CONCURRENT V / I MONITOR FOR MIXED SIGNAL DESIGNS (AMS OR DMS)

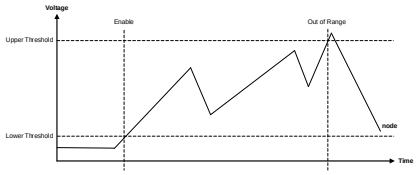
PROBLEM STATEMENT AND REQUIREMENTS

- **Ability to monitor internal nodes** in the design which based on netlist configuration (HED) could change abstraction.
  - In one HED configuration the node resolves to a Real, in another UDN and in another Electrical.
  - Node may not exist in some HED configurations!
  - Effects of node can't easily be monitored on DUT IO's. (E.g. Internal bias currents/references)
  - Monitor Voltage, Port Current or Port/Block Power.

- **Requirements (Wish list)**
  - No elaboration error if the path does not exist. Use a string as a path to a scalar node/port.
  - Work's in DMS/AMS without user interaction.
  - Ability to change trigger levels during the simulation.
  - Dynamically enable/disable the monitor during the simulation.
  - Limit simulation overhead.
  - Could be used for UPF supply monitoring thus works in DMS/AMS without user interaction or vendor extensions.

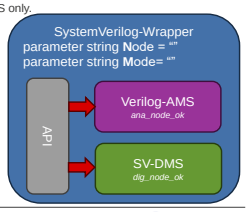
WHY AND WHAT TO MONITOR

- Automatically enable monitor on internal signals. E.g. Bandgap enable. (Allows concurrent checking for any simulation.)
  - Noise on some nodes are hard to see unless a catastrophic issue happens, resulting in a *verification hole*.
- Detect if a circuit kicks a reference when starting up. E.g. Comparator's capacitive feedback kicks high impedance node.
- Monitor static bias currents to blocks. If a bias goes to 2 blocks by accident it might not be easy to detect in AMS.
- In DMS most of this is not as important, but DMS can be used to setup/debug the monitor in a fraction of the time.



OUTLINE OF MODULE(S) AND FLOW

- Must be a ***module*** to access continuous time nodes at every analog timestep.
- Utilized simulator initialization sequencing, $analog_node/port_alias and VPI routines.
  - Parameters to set **N**ode, **M**ode and whether DMS+AMS, DMS only or AMS only.
- Hierarchy of 3 modules, with multiple views of Verilog-AMS version.
  - Dummy module for DMS netlist (ana_node_ok = 1'b0)
  - Verilog-AMS for AMS simulations.
- Matches UVM-MS proposed standard architecture
  - SV Bridge (SV wrapper), instancing API via proxy.
  - DUT abstraction for 'analog resource(s)'



SEQUENCE – ALL BEFORE DC OP EVEN ATTEMPTED.

1. Analog initial block(s) executed and checks if node is electrical. ana_node_ok variable set accordingly.
2. Control passed to digital engine what starts processes, triggering initial block(s);
   I. Call's function get_ana_node_ok(), which uses up name referencing to call function in wrapper layer that calls the function in the VAMS module to get the analog owned variable. Creates local copy in wrapper cell.
   II. If the return value is 0 then node is not electrical or does not exist. Try DMS using VPI to register Call back
   III. If dig_node_ok is 0 then node does not exist. Depending on AMS only, DMS only or AMS+DMS error.
3. Transient simulation carries on and API can be used to change enable and the thresholds.

ANALOG OOMR PROBING

- Verilog-AMS has $analog_node_alias and $analog_port_alias functions.
- ana_node_ok is a continuous variable, so a discrete function is required to get value from wrapper SV layer.

```
electrical int_node;

function integer get_ana_node_ok(input dummy);
  begin
    get_ana_node_ok = ana_node_ok;
  end
endfunction
```

Discrete function for readback for ana_node_ok

```
analog initial begin
  ana_node_ok = 0;
  if(mode == "potential") ana_node_ok = $analog_node_alias(int_node,node);
  if(mode == "flow")      ana_node_ok = $analog_port_alias(int_node,node);
end
```

Hierarchical connection based on string node

```
analog begin
   value = ana_node_ok ? ((mode == "potential") ? V(int_node):I(<int_node>)):0.0;
  //Monitor code
end
```

Probe and monitor check.

DIGITAL OOMR PROBING

- SystemVerilog has no $digital_node_alias/$digital_port_alias function. (Vendors don't have a generic version for any netType.)
- Direct assign via OOMR's would force Connect Module in AMS or a netType conflict in pure SystemVerilog.
- No native port current probing with SystemVerilog UDN's. Knowledge of UDR required.
  - "The Importance of Verilog Procedural Interface to Utilize a Single Environment/Testcase for DMS/AMS (Peter Grove" CDNLive 2023 Europe track winner)
- Solution query the simulator at runtime using the Verilog Programming Interface (VPI) to access the data objects
  - Make the VPI code unbound to a specific net/variable structure and expandable for other datatypes in the future

- Use a function in SV to call some VPI (*Verification Procedural Interface*) code.
- VPI code queries the simulator to get the information.
- Can then handle many different use cases but keep SV function the same.
- All LRM compliant.

VPI – SET UDN CHANGE CALLBACK

- Imported custom C function dpi_udn_change_callback() to SV as setUDNChangeCallback() using DPI.
  - SV function registers a VPI call back when specific object change value.

```
import "DPI-C" context dpi_udn_change_callback = function int dpi_udn_change_callback( input  string udn_net,
                                                                                        output string error_msg,
                                                                                        input  string scope);

//Users function call to handle any error condition
function automatic bit setUDNChangeCallback(input string udn_net, input string error_level = "fatal", input string scope="");
   string error_msg;
   if(!dpi_udn_change_callback(.udn_net(udn_net), .error_msg(error_msg), .scope(scope), .debug(`DLGMS__VPI_DEBUG))) begin
      $display("Error : %s", error_msg);
      return 0;
   end
   return 1;
endfunction
```

Imports C function to SV

- `udn_net` – Input string to the net to be monitored. (In this case the VPI code only support UDN, but could be extended.)
- `error_msg` – Output string variable to return any error messages from the VPI code.
- `scope` – Input string that should be set to `$sformatf("%m")`. Can't be default value as function could be in Package. And used to set the scope for the call back task.
- `setUDNChangeCallback` returns 0 or 1 to indicate a call-back successful or not. Error messages automatically displayed.

VPI – SET UDN CHANGE CALLBACK

- Custom C function registers a callback (*vpi_register_cb*) when an objects value changes.
  - Monitor V field of UDN for Voltage probe, and additionally R field for Current probe, based on specific UDN.
  - Callback C function triggered when V or R field changes value, this could be during delta cycles.
  - Callback C function calls the SV task function that has been imported to the C layer, using svScope the dpi_udn_change_callback() was called from.

```
net_h = vpi_handle_by_name(path, 0);
memI  = vpi_iterate(vpiMember, net_h);
while ((memH = vpi_scan(memI))) {
 /* search through the member names till we find the voltage */
 char *memName = vpi_get_str(vpiName,memH);
 if (0==strcmp(memName,"V")) {
   time_s.type       = vpiSuppressTime;
   value_s.format    = vpiStrengthVal;
   cb_data_s.reason  = cbValueChange;
   cb_data_s.cb_rtn  = dpi_udn_change_callback_fn;
   cb_data_s.time    = &time_s;
   cb_data_s.value   = &value_s;
   cb_data_s.user_data = malloc(strlen(svGetNameFromScope(svGetScopeFromName(scope)))+1);
   strcpy(cb_data_s.user_data, scope);
   vpi_register_cb(&cb_data_s);
} ...
```

Handle to UDN and iterator over struct.

```
typedef struct t_cb_data {
 PLI_INT32 reason;                    /* callback reason */
 PLI_INT32 (*cb_rtn)(struct t_cb_data *); /* call routine */
 vpiHandle obj;                       /* trigger object */
 p_vpi_time time;                     /* callback time */
 p_vpi_value value;                   /* trigger object value */
 PLI_INT32 index;                     /* index of the memory word or var
                                         select that changed */
 PLI_BYTE8 *user_data;
} s_cb_data, *p_cb_data;
```

Build Call-Back struct and register it with the simulator.

C-Code

VPI – SET UDN CHANGE CALLBACK

- Exported SV task update_udn_event() to C.
  - SV task uses non-blocking event trigger ->>, so multiple changes on the UDN only call the event once.

```
void dpi_update_udn();
static PLI_INT32 dpi_udn_change_callback_fn(p_cb_data cb_data_p) {
 svScope scope = svGetScopeFromName(cb_data_p->user_data);
 svSetScope(scope);
 dpi_update_udn();
 return 1;
}
```

C function called by registration to vpi_register_cb

Sets scope to call the imported task obtained from user_data field set at callback generation..

C->SV link

```
export "DPI-C" dpi_update_udn = task update_udn_event;

event node_changed;

task update_udn_event(); if(en && dig_node_ok) ->>node_changed; endtask

always@(node_changed) begin
  //Get value based on mode using VPI
end
```

Non-blocking Event

AMS MONITORING CODE

- Use an integer variable to toggle 0/1. (`above` or `cross` not used, use `last_crossing` to get interpolated crossing time
  - Hysteresis is important to avoid oscillation due to tolerances on the Voltage/Current been monitored
- Use `absdelta`() function to sample into discrete domain for wrapper SV to probe via OOMR.
  - Set delta_expr and time_tol to 0 so a change in value will be picked up unless it toggles faster than time precision.

```
analog begin
   //Gate process based on ana_node_ok to save CPU cycles.
   if(ana_node_ok && en_a) begin
      value = (mode == "potential") ? V(int_node):I(<int_node>);
            if (value > (upper_a + (in_range_int_a ? hyst:-hyst))) in_range_int_a = 0; //Not OK
      else if (value < (lower_a + (in_range_int_a ? -hyst:hyst))) in_range_int_a = 0; //Not OK
      else                                                       in_range_int_a = 1; //OK
   end
   else begin
      in_range_int_a = 0; //Not OK
   end
end

//Sample back onto discrete domain.
always@(absdelta(in_range_int_a, 1, 0, 0, 1)) in_range_int = in_range_int_a;
```


DMS MONITORING CODE

- Call back will use non-block event on `node_changed`, so it triggers in non-block region of simulation cycle.
- Use VPI routines to get the Voltage and Current but only if enabled as this can be costly.
- Additionally need to check for `wrealXState/`wrealZstate that could be a return value. (Not in LRM but all vendor support)

```
always@( node_changed or en or posedge dig_node_ok) begin
   if(en && dig_node_ok) begin
      //Get value which should not fail if node is valid
      if(mode == "potential" ) void'(getVoltageFromPath(.path(node), .voltage(value), .error_level("fatal")));
      else                     void'(getCurrentFromPort(.path(node), .current(value), .error_level("fatal")));

      //Range checker
           if (isZX(value)  ) in_range_int = 1'b0; //Not OK
      else if (value > upper) in_range_int = 1'b0; //Not OK
      else if (value < lower) in_range_int = 1'b0; //Not OK
      else                    in_range_int = 1'b1; //OK
   end
   else                       in_range_int = 1'b0; //Not OK
end
```


MONITOR REPORTING

- SV wrapper use ana_node_ok and dig_node_ok to monitor the right module.
  - SV task uses non-blocking event trigger ->>, so multiple changes on the UDN only call the event once.

```
   // Monitor the submodule.
   always begin
     if(ana_node_ok) @(u_ams_monitor_core.in_range_int iff u_ams_monitor_core.en) begin
       if(!u_ams_monitor_core.in_range_int) `uvm_error("Node Monitor", ... ))
       else `uvm_info ("Node Monitor", ... ), UVM_MEDIUM)
     end
     else if(u_dms_monitor_core.dig_node_ok) @(u_dms_monitor_core.in_range_int iff u_dms_monitor_core.en)
   begin
       if(!u_dms_monitor_core.in_range_int) `uvm_error("Node Monitor", ... ))
       else `uvm_info ("Node Monitor", ... ), UVM_MEDIUM)
     end
     else @(ana_node_ok or u_dms_monitor_core.dig_node_ok);
   end
```


VERIFICATION METHODOLOGY

METRIC DRIVEN MINDSET

Find bugs which almost certainly exist in the design
- Stress protocols – don't just interoperate with them
- Explore corner cases
- Zero tolerance of design inconsistency

Reuse and debug should be considered from the start
- Using a methodology in the *right* way addresses reuse
- Appropriate and consistent messaging scheme

The right use of metrics
- Test-case pass rate is a "design" mindset
- Coverage closure is the objective, test cases are a means to get there
- *Never* collect coverage without checks

PG0

UVM-MS

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

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

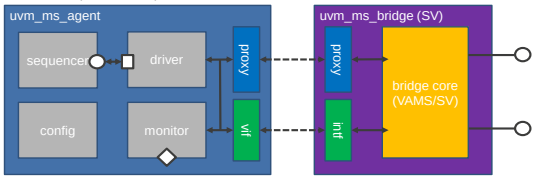
Named UVM-MS as the focus is to support any MS system; DMS, RNM, Spice or a mixture.

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 STATUS

Public Draft Released 9th August 2023

Peter and Steve developed the concept



- UVM -> UVM-MS differences
  - The uvm_ms_agent send parameter via the proxy to the bridge core. E.g. Phase, Freq for sine wave
  - Bridge Core generates the signal.

CONCLUSION

CONCLUSION

- Constrained-Random Verification is applicable for Mixed Signal designs, if not essential.
- More effort is required to assemble a Metric-Driven platform
  - However, re-use can pay off here
- Fast DMS regression can be used to select seeds for slower AMS tests
- Most successful if a Metric Driven Mindset is adopted from the vPlan onwards

- The VPI enables a sophisticated concurrent checking system for ‘analog’ intent signals whether in DMS or AMS.
  - Verification engineers does not need to know the DUT’s abstraction.

- UVM-MS will drive this capability.

Renesas.com

Adnan Hamid

Breker Verification Systems

Founder and CTO

RISC-V Certification: Applying Advanced RISC-V Core and SoC Verification Towards the Anticipated Certification Requirements

Gold Sponsor

Abstract

RISC-V processor cores are becoming more complex and varied, driving an ever greater need to prove their quality through advanced verification. RISC-V International has instigated a new Certification Group to target a quality metric, and this requires a range of new, automated verification test solutions. Through his work with multiple RISC-V core providers to extend their verification environments to meet modern processor verification requirements, the author has a unique perspective towards meeting these certification requirements. The presentation proposes a range of test "layers" that extend beyond standard random instruction generation to include system integrity verification. This presentation will demonstrate a number of these verification capabilities, how they have improved RISC-V verification, and how they may meet the certification needs of the future.

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.

BREKER™

RISC-V CoreAssurance 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

RISC-V SoCReady 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™

SystemVIP with Test Synthesis Amplification

Pre-packaged test availability

Quality SystemVIP, broad test suites

Ultrahigh coverage & bug hunting

Auto bug tracking for complex scenarios

Portable & Reusable

Same tests across platforms and projects

- Cache-System Coherency
- Arm SoCReady
- RISC-V SoCReady
- RISC-V CoreAssurance
- Power Management
- Security
- Networking & Interfacing

Advanced RISC-V Core and SoC Verification Towards the Anticipated Certification Requirements

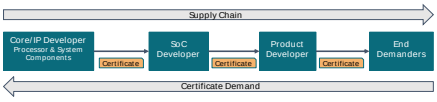
Verification Futures Austin 2024

Adnan Hamid

A Look At RISC-V

- Open Instruction Set Architecture (ISA) 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 new thinking
- RISC-V International developing certification program to provide an assurance metric for RISC-V devices

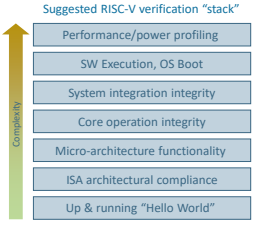
RISC-V International Certification



- RVI Certification Steering Committee (CSC) recently formed with the intent of increasing industry confidence in certified RISC-V cores and IP
  - Considered important as RISC-V gains in popularity
- CSC now working on test sources, process plans, etc.
  - May look to commercial entities to provide rigorous certification tests
- Breker involved as our SystemVIPs are aligned with potential CSC tests

Meeting RISC-V Verification Challenges

- Reuse & automation to meet quality expectation
  - Automated test generation key
- RISC-V special requirements
  - Custom instruction verification
  - Compliance assurance
  - Broad range of architectures
- Different processors have different needs
  - Embedded cores
  - Processor clusters
  - Application processors



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

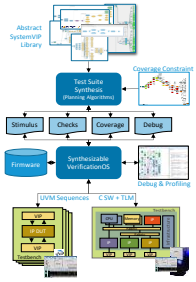
RISC-V SoC Verification Challenges

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

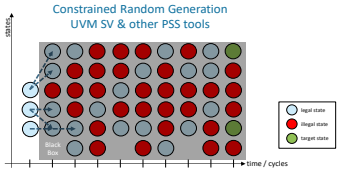
Breker Background: Test Suite Synthesis for RISC-V Cores & SoCs

- Breker is a key, longstanding part of the verification ecosystem for processors and SoCs based on x86 and Arm architectures
- Breker has become part of the verification ecosystem for processors and SoCs based on RISC-V architectures
  - Working with multiple RISC-V developers and users/integrators
- RISC-V has room to grow if we solve the verification barrier
  - We are experienced in x86 and Arm verification, now are sharing this experience with RISC-V teams through automated tests

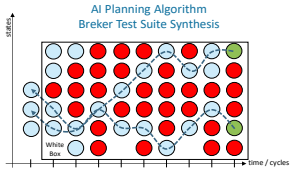
The Breker SystemVIP Library
- *Core Integrity FastApps*
- *RISC-V System Integrity TrekApp*
- *ARM System Integrity TrekApp*
- *Cache Coherency TrekApp 2.0*
- *Firmware-First TrekApp*
- *Power Management TrekApp*
- *Security TrekApp*
- *Networking TrekApp*



Constrained Random vs AI Planning Algorithm Synthesis

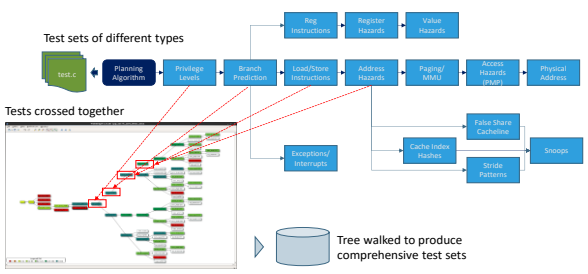


Design black box, shotgun tests to search for key state
Low probability of finding complex bug

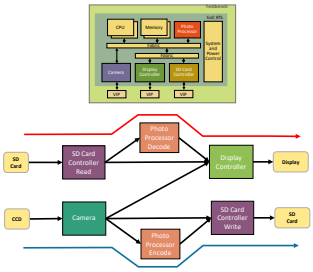


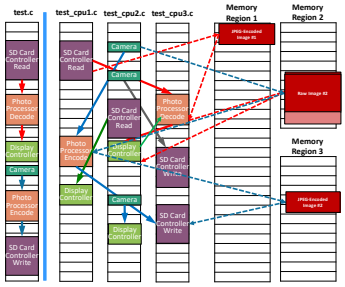
Starts with key state and intelligently works backward through space
Deep sequential, optimized test discovers complex corner-cases

Crossing RISC-V Core Verification Components



Concurrent Test Execution





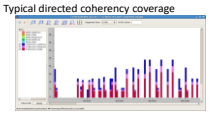
High Coverage and Bug Hunting

Recent examples of bugs discovered in real designs

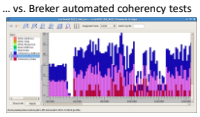
- RISC-V spec misunderstanding between core vendor and user
- Coherent Mesh Network (CMN) programming issues
- Misconfigured ARM CMN pin to enable coherent traffic
- DDR model unable to handle AXI "wrap" transactions.
- Common cache line access reveals deadlock
- Custom instruction bugs discovered by stress tests
- Results mismatch with ultrawide address strides
- Incorrect exception for guest virtual address[63:38] = 0x1fffffff
- Bad mcause value for guest physical address[63:31] != 0x0

SystemVIP Test Suite Synthesis Coverage Comparison

Typical directed coherency coverage



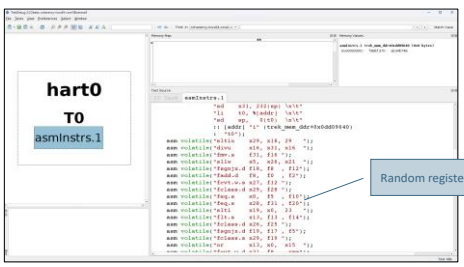
... vs. Breker automated coherency tests



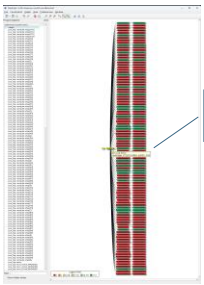
Core-Integrity Challenges

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

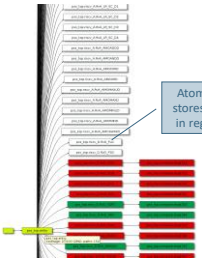
RV64 Core Instruction Generation



Instruction Coverage Analysis

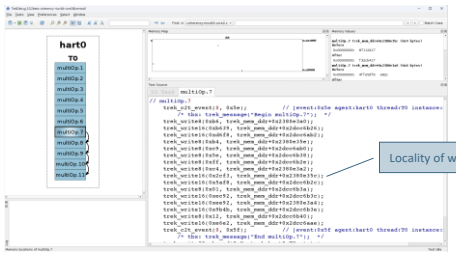


27/103 reachable opcode have been exercised



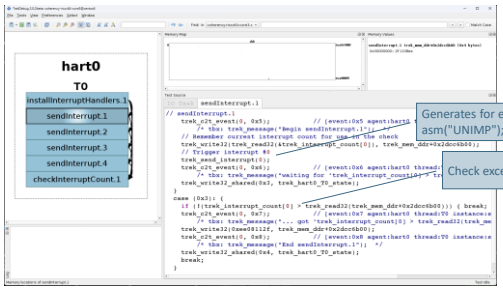
Atomics, loads and stores not reachable in register only test

RV64 Core Load/Store



Locality of write addrs

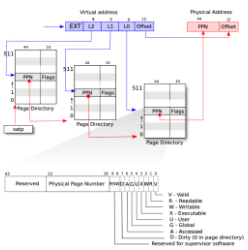
RV64 Core Exception Testing



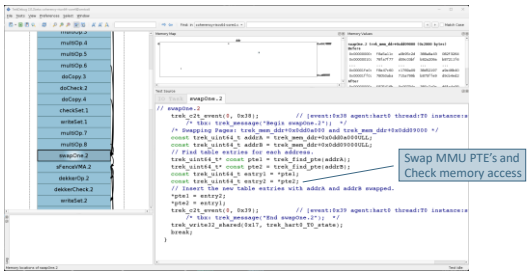
Generates for example, asm("UNIMP");

Check exception counts

Page Based Virtual Memory Tests



RV64 Core Page Based MMU Tests

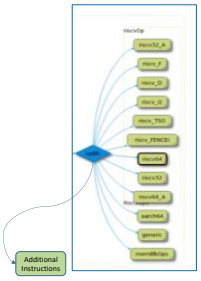


Core-Integrity: Single Core, 4 Threads

Tests utilizes processor's available resources/software threads

Testing a Custom Instruction

- RISC-V ISA custom instructions pose a particularly difficult verification challenge
- Custom instructions need to be tested with the processor tests, not as an afterthought
- Breker solution allows custom instruction tests to be easily added into test graph
- Breker synthesis combines these tests with the app to ensure full custom processor testing



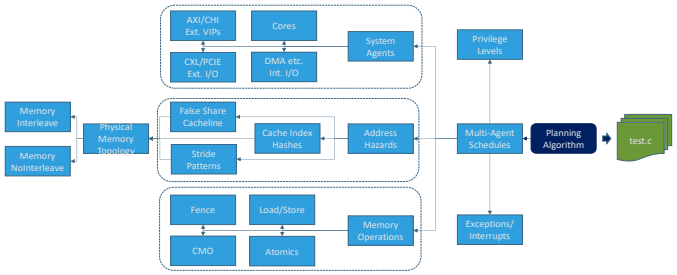
SoC-Integrity Challenges

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
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

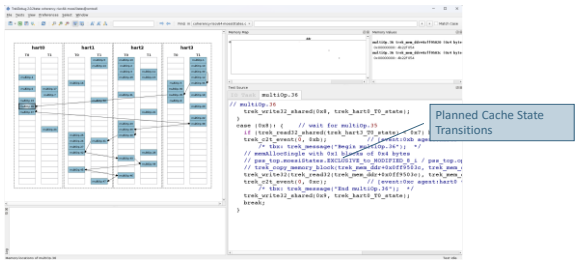
Breker
RISC-V SoC-Integrity
SystemVIP

- **End-to-End use cases**
- **Early Firmware Testing**
- **Performance-Power Profiling**

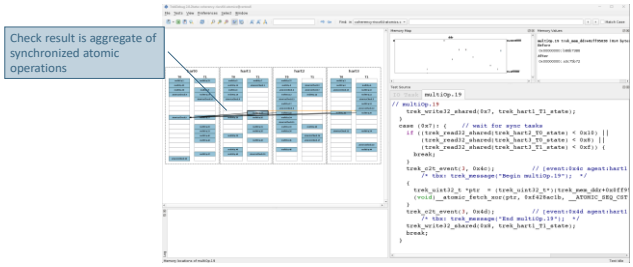
RISC-V SoC Integrity TrekApp



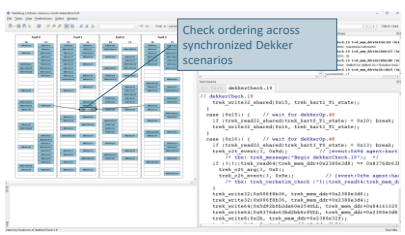
RV64 MultiCore MoesiStates



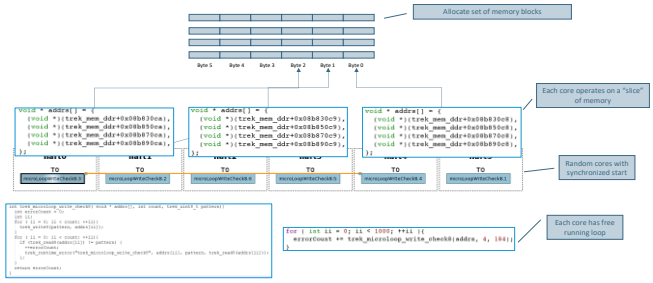
Atomics Testing



Dekker Memory Ordering



False-Share Memory Stress Tests



Summary

- RVI is creating a certification program that requires rigorous testing
  - Probably will be an important part of RISC-V development in the future
- Breker is now providing state-of-the-art test solutions that accelerate and amplify RISC-V core and SoC test quality
- RVI certification is likely to borrow heavily from commercial verification solutions such as Breker SystemVIP and Test Suite Synthesis

Thanks for Listening!
Any Questions?

Notes

Dilip Kumar

TessolveDTS Inc
Design Lead

Practical applications of machine learning in design verification and ISO 26262 practices

User Paper

Abstract
ML (Machine Learning) is transforming the way we work in a wide range of industries and this has been accelerated by generative AI (Artificial Intelligence) applications such as ChatGPT. In this presentation we investigate how ML and AI can potentially be applied in DV (Design Verification), from automation of requirements/specification analysis and test plan generation, through test bench creation and test generation, to debug and coverage closure. There is a very wide range of ML techniques available and this presentation first surveys those techniques and how they have been applied (successfully and unsuccessfully) to DV in both academia and in real projects. The objective is to better understand how to apply the most promising techniques to a wide range of DV activities to ultimately make DV both more efficient and effective. The main objective of the presentation is to give the audience a better understanding of what is achievable of applying ML in DV and to give practical suggestions on their adoption.

The integration of Generative AI (GenAI) into ISO 26262 practices represents a transformative approach to automotive functional safety. As the demand for safer automotive systems grows, it is essential to leverage strengths from other domains to enhance automotive safety practices. Recent advancements in Generative AI models and practices have opened new doors to innovative approaches. Beginning with an understanding of Functional Safety concepts, the trained AI model proves useful in Hazard Analysis and Risk Assessment (HARA), requirements management, Failure Mode and Effects Analysis (FMEA), Automotive Safety Integrity Level (ASIL) determination, and compiling compliance evidence The presentation features several noteworthy use cases where the trained AI model is particularly useful for the Functional Safety team.

Biography

Dilip has 10 years of industry experience as a design verification engineer and has worked extensively on verifying designs at full chip level as well as subsystem and IPs. Dilip has worked with leading semiconductor companies on world class products and has gained in-depth knowledge on the concepts of verification. Dilip currently works as a verification consultant for Xilinx-AMD and is involved in verifying the FPGA designs at full chip level.

TESSOLVE
A HERO ELECTRONIX VENTURE

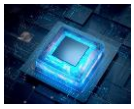
Practical Application of Machine Learning in Design Verification and ISO26262

Mike Bartley, SVP, CoE | Marmik Soni, Lead, CoE | Dilip Kumar, Lead, CoE


Chip Design


Test Engineering


Hardware Design


Embedded Systems

Ver0.1MS

Proven Capabilities Serving Leading Semicon Companies

TESSOLVE
A HERO ELECTRONIX VENTURE

Deep domain expertise across service lines built over a period of time

	VLSI Design	Test Engineering	PCB Design	Embedded Systems / Products
Strengths & Capabilities	**Digital** • RTL (ASIC / FPGA) • Design Verification • Design for Test • Physical Design / STA • FPGA Emulation • Pre-Si Validation **Analog** • Feasibility Study • Macro Modeling • Circuit Design • Layout Design **AMS / SoC Design** • IP Level Verification • AMS Verification • Chip Integration • Full Chip Verification	• Test Program Development • Platform Conversions • Bench To ATE Correlation • Yield Optimization • Failure Analysis	• ATE Board Design • Component Library • Simulation Services • Eval / EVM Boards • Prototypes	• System Hardware Design & Integration • System Development • System Manufacturing • System Level Test • Embedded Software Services
Key Expertise	180nm to 7nm Advanced Process node down to 7nm including high voltage process node up to 180 nm VIPs Verification IP (VIP) solution for different protocols & IPs	900+ Test programs released 30+ Automated Test Equipment supported Trusted Thought Partner to Clients	130+ PCB designs per month 60+ Experience with 60+ layers PCB design Critical Partner and Vendor of Choice	20+ Embedded Systems released 50% Over 50%employees with 10+ years experience

Outline

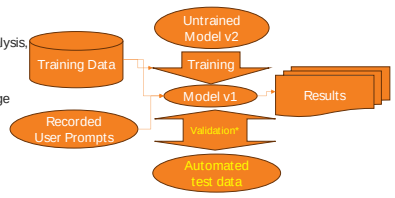
TESSOLVE
A HERO ELECTRONIX VENTURE

- AI Strategies
- Background
  - Where are the main challenges in verification?
  - Where do we think AI/ML could help?
- Experiences in applying LLM (Large Language Models)
- Experiences in applying ML to
  - Coverage closure
  - Debug
  - Regression suites
- Practicalities
- Application in ISO26262
- Conclusions

AI Strategy

TESSOLVE
A HERO ELECTRONIX VENTURE

Features:
- Preserved training data
- Preserved prompts
- Model selection & upgrades (for conversational, coding assistance, analysis, computational needs)
- Response validation
- Migration to AI agents
- Needs and security-based AI solution range

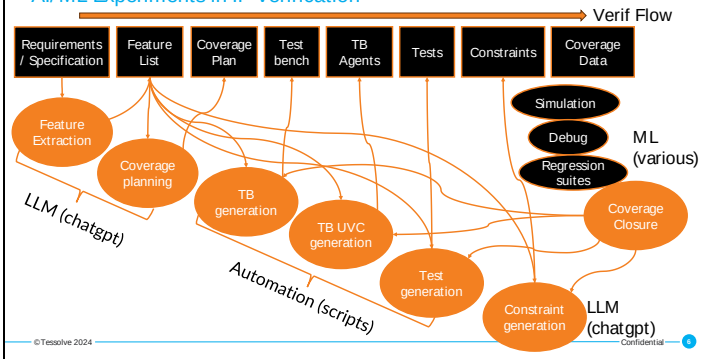


Currently validation is manual*

Historic summary from Tessolve "Verification Futures"

Challenge	Mentions	
Complexity	12	⇔ Non-mover: Design complexity stabilising
Debug	12	
Resources	12	
Integrating Methods, Languages and Tools	10	⇔ Non-mover: 70% projects use UVM; All other methods <10%
Completeness of the verification e.g. coverage	10	⇑ On the rise: 44% of IC/ASIC projects are safety-critical
Safety verification	9	
Scalability	8	
Mixed Signal	7	
Power Verification	6	
Productivity	5	58% of IC/ASIC projects add security features to their designs
Reuse	5	
HW/SW	5	
Security verification	5	⇑ On the rise
Using AI/ML in verification	4	⇐ New entry – the focus for the day

AI/ML Experiments in IP Verification



Simple Example: Xilinx Synchronous FIFO V3.0

Applying LLM to Specification Analysis and Feature Extraction

Specification

Signal Name	I/O	Description
DIN[N:0]	Input	Data Input
WR_EN	Input	Write Enable (request)
WR_ACK	Output	Write Acknowledge: Handshake signal indicates successful write operation
WR_ERR	Output	Write Error: Handshake signal indicates unsuccessful write operation
RD_EN	Input	Read Enable (request)
RD_ACK	Output	Read Acknowledge: Handshake signal indicates successful read operation
RD_ERR	Output	Read Error: Handshake signal indicates unsuccessful read operation
CLK	Input	Clock for read and write operations
SINIT	Input	Synchronous initialization of all FIFO functions, flags, and pointers
DOUT[N:0]	Output	Data Output
FULL	Output	Full Flag: Indicates that no additional writes can be performed
EMPTY	Output	Empty Flag: Indicates that no data is present
DATA_COUNT[C:0]	Output	Data Count: Vector

Features to be Verified:

Data Read and Write Operations:

Verify that data presented at the data-input port (DIN) is correctly written into the FIFO on a rising clock edge when the write-enable input (WR_EN) is high.

Verify that data can be read out of the FIFO via the data-output port (DOUT) in the order it was written by asserting read-enable (RD_EN) prior to a rising clock edge.

Status Flags:

Verify the behavior of the memory-full status output (FULL) to indicate when no more empty locations are available in the internal memory.

Verify the behavior of the memory-empty status output (EMPTY) to indicate when no more data resides in the internal memory.

Invalid Request Handling:

Verify that the FIFO status is not corrupted by invalid read or write requests.

.....

CPU Register Extraction using AI Model

Sr. No.	Register Name	Register Address	Register Size	Access	Importance	Functionality
1	[redacted]RegX	0x33	32 bits	R/W	High	Stores the 32-bit secret key value used for secure operations.
2	[redacted]RegX	0x33	32 bits	R/W	High	Stores the 32-bit decryption key value used for secure operations.
3	Host Interrupt/ Debug Register	0x34	32 bits	R/W	Medium	Contains interrupt and debug-related information for host communication.
4	[redacted]Status	0x34	1 bit	R	Medium	Indicates the presence of an interrupt condition.
5	IntEnable	0x34	1 bit	R/W	Medium	Enables or disables interrupts.
6	KeySize	0x34	1 bit	R/W	Medium	Selects the key size (128-bit or 256-bit).
7	[redacted]AccessGranted	0x34	1 bit	R	Medium	Indicates whether debug access is granted or not.
8	HostIntRegClr	0x34	1 bit	R/W	Medium	Clears the host interrupt.
9	External Status Register 0	0x50	32 bits	R	Low	Connected to the chip top for external status information.
10	External Status Register 1	0x54	32 bits	R	Low	Connected to the chip top for external status information.
11	External Status Register 2	0x58	32 bits	R	Low	Connected to the chip top for external status information.
12	Lifecycle Register	0x5C	32 bits	R	Low	Contains the lifecycle word read from OTP memory.
13	Clock/ Reset Register	0x60	32 bits	R/W	Low	Controls the reset and enable signals for the subsystem.
14	[redacted]Control/ Status Register	0x64	32 bits	R/W	Low	Control and status register with undefined usage.

FIFO ctd. - Applying LLM to a simple directed test bench

```
module tb_fifo;
// Declare signals and variables for the testbench
 logic clock; logic reset;
 logic wr_en; logic rd_en;
 logic [DATA_WIDTH-1:0] data_in;
 logic [DATA_WIDTH-1:0] data_out;
 // Instantiate the FIFO module
 fifo #(DATA_WIDTH, FIFO_DEPTH) dut ( .clock(clock), .....)
 always #5 clock = ~clock; // initialize signals and variables
  initial begin clock = 0;
  reset = 1;
  wr_en = 0;
  rd_en = 0;
  data_in = 0; // Reset the FIFO
  #10 reset = 0; // Write data into the FIFO
  #20; wr_en = 1;
  data_in = 8'hFF; // Example data value
  #10; wr_en = 0; // Read data from the FIFO
  #30; rd_en = 1;
  #10; rd_en = 0;
  #10; $finish;
end // Display the output data
always @(posedge clock)
begin
  $display("Data Out: %h", data_out);
end
endmodule
```

Simple directed test bench

Assertion for FIFO empty and full signals

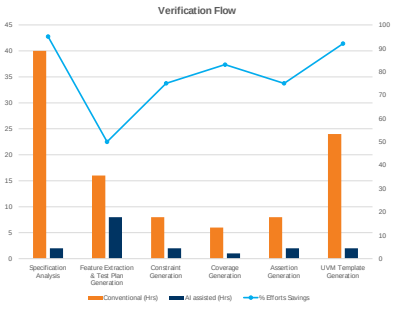
```
assert property (fifo_inst.EMPTY |-> !fifo_inst.FULL);
```

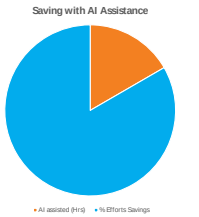
```
covergroup fifo_coverage @(posedge clk);
 // Declare coverpoints for different events
 coverpoint data_count {
   bins empty = {0};
   bins non_empty = [1:$];
 }
 coverpoint write_enable;
 coverpoint read_enable;
 coverpoint full;
 coverpoint almost_full;
 coverpoint almost_empty;
 // Declare cross coverage
 cross data_count, write_enable;
 cross data_count, read_enable;
 cross full, write_enable;
 cross almost_full, write_enable;
 cross almost_empty, read_enable;
endgroup
```


Applying LLM on real projects

DV Flow	Conventional Efforts	AI Assisted Efforts	Saving		
			Engineering Efforts Saved (Hrs)	% Efforts Savings	Speedup (times)
Specification Analysis	1 Week, 1 Engineer	2 Hrs, 1 Engineer	38	95	20
Feature Extraction & Test Plan Generation	2 Days, 1 Engineer	1 Days, 1 Engineer	8	50	2
Constraint Generation	1 Day, 1 Engineer	2 Hours, 1 Engineer	6	75	4
Coverage Generation	6 Hours, 1 Engineer	1 Hour, 1 Engineer	5	83	6
Assertion Generation	1 Day, 1 Engineer	2 Hours, 1 Engineer	6	75	4
UVM Template Generation	3 Days, 1 Engineer	2 Hours, 1 Engineer	22	92	12

Savings on Hours with AI Assistance





Conventional (Hrs)	AI assisted (Hrs)	% Efforts Savings
102	17	85

Applying ML to debug

Daniel Hanson,
Verifyter (now Cadence) Tessolve DVClub 2021

Debugging failures from a regression

- Indicators from code source for where to look for the root cause
  - Commit time
  - Number of editors (1, 2, many)
  - Multiple files committed together
  - History of "bugginess" of a file
  - Change in density of comments
  - Who made the commit!

How to use this?

- First use historic data for training

Once we have a trained model

- Indicate where to start looking
- Identify "risky" commits as they happen
  - Run more check-in tests?
  - Require reviews of riskiest commits?
  - Use lint for design and test bench code

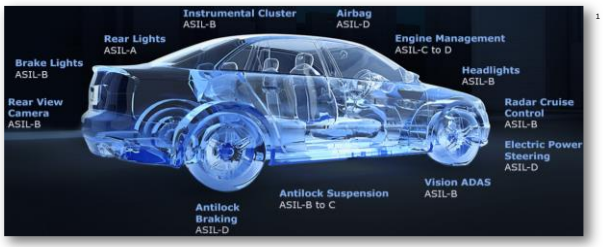
Tessolve Verif Futures USA Sept14 2023

Practicalities of applying AI/ML in verification

- Use of LLM
  - Understanding on using LLM
- Data Set size
  - Large data sets (of clean, unbiased data) are needed
- Security
  - LLMs use the cloud normally

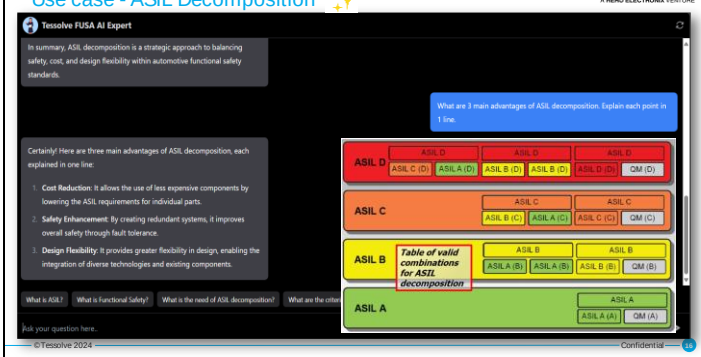
ISO26262 – Automotive Functional Safety

- ISO 26262 is an international standard for the functional safety of electrical and electronic systems in production automobiles.
- A framework to ensure that safety-related systems perform reliably and safely throughout their lifecycle.



Use case - ASIL Determination

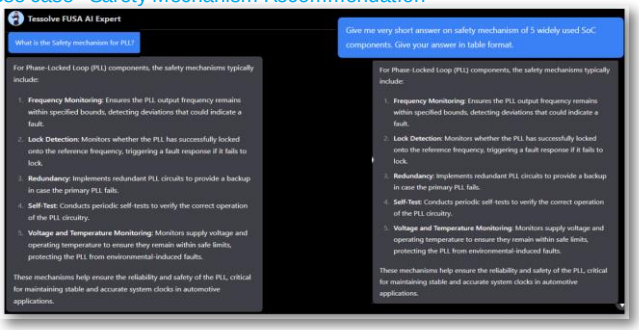
Use case - ASIL Decomposition



Use case - HARA

Hazard ID	Hazard Description	Severity (S)	Exposure (E)	Controllability (C)	ASIL	Safety Goal (SG)	Safety Measure (SM)
H1	Both headlights fail simultaneously	S3	E3	C3	ASIL C	SG1: Ensure both headlights do not fail simultaneously	SM1: Use redundant power supplies and control circuits. Design the headlight system with dual independent circuits.
H2	One headlight fails	S2	E3	C2	ASIL B	SG2: Ensure at least one headlight remains operational if the other fails	SM2: Implement diagnostic checks that can isolate a failed headlight and ensure the other continues to function.
H3	Headlights fail to switch to low beam from high beam	S2	E3	C2	ASIL B	SG3: Ensure proper switching between high and low beam	SM3: Use reliable sensors and control algorithms to manage the switching of beams based on real-time data. Employ adaptive lighting systems that respond to oncoming traffic and environmental conditions.
H4	Headlights do not turn on automatically in low light conditions	S3	E2	C2	ASIL B	SG4: Ensure automatic activation of headlights in low light conditions	SM4: Incorporate ambient light sensors and fail-safes to activate headlights when low light conditions are detected. Regularly test and calibrate sensors.

Use case - Safety Mechanism Recommendation



Conclusion

- AI/ML will change the way we do verification, results are promising with
  - Spec analysis, Feature extraction, test bench infrastructure
  - EDA are adding AI/ML under the hood
- Challenges are,
  - Model accuracy
  - Rigorous validation
  - Reliability
  - Data security
  - Standard for GenAI usage

TESSOLVE
A HERO ELECTRONIX VENTURE

THANK YOU

www.tessolve.com
mike.bartley@essolve.com
Click to add text

Chip Design | Test Engineering | Hardware Design | Embedded Systems

Notes

Larry Lapides

Synopsys
Exec. Director, Business Development

A Holistic Approach to RISC-V Processor Verification

Gold Sponsor

Abstract
Processors using the open standard RISC-V instruction set architecture (ISA) are becoming more and more common, with an estimated 30% of SoCs designed in 2023 containing at least one RISC-V core. Whether licensing RISC-V IP and adding custom instructions, using open-source RISC-V IP, or building a RISC-V processor from scratch, verification of RISC-V processors is a task in the SoC project plan. With the variety of sources for the processor IP, the range of complexity and the span of use cases, a one-size-fits-all approach to RISC-V processor verification does not work.
This session will present a holistic approach to RISC-V processor verification. It will address processor complexity from microcontrollers to application processors to arrays of processors for AI accelerators, different levels of integration from unit to individual processor to processing subsystem to SoC and cover different scenarios depending on the source of the processor IP. Matching different technologies and methodologies to this multidimensional verification space is critical.
In addition, we will elaborate on different decisions that go into the verification plan for RISC-V processors and review the different technologies and methodologies that are employed in a holistic approach to processor verification.

Biography
Larry Lapides is Executive Director, Business Development at Synopsys, responsible for the Imperas-branded products. Prior to Synopsys' acquisition of Imperas Software Ltd. in 2023, Larry was VP of Worldwide Sales & Marketing and a member of the founding team. Before Imperas, he held several roles in sales and marketing including VP of worldwide sales during the run-up to Verisity's IPO. Larry holds a BA in Physics, with General Distinction in Physics, from the University of California Berkeley, a MS in Applied and Engineering Physics from Cornell University and an MBA from Clark University.

SYNOPSYS®

SYNOPSYS®

Synopsys Verification Family

Best-in-Class verification — Complete product family with #1 products in all categories

Fastest Engines — Highest-performance engines accelerate time-to-market

Broad Support — Comprehensive, ready-to-use design, verification, and IP solutions for RISC-V

www.synopsys.com/verification

SYNOPSYS®

A Holistic Approach to RISC-V Processor Verification

Larry Lapides, Exec. Director, Bus. Dev.
Verification Futures Austin, 2024

Agenda

- Why should we use RISC-V?
- Challenge: the RISC-V verification disconnect
- A RISC-V processor verification solution
- Dynamic verification
- Formal verification
- Summary

Why should we use RISC-V?

Anyone can design their own processor based on the RISC-V ISA

Modular ISA = choice of which features to include/exclude

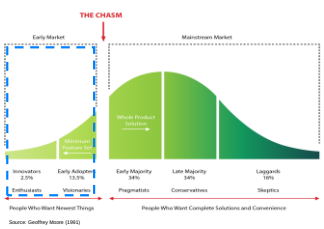
Extensibility and freedom to customize at ISA and micro-architectural levels

RISC-V enables the creation of domain-specific differentiated processors

RISC-V is Crossing the Chasm: 2023-2024

Moving beyond early adopters, into early mainstream

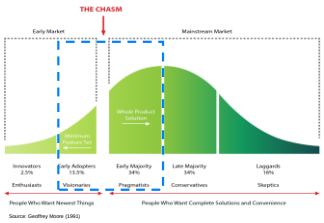
- Initially only used by 'visionaries' like SiFive, Andes, Nvidia, Microchip
- Then systems companies wanting domain specific processors
  - Meta Infrastructure, Google, …
  - IoT companies
  - and early adopter semiconductor companies e.g. Qualcomm, Nvidia Networking (Mellanox), Silicon Labs



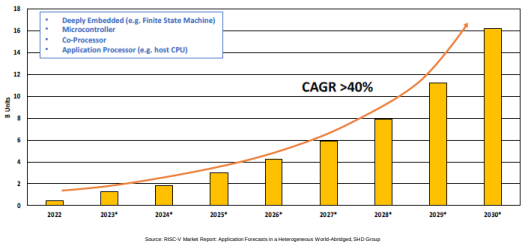
RISC-V is Crossing the Chasm: 2023-2024

Moving beyond early adopters, into early mainstream

- Now…
  - Every semiconductor vendor has a RISC-V SoC project in flight
  - Every hyperscaler company has a RISC-V project at least at the test chip phase
  - Every automotive OEM and Tier 1 has a RISC-V project at least at the test chip phase



Expected RISC-V Market Growth



Source: RISC-V Market Report: Application Forecasts in a Heterogeneous World-Abridged, SHD Group

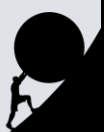
The RISC-V Verification Disconnect

RISC-V Core **User**:
- Expects core quality to be the same as ARM
- 10^{15} verification cycles =10^{4} RTL simulators running 24/7!

RISC-V Core **Developer**:
- Needs to deliver high-quality core
- Potential issues with necessary expertise, methodologies, technologies, resources

Challenges in RISC-V Processor Verification

- Design complexity – architecture, micro-architecture, implementation choices, custom features
- Source of processor IP (in-house, open source, vendor + custom instructions)
- Use case: microcontroller – application processor; closed versus open to external software development
- Verification productivity and time to closure
- Team experience (designers and verification engineers)
- Processor verification methodology
- Tool selection



What have we learned in the last 7 years?

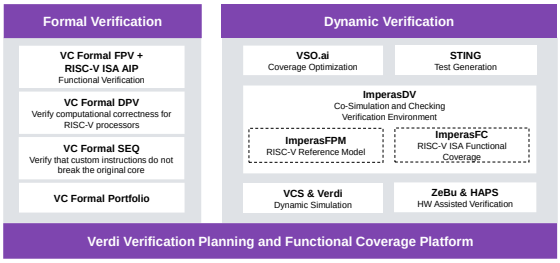
- A verification plan is needed
- Different than with SoC DV, a high-quality, fully functional reference model is needed
- As with SoC DV, the full range of verification technologies is needed
  - Dynamic verification
  - Formal verification
  - Hardware-assisted verification

RISC-V Processor Verification Process

Design verification from unit to SoC

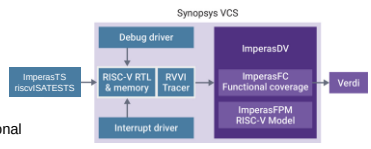
Design Level	Example	Tool/Methodology
Unit	Pipeline, FPU	Formal + predefined assertion IP
	Security	Formal + predefined security assertion IP
Architecture	ISA	Dynamic
		Formal + predefined assertion IP
Custom instructions, CSRs	Custom DSP, matrix	Dynamic
		Formal sequential equivalence checking, register verification, datapath validation
Processing subsystem	Coherent cache, multi- or many-processor accelerator	Dynamic, especially using hardware assisted verification
		Formal property verification for cache coherence verification

Synopsys RISC-V Processor Verification Solutions

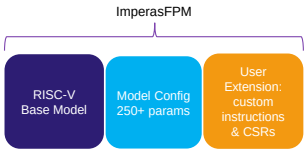


Dynamic Verification: ImperasDV

- **ImperasFPM RISC-V Processor Model**: for comparison of correct behavior; extendable for custom instructions
- **ImperasDV**: provides configuration, comparison and checking, pipeline synchronization and scoreboarding
- **ImperasFC**: deploys SystemVerilog functional coverage code for each ISA extension
- **riscvISATESTS/ImperasTS**: provides directed test suites



ImperasFPMs (Fast Processor Models) for RISC-V



- Base Model implements RISC-V specification in full
- Fully user configurable to select ISA extensions and versions
- Pre-defined configurations and custom instructions for processor IP vendors
- User extensions built in a separate library do not perturb the verified Base Model, help reduce maintenance
- Because every ImperasFPM uses the RISC-V Base Model, and including users of both commercial and free tools, over 150 companies, organizations and universities have used the ImperasFPM

ImperasFC: SystemVerilog Functional Coverage for RISC-V

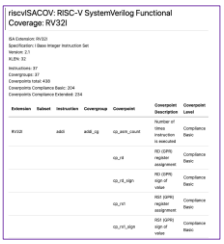
- Functional coverage code generation
  - Manual creation would be tedious, time consuming and error prone
  - >100K lines of code
  - Synopsys tools can automatically generate functional coverage code for custom instructions
- Functional coverage is the key verification metric

Machine-readable RISC-V ISA specification

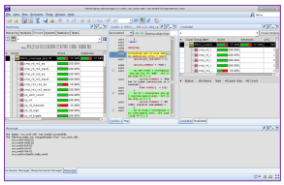
SystemVerilog coverage code generator

https://github.com/riscv-verification/riscvISACOV/tree/v20240124/documentation for list of covered extensions

Integrating ImperasDV with Verdi



Auto-generated documentation in markdown and csv formats for inclusion in Verification Plans



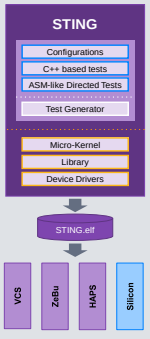
Functional coverage data is reported in verification tools such as Verdi

STING

Preventing bug escapes for complex RISC-V designs

- Self-checking test generation for RISC-V
- Addressing single CPU and complex many core SoC designs
- Generates constrained-random, directed stimulus, and combinations of the two
- Portable across simulation, emulation, prototyping and silicon
- Full support for the RISC-V ISA specification
- Extensible to custom instructions and peripheral devices

- ImperasDV adds comprehensive checking and functional coverage when STING output is used in VCS

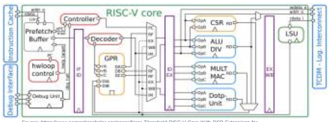


Formal Verification: VC Formal

- Formal verification provides exhaustive proof of correct behavior
- Excellent tool for unit-level DV
  - Can get started early, even with design engineers
  - Unit-level includes pipeline, floating point unit, load/store unit, …
- RISC-V ISA Assertion IP (AIP) available to enable early use of VC Formal
- VC Formal Apps improve verification efficiency of many tasks
  - Register verification, datapath validation, connectivity checking, security verification …

VC Formal Apps

RISC-V Formal Verification



Source: https://www.semanticscholar.org/paper/Near-Threshold-RISC-V-Core-With-DSP-Extensions-for-GautschiSchiavone/47f8ce7e0f0f64d0707a13c83c32c30959aa64d5/figure/6

- VC Formal FPV + AIP (Model Checking):
  - Prefetch Buffer
  - LSU – Load/Store unit
  - Pipeline
- VC Formal DPV (Equivalence Checking):
  - ALU/MULT/Dotp
  - Decoder
- SEQ (Equivalence Checking):
  - Clock gating verification in every functional unit
  - Design comparison in presence of new features/timing changes
- FRV (Formal Register verification)
  - CSR
- FSV (Formal Security Verification)

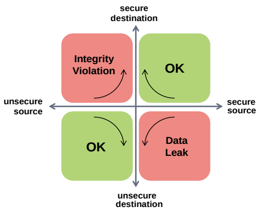
RISC-V Assertion IP (AIP)

RV32I base ISA, for example:
- LOAD - LSU
- STORE - LSU
- BRANCH/JUMP/LUI/AUIPC - PFU
- OP-IMM - EXU
- OP - EXU

– Environment call/break point
– Zicsr extension
– CSR Write
– CSR Read

VC Formal FSV: Formal Security Verification

Ensure data security objectives are met through exhaustive formal analysis



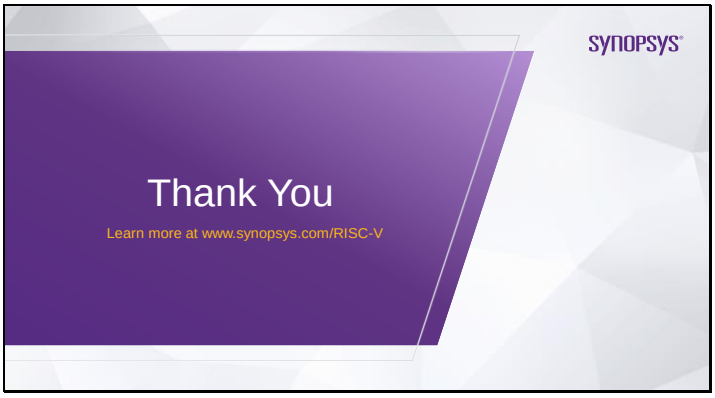
VC FORMAL FSV

- Detect security issues that are hard to find through other techniques
- Users define security properties to check for data integrity and data leakage issues
  - Flexible property creation and management
- ML powered engines for fast performance
- Data propagation analysis and debug with temporal flow view
- Verification of multiple scenarios in one session

What have we learned in the last 7 years?

Starting point:
- A verification plan is needed
- Different than with SoC DV, a high-quality, fully functional reference model is needed
- As with SoC DV, the full range of verification technologies is needed
  - Dynamic verification
  - Formal verification
  - Hardware-assisted verification

Keys to Success:
- Support for full RISC-V specification
- Custom instructions easily added and verified
- Silicon-proven processor verification tools, models, and methodologies


SYNOPSYS
Thank You
Learn more at www.synopsys.com/RISC-V

Vikram Khosa

Arm

Principal Engineer

Pushing forward the frontiers of formal in Arm CPUs

User Paper

Abstract

Over the last decade, formal verification has been steadily maturing into a mainstream methodology and seeing serious adoption across various semiconductor companies, including Arm.

However, the long-held promise of being able to deploy formal to exhaustively verify designs in a timely manner is still limited by size, complexity, and cost of compute. In addition, opportunities for novel formal use-cases are also tempered by their own fresh set of challenges.

This talk covers some of these challenges and the innovative solutions being developed to address the twin demands of convergence and efficiency.

Biography

Vikram Khosa has been spearheading initiatives in formal methodology as well as involved in hands-on deployment of formal methods for functional and security verification, across various roles in A-class CPU design projects, at Arm's Austin design center since 2013. Previously, he led memory-system verification for the Cortex A15 CPU. He has also worked in various verification roles for other companies, including 2 early-stage start-ups. He holds a master's in computer engineering from the University of Minnesota, Twin Cities and a B.E. (Hons.) from BITS Pilani.

Notes

Outline

- Early Hurdles
- Path to Mainstreaming
- Present Challenges
- Contours of the Future
- Vectors of Innovation
- Conclusions

arm

Early Hurdles

Early Hurdles

- Designer Engagement
- Management Buy-in
- Right scope
- Ownership
- Resourcing
- Criteria for Success/Completion

arm

arm

Path to Mainstreaming

Path to Mainstreaming

- Integrating formal in development flow
  - Goal to get formal out of its silo
  - Evolve away from centralized team
  - Position formal as a desirable skill for all verification engineers
  - Formal engineer embedded within the respective unit team
    - Owns both simulation and formal
    - Move TB to formal-only opportunistically

arm

Path to Mainstreaming

- Scoring minor successes and building on them
  - Critical to developing credibility
- Key drivers towards achieving left-shift
  - Datapath C2RTL + theorem-proving (novel methodology)
  - Specialized flows : Clock-gating, X-prop
  - Formal TBs around smaller units
    - Precondition : ability to stay in sync with RTL development
    - Required a dedicated, focused formal engineer
    - Eventually enables formal-only TB

arm

Path to Mainstreaming

- Role of management
  - Strategic patience supporting long-term vision
- Organizational adjustments
  - Consciously push engagement with val/unit leads
  - CPU Formal Council to enable cross-site alignment
  - Cross-site working groups for technical collaboration & reuse
- Smart bets on small/special problems
  - Sustain focused resourcing till we see results
- Limited investment in solving tougher challenges
  - Become drivers to develop new technologies/flows
  - Enables progressive scaling formal to bigger problems
  - Architectural-specification based / end-to-end flows

arm

arm

Present Challenges and Solutions

Compute Cost -- Challenges

- Overall compute footprint driven by increasing
  - design/feature complexity
  - deployment surface
- Increased compute bandwidth requirements
  - driven by shorter overlapping project cycles
  - reduced time-to-market across increased number of served segments
- Formal is no exception
- Additional factors
  - More unit/multi-unit environments
  - More users
  - More design configurations → more regressions
  - More specialized formal flows (clock-gating, X-prop, functional-safety, security)
  - More bug-hunting strategies
  - More formal-engine solver threads/partitions distributed across cluster

arm

Compute Cost – Cluster/Flow-based Solutions

- Cloud Migration
  - Demand-based cost structure
  - Increased efficiency for burst/peak demands
  - Pre-emption overhead
- Dedicated lower-cost clusters
  - Lower-cost CPUs
  - Minimize/avoid contention with other formal/non-formal runs
  - Also improves startup latency
- Debug-fitted regressions
  - Don't generate more fails than you have resources to debug between runs
  - Stop regression after some number of fails
  - Don't generate the same failure more than once
- Optimally-sized regressions : don't spread resources too thin
  - Running on too many (disparate) properties at once can undermine effectiveness and coverage

arm

Compute Cost – Tool-supported Solutions

- Proof Caching/Profiling
  - Restore results from previous run if no changes
  - Learn optimal engine settings etc. across regressions
  - Bound aggregation
- Hybrid Flows
  - Enabled by stagnation-detection
  - Exhaustive Proofs followed/overlapped by Bug-Hunting
  - (Lack of) Overlap controlled by saturation of progress
- ML-based Optimizations
  - Proof strategies within and across runs
  - Tuned to specific designs/properties

arm

Complexity Squeeze - Challenge

- Ever-increasing complexity across the board
  - Even lower-profile CPUs adding more speculation for performance
- Inter-unit Complexity → Upward Pressure
  - Desire to test complex features across a broader surface-area →
  - Bigger DUTs
  - (Typically) Lower Interface Complexity/Volatility
  - Higher Design-Complexity
  - Tool Burden
- Intra-unit Complexity → Downward Pressure
  - Desire to test at subunit level for better coverage
  - Smaller DUTs
  - Smaller Design-Complexity
  - Higher Interface Complexity/Volatility
  - Designer Burden

arm

Complexity Squeeze - Solutions

- Interface/Stimulus Complexity
  - Intentional Over-Constraints
  - Narrower Scope of Checks
  - Bringup-only TBs
- Design Complexity
  - Transaction Limiting Profiles
  - Abstraction Models
  - IVAs
  - Design Mutations

arm

Designer Engagement - Challenges

- Engagement varies by site
- Common challenges
  - Ownership of interface properties
    - especially related state-tracking
  - Additional formal TB support
    - Abstractions etc.
- Attempts to promote formal-based designer bring-up
  - Only successes were organic and based on designer's innate interest
  - Top-down mandates don't work
  - Positive factors
    - Formal-friendly designer
    - Suitable scope
  - Negative factors
    - Existing mature simulation TBs
    - Excessive schedule pressure

arm

Designer Engagement – Strategies

- Designer training
- Formal integration into smokes and other CI flows
- Push-button debug & fix-qualification flows
  - Sometimes requires dumping FSDBs from FV tool
- Support of a focused FV engineer
  - Offloads maintenance while allowing them to explore the design
  - FV owner support interface constraints that require state tracking
    - Allows them to maintain lighter-weight constraints in some cases

arm

arm

Contours of the Future

Contours of the Future

- Proliferation of FPV
  - Number of environments
  - Size
  - Complexity
- Increased Areas of Focus
  - Regression Efficiency
  - Architecture/Specification Derived Checkers for Niche Areas
    - Decode, RAS
  - Liveness
  - Security
  - Functional Safety
- Areas of Increased Alignment
  - E2E Checker Methodology
  - Front-end Tools & Specialized flows
    - Central formal team to own and support

arm

arm

Vectors of Innovation

Convergence - I

- Human-guided Proof Closure
  - Automated invariant generation based on user-curated list of signals
  - Experiments with engine-level integration
  - JUG Paper '22
- Structured Proofs
  - Proof Decomposition
  - Helper Integration
  - Book-keeping enables Safe Composition
    - Both full and bounded proofs
  - Automatic case-splitting based on key, independent variables
- Proof-orchestration improvements (stagnation-detection)

arm

Convergence - II

- Bug-hunting flow improvements
  - Mitigate effect of free variables (oracles) across a leapfrogging trail of covers
- Trace profiling
  - Mine known bug/cover traces for interesting covers/transitions
  - Apply these as soft constraints to direct engines to similar scenarios
- Regression clumping
  - Smaller groups of similar properties
    - Similar COI, complexity
  - Exploit locality
    - Multi-property engines, manual out-of-COI abstractions
- Flow-graph based approach (JUG Paper'22)
- Bound Aggregation for elastic BMC

arm

Special Flows

- Architectural Specification based flows
  - ISA-F
  - MMU-F
  - Decoder-F
  - MEM-F
  - CAT2SVA
- Liveness, FSM deadlocks
- X-Prop & Clock-gating Improvements

arm

Novel Use-Cases

- Security (information-flow based)
  - Evolution of semantics & operators
  - Taint-propagation based abstractions
  - Exploration of various attacks/vulnerabilities
    - Spectre, Meltdown, Spoiler, Spectre-BHB
    - JUG Paper '22
  - Verifying mitigations
- Symbolic Simulation
- Multi-Cycle Path Verification

arm

Conclusions

Conclusions

- Formal no longer a fringe/novel technology at Arm
  - as with rest of industry
- Increased maturity/adoption/success → increased compute footprint
- New challenges driven by
  - Dueling demands of efficiency vs. effectiveness
  - Increasing complexity of designs targeted by formal
  - New areas of application for formal methods
- Meeting these challenges and preparing for the future
  - Requires investment in innovation
    - Extending to novel use-cases
    - Reducing compute-cost and complexity
    - Improving convergence

arm

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
תודה

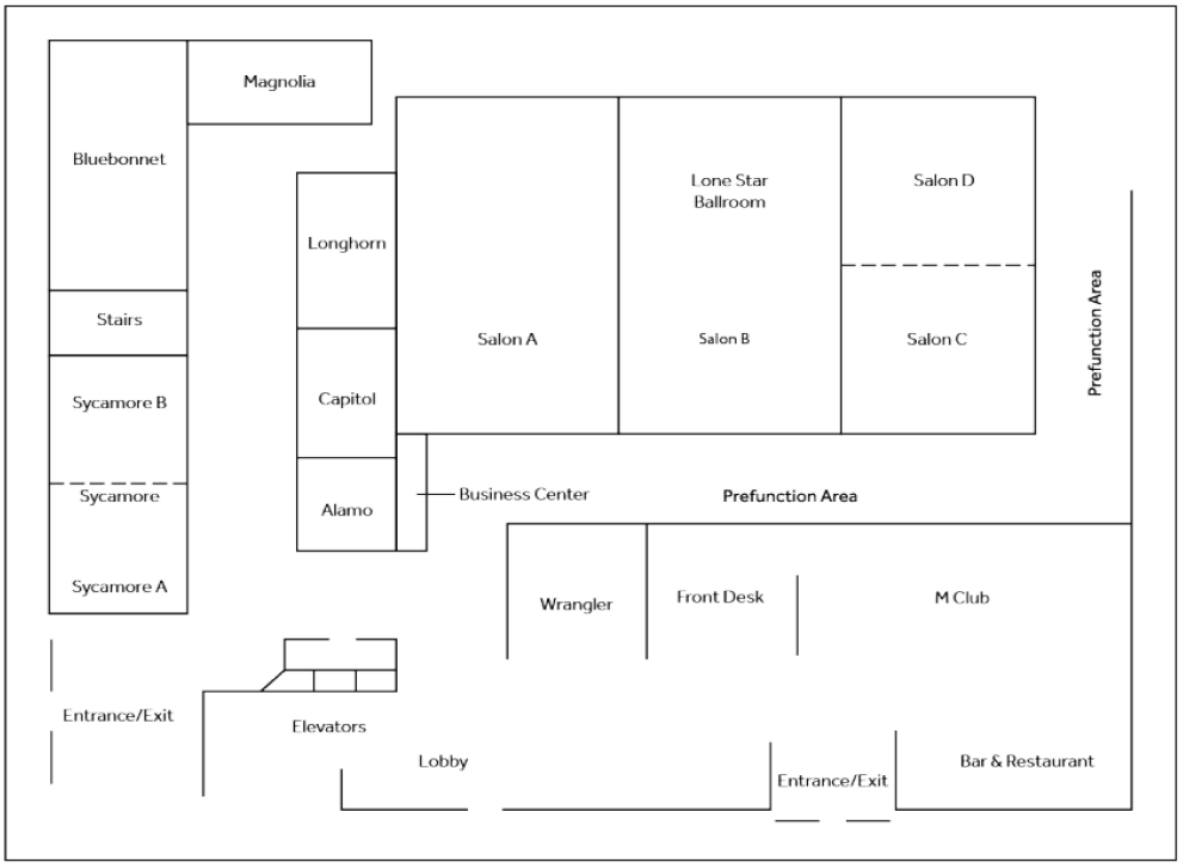
arm

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

Nianhang Hu

University of Nebraska - Lincoln

Graduate Student

Using Symbolic Execution to analyze Hardware TCP/IP StacksBased on HLS Development

User Paper

Abstract

Hardware accelerators developed based on High-Level Synthesis(HLS) are becoming increasingly popular in modern computing systems. Symbolic execution is a powerful technique for analyzing software programs, but its application to hardware accelerators developed with HLS presents some challenges. This paper explores the use of symbolic execution to analyze a real-world HLS-based hardware accelerator, hardware TCP/IP stack. We discuss the challenges encountered in this context, and propose feasible solutions to use symbolic execution to improve code coverage testing. Overall, this paper highlights the great potential of symbolic execution in analyzing HLS hardware accelerators and suggests directions for future research.

Biography

Mr. Nianhang Hu has been pursuing a Master's degree at the University of Nebraska-Lincoln since 2022. He earned his Bachelor's degree in Electronic Engineering from Zhengzhou University in 2005. With extensive industry experience in chip functional verification and bring-up, he has worked at prominent companies including Foxconn, TI, and Nufront. Notably, he led a team of 20 firmware engineers for three years, achieving a 100% first-pass success rate and 98% functional test coverage in bringing up four ARM-based chips. His exceptional skills in both software and hardware debugging have been instrumental in his success.

UNIVERSITY of NEBRASKA
LINCOLN

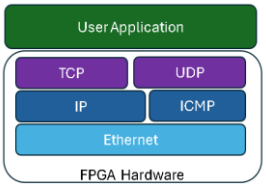
Notes

Using Symbolic Execution to analyze Hardware TCP/IP Stacks Based on HLS Development

Nianhang Hu
12Sep2024e
Department of Computer Science and Engineering
University of Nebraska-Lincoln, Lincoln, NE 68588

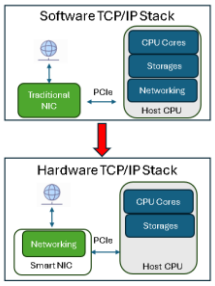
Hardware TCP/IP Stack (I)

- Implementing the TCP/IP protocol on the FPGA.
- Using High-Level Synthesis (HLS) technology to convert high-level languages into hardware description language.



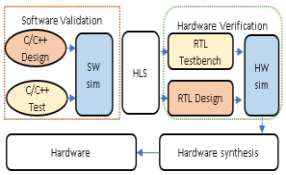
Hardware TCP/IP Stack (II)

- Enhanced Offloading Capabilities.
- Increased Throughput and Reduced latency.
- Programmability
- Reduced CPU Overhead
- Scalability



High-Level Synthesis (HLS)

- Increase design efficiency
- Support high-level abstraction
- Reduce design complexity



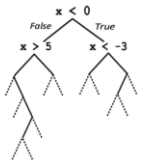
- Difficult to find the correspondence between high-level and low-level code.
- Challenges in testing and verification.
- Lack of effective debugging tools.

Motivation

- Propose a symbolic execution tool.
- Identify issues at an early stage of code development.
- Enhance code coverage for the hardware TCP/IP stack.

Symbolic Execution

```
void read( int x ){
  if (x < 0) {
    if (x < -3)
      foo(x);
    else {
      ...
    }
  } else {
    if (x > 5)
      bar(x);
    else {
      ...
  ...
```

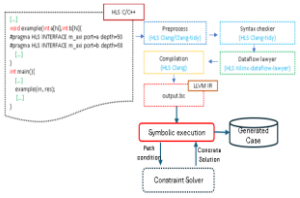


* https://www.researchgate.net/publication/220623844_Cloud9_A_Software_Testing_Service

Analyze C/C++ Code

Problems:

- Special data type in the HLS library
  - Arbitrary precision integer types
  - Streaming processing data types
- Parallel computing in the HLS library



HLS special data type (I)

Using basic data types to construct arbitrary bit-width data types

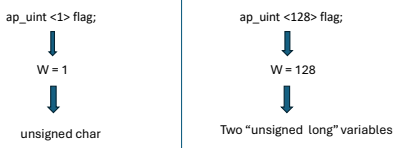
Basic data types:

- unsigned char
- unsigned short
- unsigned int
- unsigned long

→ ap_uint<W> data;

* W: bit width, from 1 to 4096

HLS special data type (II)



HLS special data type (III)

```
void make_symbolic_case1(){
        ap_uint<128> tcpData;
        unsigned long temp[2];
        make_symbolic(&tcpData,
sizeof(tcpData), "tcpData");
        for(int i = 0; i < 2; i ++){
            tcpData.range((i+1) * 64 - 1,i * 64) = temp[i];}}
```

Concurrent Computation

Remove the code that supports concurrent access in the HLS library and replace it with single-threaded sequential access.

- Modify multithread tasks to single-thread tasks.

Refactor functions unsupported by symbolic execution

Implement some library functions that symbolic execution does not support in C++

- std::vector, std::map and std::cout function.

Evaluation

- Enhance the verification efficiency and code test coverage of hardware TCP/IP stacks.
- Using the TCP three-way handshake code as the test subject, comparing symbolic execution and random testing in terms of testing efficiency and coverage.

Q & A

Rahul Kande

Texas A&M University

Ph.D. student

Hardware Fuzzing to Secure Modern Hardware

User Paper

Abstract

Hardware is at the heart of computing systems. However, recent years have seen increased attacks exploiting hardware vulnerabilities and exploits, which even traditional software-based protections cannot prevent. Hardware fuzzing has shown promise in detecting vulnerabilities in large-scale designs like modern processors. In this talk, I will describe the hardware vulnerabilities in hardware description languages, such as Verilog and VHDL. Then, I will explain a new and radical approach called hardware fuzzing to find these vulnerabilities and detail how fuzzing techniques can be combined with existing formal verification techniques to detect vulnerabilities efficiently. Finally, I will discuss a strategy for pinpointing vulnerabilities to accelerate the mitigation process and briefly talk about improving the efficiency of hardware fuzzing using ML/AI techniques, such as multi-armed bandit (MAB) and large language models (LLM).

Biography

Mr. Rahul Kande is a Ph.D. student in the Computer Engineering and Systems Group at Texas A&M University since 2018. He completed his Bachelor of Technology in Electronics and Communication Engineering from the Indian Institute of Technology Guwahati in 2017. He developed TheHuzz, a novel hardware fuzzer that detected new vulnerabilities in popular open-source processors. This work received great traction in the industry and academia, especially Intel and the Office of Naval Research (ONR), USA, who are now jointly funding their future fuzzing projects. Rahul has won several awards and scholarships during his Ph.D. program, such as the departmental Quality Graduate Student Award in 2023 and Graduate Merit Scholarship in 2018, 3rd place at IEEE HOST 2022 Hardware Demonstration, and USENIX Security Symposium Student Grant in 2020 and 2021. His research involves developing more efficient and automated hardware fuzzing techniques to detect vulnerabilities in hardware, especially processors.

More on Rahul Kande: https://www.rahulkande.com/

More on Rahul Kande's research group at Texas A&M University: https://seth.engr.tamu.edu/

Notes

Hardware Fuzzing
Why? What? How?

Rahul Kande

Advisor: Dr. Jeyavijayan (JV) Rajendran

Secure and Trustworthy Hardware Lab

TEXAS A&M UNIVERSITY

Sandia National Laboratories · intel · SYNOPSYS · LOCKHEED MARTIN · ONR · TECHNISCHE UNIVERSITÄT DARMSTADT

Overview

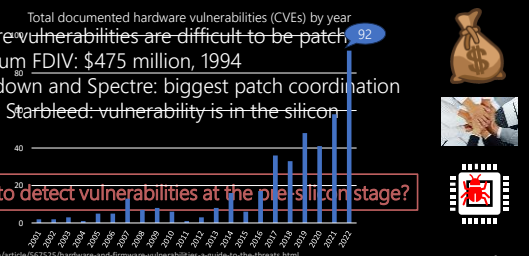
Part1: Hardware Security Vulnerabilities
[GLSVLSI'21,DAC'22]

Part2: Hardware Fuzzing
[USENIX'22]

Part3: Hardware Fuzzing Extensions
[USENIX'23,ICCAD'23,DATE'24,USENIX'24,DAC'24,TIFS'24]

Massive Growth in HW Vulnerabilities

- Hardware vulnerabilities emerging at an alarming rate



- Hardware vulnerabilities are difficult to be patched
  - Pentium FDIV: $475 million, 1994
  - Meltdown and Spectre: biggest patch coordination
  - Xilinx Starbleed: vulnerability is in the silicon

How to detect vulnerabilities at the pre-silicon stage?

https://www.csoonline.com/article/567525/hardware-and-firmware-vulnerabilities-a-guide-to-the-threats.html

Source: National Vulnerability Database NVD (08/2023)

HacktheSilicon

- International hardware security capture-the-flag competitions @ DAC, USENIX Security, and CHES
- Goal: Promote hardware security verification
- RISC-V SoC platform
- Bugs injected in collaboration with Intel

RISC-V · intel inside · CVE · CWE · intel · TECHNISCHE UNIVERSITÄT DARMSTADT · SYNOPSYS

https://hackthesilicon.com/

HacktheSilicon Impact

- 1000+ participants from 35+ countries
- 1000+ vulnerabilities detected
- 10+ competitions

[1] Ghada Dessouky, et al. HardFails: Insights into Software-exploitable Hardware Bugs, USENIX Sec'19.
[2] AR. Sadeghi, et al. Organizing The World's Largest Hardware Security Competition: Challenges, Opportunities, and Lessons Learned, VLSI'21.
[3] Chen Chen, et al. Trusting the Trust Anchor: Towards Detecting Cross-Layer Vulnerabilities with Hardware Fuzzing, DAC'22.

Motivation

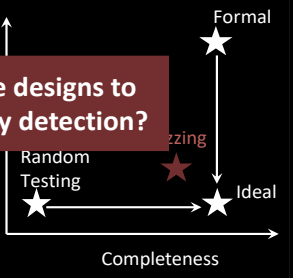
Android Goes All-in on Fuzzing
August 29, 2023

OSS-Fuzz - Google's continuous fuzzing service for open source softw

Google Open S

Google touts 'fuzzing' open source tool after discovering TinyGLTF bug

Can we fuzz hardware designs to accelerate vulnerability detection?

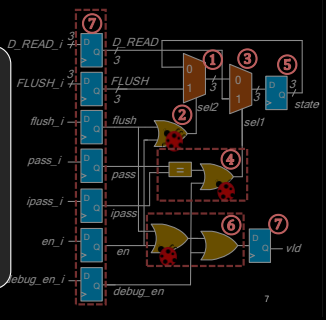


[1] https://security.googleblog.com/2023/08/
[2] https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/serebryany
[3] https://www.darkreading.com/application-security/google-open-sources-ai-boosted-fuzzing-framework
[4] https://therecord.media/google-touts-fuzzing-open-source-tool-after-discovering-tinygltf-bug

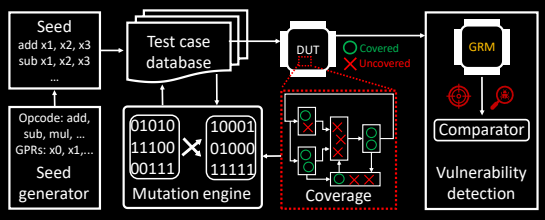
Hardware Fuzzing Framework

Coverage metrics

- Statement: All statements in RTL code
- Branch: Control signals *(sel1,sel3* of ③,①)
- Toggle: 0→1/1→0 transitions of flip-flops ⑦
- FSM: States & state transitions of FSM ⑤
- Condition: Control path combinational logic (*AND* gate in ②)
- Expression: Data path combinational logic (gates in ④ & ⑥)



Hardware Fuzzing Framework



Kande R., et al., **TheHuzz: Instruction Fuzzing of Processors Using Golden-Reference Models for Finding Software-Exploitable Vulnerabilities**, in USENIX Security, 2022.

TheHuzz Impact

Compatible

CHIPS Verilator VERILOG SYNOPSYS

Automated

Practical

Simple to run (60+ students trained)

Efficient

Detected 11 vulnerabilities, higher coverage than existing tech

Hardware Fuzzing Overview

HyPFuzz covers hard-to-reach design spaces

ChatFuzz generates inputs with data and control flow entanglement

Seed
add x1, x2, x3
sub x1, x2, x3
...

Test case database

DUT

Vulnerability detection

LLM-assisted Assertion Generation

Functional: *TheHuzz, HyPFuzz*
Timing: *WhisperFuzz*
Speculative: *Specure*

Seed generator

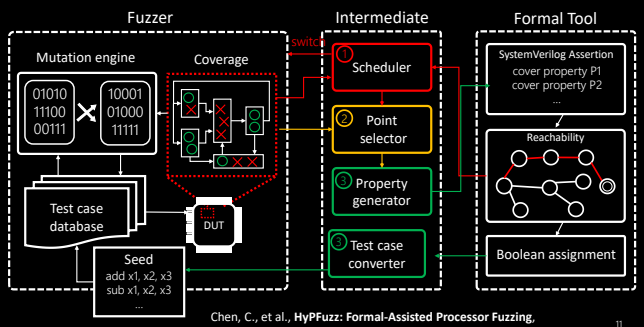
Mutation engine

Coverage

MABFuzz identifies optimal seeds that generate "interesting" inputs

PSOFuzz identifies optimal schedule of mutators and seed generator

HyPFuzz: Framework

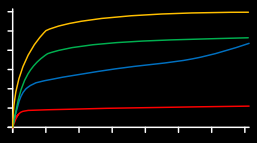


Chen, C., et al., **HyPFuzz: Formal-Assisted Processor Fuzzing**, in USENIX Security, 2023

HyPFuzz Impact

- Existing 11 vulnerabilities 3.06 × less time
- Three new vulnerabilities, two CVEs: CVE-2022-33021, CVE-2022-33023
- Coverage achievement: 41.24 ×, 6.84%
- Compatible to different coverage metrics

CVE



RISC-V

Hardware Fuzzers Impact

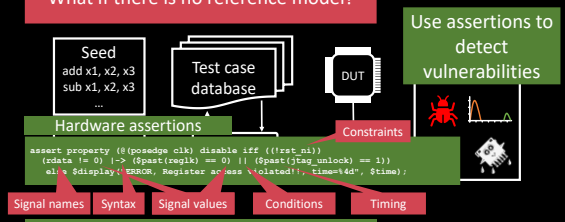
PSOFuzz	*MABFuzz*	*ChatFuzz*

• Vulnerability detection: 4.23 × faster
• Coverage achievement: 2.22 × faster
	• Vulnerability detection: 56.49 × faster
	• Coverage achievement: 3.05 × faster
		• Vulnerability detection:
		  • 4.23 × faster
		  • Two new vulnerabilities
		• Coverage achievement: 30.00 × faster

WhisperFuzz	*IFTFuzz*
• 12 new timing vulnerabilities • Eight violates zero latency requirement [1], considered as serious security vulnerabilities	• Spectre, (m)WAIT, Zenbleed • Not limited to timing side-channels

LLMs for Assertion Generation - Motivation

What if there is no reference model?

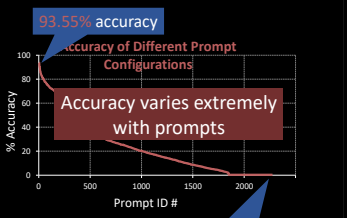


Use LLMs to automatically generate assertions!

LLMs for Assertion Generation

LLM evaluation framework

10 Benchmarks ⟷ HW CWEs
7 Prompt Evaluation Metrics
2K+ Prompt Configurations
200K+ Assertions Generated



Kande, R., et al., **(Security) Assertions by Large Language Models**, in IEEE Transactions on Information Forensics and Security, 2024

Hardware Fuzzing Impact

- **14** new functional vulnerabilities
- **12** new microarchitectural vulnerabilities
- **7** new CVEs:

CVE-2021-40506	CVE-2021-40507	CVE-2021-41612
CVE-2021-41613	CVE-2021-41614	CVE-2022-33021
CVE-2022-33023		

Publications

- **Kande, R.**, Crump, A., Persyn, G., Jauernig, P., Sadeghi, A.R., Tyagi, A. and Rajendran, J., "TheHuzz: instruction fuzzing of processors using Golden-Reference models for finding Software-Exploitable vulnerabilities," *USENIX Security Symposium*, pp. 3219-3236, 2022.
- Chen, C., **Kande, R.,** Nguyen, N., Andersen, F., Tyagi, A., Sadeghi, A.R. and Rajendran, J., "HyPFuzz: Formal-Assisted Processor Fuzzing," *USENIX Security Symposium*, pp. 1361-1378, 2023.
- **Kande, R.,** Pearce, H., Tan, B., Dolan-Gavitt, B., Thakur, S., Karri, R. and Rajendran, J., "(Security) Assertions by Large Language Models," *IEEE Transactions on Information Forensics and Security*, 2024.
- Gohil, V.*, **Kande, R.*,** Chen, C., Sadeghi, A.R. and Rajendran, J., "MABFuzz: Multi-armed bandit algorithms for fuzzing processors," *IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1-6, 2024.
- Rostami, M., Chilese, M., Zeitouni, S., **Kande, R.,** Rajendran, J. and Sadeghi, A.R., "Beyond random inputs: A novel ml-based hardware fuzzing," *IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1-6, 2024.
- Borkar, P., Chen, C., Rostami, M., Singh, N., **Kande, R.,** Sadeghi, A.R., Rebeiro, C. and Rajendran, J., "Whisperfuzz: White-box fuzzing for detecting and locating timing vulnerabilities in processors," *USENIX Security Symposium*, 2024.
- Chen, C., Gohil, V., **Kande, R.,** Sadeghi, A.R. and Rajendran, J., "PSOFuzz: Fuzzing processors with particle swarm optimization," *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1-9, 2023.
- Rostami, M., Zeitouni, S., **Kande, R.,** Chen, C., Mahmoody, P., Rajendran, J. and Sadeghi, A.R., "Lost and Found in Speculation: Hybrid Speculative Vulnerability Detection," *ACM/IEEE Design Automation Conference (DAC)*, 2024.
- Sadeghi, A.R., Rajendran, J. and **Kande, R.,** "Organizing the world's largest hardware security competition: challenges, opportunities, and lessons learned," *Great Lakes Symposium on VLSI*, pp. 95-100, 2021.
- Chen, C.*, **Kande, R.***, Mahmoody, P.*, Sadeghi, A.R. and Rajendran, J., "Trusting the trust anchor: towards detecting cross-layer vulnerabilities with hardware fuzzing.," *59th ACM/IEEE Design Automation Conference*, pp. 1379-1383, 2022.

* → equal contribution

Thank You!

Rahul Kande
rahulkande@tamu.edu

https://www.rahulkande.com

https://seth.engr.tamu.edu

Notes

Ishaq Unwala

University of Houston Clear Lake

Associate Professor of Computer Engineering

Verification education: Opportunities and Challenges

User Paper

Abstract

Functional verification is facing a challenging task to verify ever more complex designs. Finding well educated, experienced verification engineers is always a challenging task. This task is made more challenging by the fact that colleges don't provide proper education on verification techniques.

To meet this challenge, there is a need to promote functional verification in colleges and present it as a career path to the students. This requires cooperation by the faculty members and support from the industry. Currently, colleges don't promote functional verification as a subject. The students are not even aware of the career opportunities in verification. Graduating Computer Engineers learn verification techniques incidentally as part of logic design class. This is not sufficient for verification of today's designs.

The opportunity for the industry is the ability to hire a well-educated functional verification engineers, instead of having to train them on-job. Cooperation between industry and colleges can meet the future verification challenges. Despite many commonalities, there are significant differences between the education of functional verification and design engineers. The presentation will also point out the similarities and differences in education requirements.

Biography

Dr. Ishaq Unwala is an Associate Professor of Computer Engineering at University of Houston Clear Lake. He received his undergraduate degree in Electrical Engineering from West Virginia University, Morgantown, WV. He received his M.S. and Ph.D. degrees in electrical and computer engineering from the University of Texas at Austin, Austin, TX, in 1986 and 1998, respectively.

University of Houston Clear Lake

After working in semiconductor industry for 20 years for VLSI Technology Inc, IBM Corp, Intel Corp, Intrinsity Inc, Apple and Oracle (SunMicro Systems), he joined University of Houston Clear Lake in 2014. At University of Houston Clear Lake, he teaches both undergraduate and graduate courses, and conducts research in various computer engineering topics. He has co-authored over 20 technical papers and articles. He has directly supervised 5 Thesis students and taught over 300 graduate students.

His professional involvement includes serving as Section Chair of IEEE Galveston Bay Section and Vice Chair of IEEE Vehicle Technology Society in Galveston Bay Section. He is the recipient of IEEE R5 Outstanding Individual Achievement Award 2019. Under his guidance IEEE Galveston Bay Section received multiple awards: IEEE Galveston Bay Section received the 2019 R5 Best Small Section award, two IEEE R5 NASA/JSC Stepp Stone awards 2019 and global 2022 IEEE MGA Outstanding Small Section award.

His research interests include Cyber-physical systems (Internet-of-Things), Computer architecture and micro-architecture, Digital systems design, Hardware design verification techniques and technologies, Formal verification methods, Functional coverage, Engineering CAD tools technologies and techniques.

Verification education: Opportunities and Challenges

Ishaq Unwala, Ph. D.
Associate Professor of Computer Engineering
University Of Houston Clear Lake

Introduction

Professional Education:

- Ph.D. University of Texas at Austin
- M.S. University of Texas at Austin
- B.S.E.E. West Virginia University

Introduction

Work experience:

- Oracle Corporation (SPARC CPUs)
- Apple. (ARM CPUs)
- Intrinsity Inc. (ARM CPUs)
- Intel Corporation (x86 CPUs)
- IBM Corporation (POWER4 CPU)
- VLSI Technology Inc. (EDA software development)
- Computer Services (Hardware and Software services)

Introduction

Joined University of Houston Clear Lake in Fall 2014

- Current classes in Computer Engineering

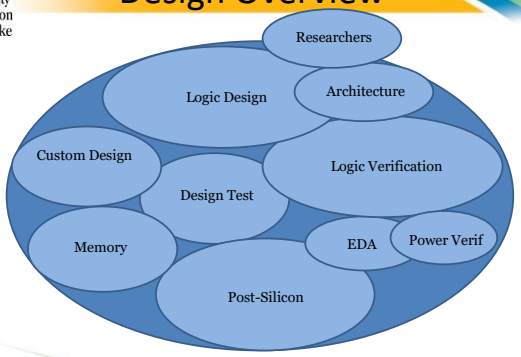
Verification of Digital Systems	Digital Systems Testing
Low Power Systems Design	Fault Tolerant Computing
Research Project and Seminar	Microprocessor Programming
Microprocessor Interfacing	Labs for classes

- Research interest:
  - Hardware verification
  - Computer architecture/micro-architecture
  - Digital Systems Design
  - Engineering CAD tools

Verification

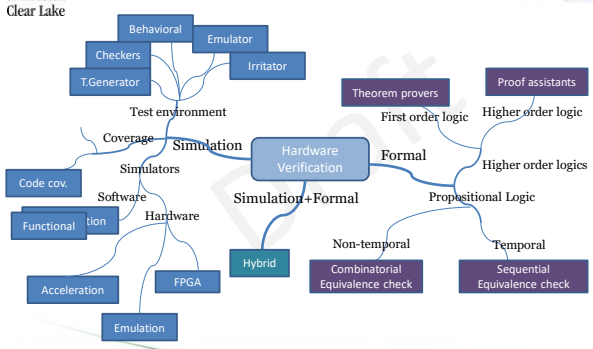
- Verification
  - Evaluation that a design meets its specification
- Today we will discuss
  - Only functional verification
- We will not discuss
  - Timing verification
  - Power verification
  - Performance verification
  - Post-silicon verification

Design Overview

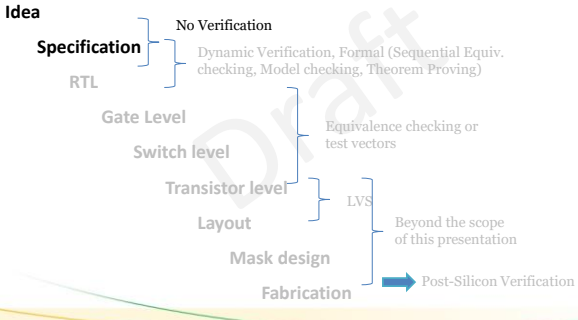


Digital Integrated Chip Design

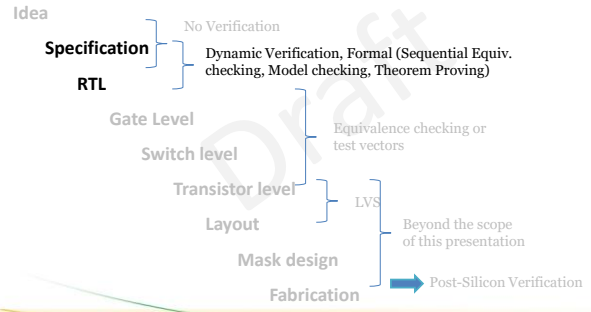
Verification Overview



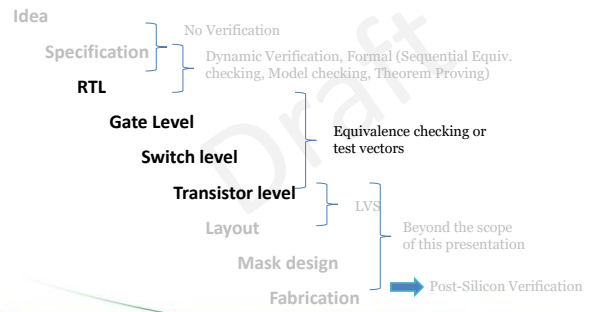
Verification



Verification



Verification



Education

- Computer Engineering basic education
  - Circuit Network concepts with labs
  - Electronic components with labs
  - Basic Digital logic with labs
  - Programming in assembly language
  - Programming in C or C++
  - Introduction to Computer Architecture
  - Verilog or SystemVerilog (or VHDL)

Design vs. Verification

- Design Engineer Specific Skills
  - Timing – design meets timing requirements
  - Efficient implementation of algorithm
  - Area – design fits in allocated area
  - Power – design stays in power envelope
  - Transistor sizes – for driving long wires
  - Routing wire congestion

Design vs. Verification

- Verification Engineer Specific Skills
  - Architectural knowledge
  - Functional understanding of the design
  - Debugging skills to find bugs
  - Effective and efficient software design
  - Coverage of events
  - Verification tools, simulation and formal
  - Test generation to exercise target logic

Verification challenge

Design Challenges

- Increasingly complex micro-architectural designs
- Increasing of functional integration (SOCs)
- Higher performance and lower power requirements
- Shorter time to market
- Practical size of verification team
- Simulation run times increasing due to
  - increase in design size
  - Larger and more complex verification environment
- Cost of “bug” in terms of lives, cost of recall, reputation, product liability, even survival of the company

Verification challenge

Work force challenge

- Colleges don't promote verification as an career
- Few college graduates with verification knowledge
- Shortage of well educated and skilled verification engineers
- Most student are not even aware of verification as a career
- Graduating students learn verification incidentally, rather than as a subject. (Insufficient to verify current designs)

Verification Opportunity

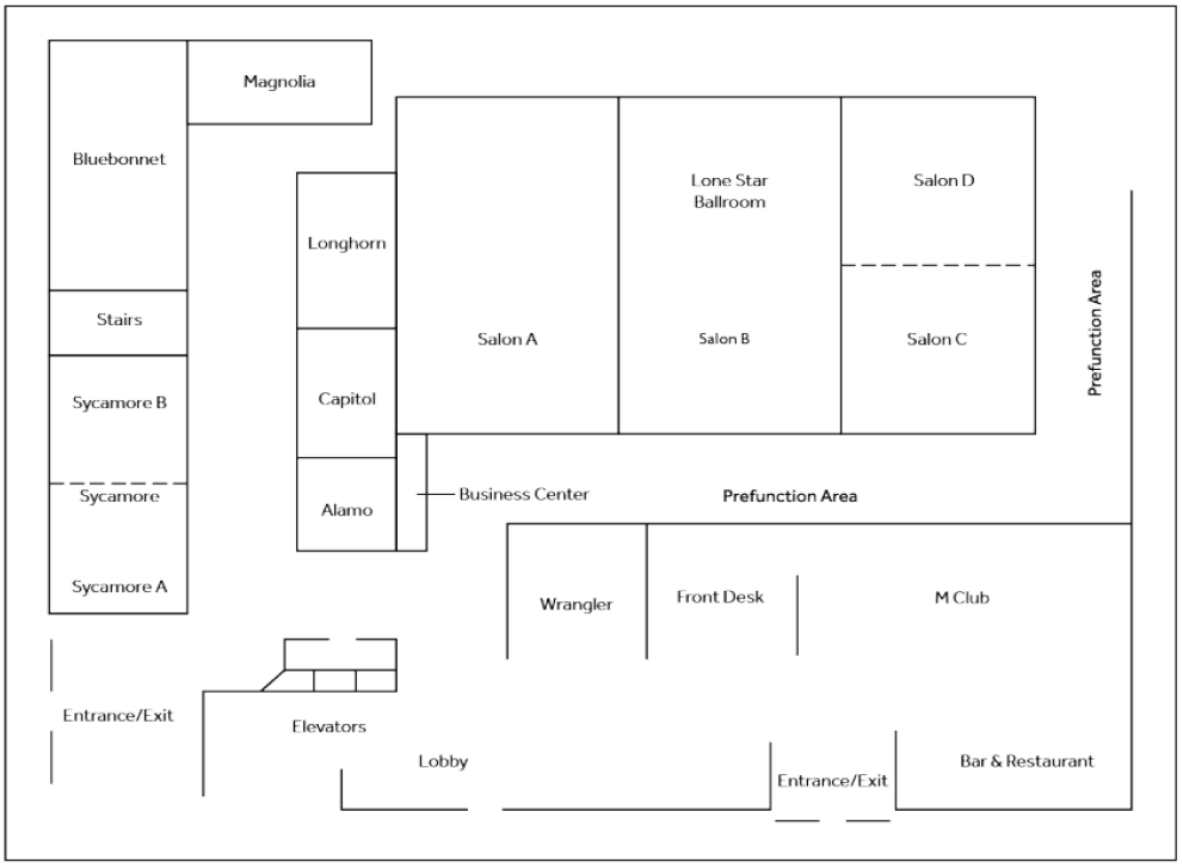
Opportunity for industry

- Academia can not solve all the verification challenges
  - But academia can provide well educated verification engineers
- Industry can get well educated verification engineers from colleges, rather that train them on job
- Industry and academia should work together to promote verification as a viable career
- Industry can provide
  - Funding for research in verification related topics (Helps to train verification engineers)
  - Internships for students to get familiar with verification

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.

Notes

Doug Smith

Doulos
Engineer / Instructor

Practical Hacks for SystemVerilog Coverage

Gold Sponsor

Abstract
Ever wondered how to reuse coverpoints across several covergroups? Or how to collect coverage in different hierarchies? SystemVerilog coverage gets the job done, but it still has some missing, unexpected, and even head scratching behaviors. This tutorial will discuss several of these issues and offer some tips and hacks to work around them.

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.

DOULOS KnowHow WEBINARS

Delivering KnowHow www.doulos.com

Practical Hacks for SystemVerilog Coverage

Presenter: Doug Smith
Engineer / Instructor

Webinar partner:

Practical Hacks for SystemVerilog Coverage

- Types of SystemVerilog Coverage
- Coverage Hacks
- Summary

Types of SystemVerilog Coverage

Cover properties …

- Use SVA temporal syntax
- Can use the same properties that assertions use
- Accessible through the assertion API (IEEE 1800-2023, Section 39)
- Placeable in structural code only

Covergroups …

- Record values of coverpoints
- Provide functional crosses of coverpoints
- Accessible by SV code and testcases
- Placeable in both objects and structural code

Assertion Coverage / Code coverage API

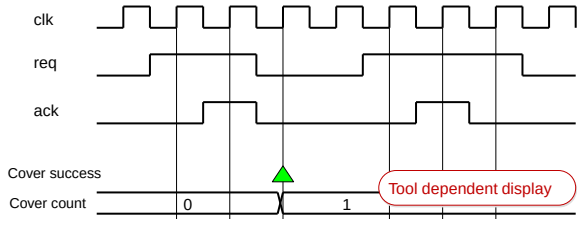
- Only assertion coverage supported by tools (IEEE 1800-2023, Sec. 39)

Cover Property

Simulators count the number of time the property holds

Display information in waveforms and in a report

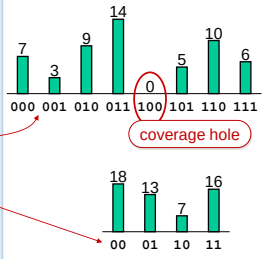
```
cover property ( @(posedge clk)
  $rose(req) |=> ((req && ack)[*0:$] ##1 !req) );
```



Covergroups

Plan for all the interesting cases, then count them during simulation

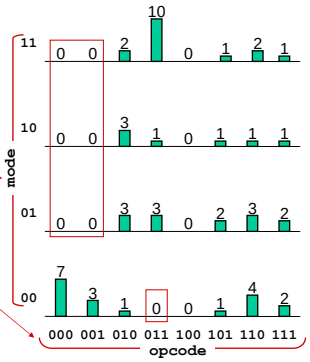
```
module InstructionMonitor (
  input bit clk, decode,
  input logic [2:0] opcode,
  input logic [1:0] mode );

  covergroup cg
      @(posedge clk iff decode);
    coverpoint opcode;
    coverpoint mode;
  endgroup

  cg cg_Inst = new;

  ...
endmodule: InstructionMonitor
```



coverage counts 2-state values

Cross Coverage

```
covergroup cg ...
  coverpoint opcode;
  coverpoint mode;
  cross opcode, mode;
endgroup
```



Count all *combinations* of crossed coverpoints.

What's Wrong with SV Coverage?

- Limitations
  - Scope – location and sampling of non-local signals
  - Features – combining coverage, extending coverage, etc.
- Cross coverage
  - Cannot cross across covergroups
  - Non-intuitive filtering
- Unexpected behaviors
  - Special keyword behavior
- Neglected by the SV Committee
  - No significant changes since 2005 until latest 1800-2023 LRM

Practical Hacks for SystemVerilog Coverage

Types of SystemVerilog Coverage

➡ Coverage Hacks

Summary

Practical Hacks for SystemVerilog Coverage

Hack #1 – Use Coverage Options Wisely

Per Instance Options

```
covergroup cg @(posedge clk);
  option.per_instance = 1;     // Turns on these options
  option.weight = 5;     // Weight in coverage calculation
  option.goal = 90;      // Percentage of at_least value
  option.at_least = 10; // Number of occurrences to reach
  option.comment = comment;

  a: coverpoint a { option.auto_bin_max = 128; };
  b: coverpoint b { option.weight = 50; };
endgroup
```

These options require per_instance = 1

```
cg g1 = new;
g1.option.goal = 100;
g1.a.option.weight = 80;
```

Instance can be used

Arguments

```
covergroup cg (ref int v, input string comment);
  coverpoint v;

  option.per_instance = 1;
  option.weight = 5;
  option.goal = 90;
  option.comment = comment;
endgroup

int a, b;

cg cg_inst1 = new(a, "This is cg_inst1 - variable a");
cg cg_inst2 = new(b, "This is cg_inst2 - variable b");
```

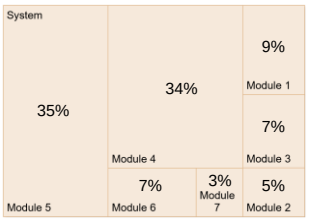
Same definition - multiple uses

Watch Your Weights

```
covergroup cg1 @(...);
  option.weight = 9;
  ...
```

```
covergroup cg2 @(...);
  option.weight = 5;
  ...
```

```
covergroup cg3 @(...);
  // No weight
  ...
```



Coverage % = ((cg1 x 9) + (cg2 x 5) + (cg3 x 1) + (cg4 x 34) + …) / ∑weights

Default

Once weighted, all items must be weighted!!

Type Options

Type options apply to the entire covergroup type

```
covergroup cg @(posedge clk);
  type_option.weight = 5;  // Weight in calculation
  type_option.goal = 90;   // Percentage of coverage
  type_option.strobe = 1;  // Sample in postponed region
  cp_a: coverpoint a {
   type_option.comment = comment;
  };
  coverpoint b;
endgroup
```

```
cg::type_option.goal = 100;
cg::cp_a::type_option.weight = 80;
```

Requires constant expressions

For Real! – real_interval

NEW! 1800-2023

```
real a, b;
covergroup cg @(posedge clk);
  // Real number range interval
  type_option.real_interval = 0.01;

  bins b1[] = {[0.75:0.85]};                // 10 bins
  bins b2[] = {[0.75:0.85],[0.90:0.92]};    // 12 bins

  // Tolerance range intervals
  bins b3[] = {[0.753+/-0.01]};   // 2 bins:
                                  //  b3[0.743:0.753]
                                  //  b3[0.753:0.763]
endgroup
```


Practical Hacks for SystemVerilog Coverage

Hack #2 – How to Cover Non-Local Signals

Pulling Values Into Covergroups

Coverpoints must be local scope

How do you pull in out-of-block signals?

How do you cross between covergroups?

```
module InstructionMonitor (
  input bit clk, decode,
  input logic [2:0] opcode,
  input logic [1:0] mode );

  covergroup cg
      @(posedge clk iff decode);
    coverpoint opcode;
    coverpoint mode;
  endgroup

  ...
```

Use Covergroup Arguments

```
covergroup cg;
  coverpoint testbench.covunit.a;
  coverpoint $root.test.count;
  coverpoint testbench.covunit.cg_inst.cp_a;
endgroup
```

Coverpoint refs not allowed

```
covergroup cg (ref logic [7:0] a, ref int b);
  coverpoint a;
  coverpoint b;
endgroup

cg cg_inst = new(testbench.covunit.a, $root.test.count);
```

Equivalent types required

Add a Sample() Method

```
class reg_tx_address extends uvm_reg;
  ...
  covergroup reg_tx_addr_cg with function
                    sample(uvm_reg_data_t data, bit dir );
     coverpoint data;
     coverpoint dir {
         bins RD = {1};
         bins WR = {0};
     }
  endgroup
```

```
class reg_tx_address extends uvm_reg;
  ...
  virtual function void sample(uvm_reg_data_t  data,
                               uvm_reg_data_t  byte_en,
                               bit             is_read,
                               uvm_reg_map     map);
    if (get_coverage(UVM_CVR_REG_BITS)) begin
       reg_tx_addr_cg.sample(data, is_read);
    end
  endfunction
endclass
```


Extend Covergroups

NEW!
1800-2023

```
class base;
  enum {red, green, blue} color;
  covergroup g1 (bit [3:0] a) with function sample(bit b);
     option.weight = 10;
     option.per_instance = 1;
     coverpoint a;
     coverpoint b;
     c: coverpoint color;
  endgroup
  function new();
     g1 = new;
  endfunction
endclass
```

```
class derived extends base;
   bit d;
   covergroup extends g1;
      option.weight = 1;       // Overrides the weight from base g1
                               // Uses per_instance=1 from base g1
      c: coverpoint color {    // Overrides the c coverpoint in base g1
         ignore_bins ignore = {blue}; }
      coverpoint d;            // Adds new coverpoint
      cross a, d;          // Crosses new coverpoint with inherited one
   endgroup :g1
endclass
```

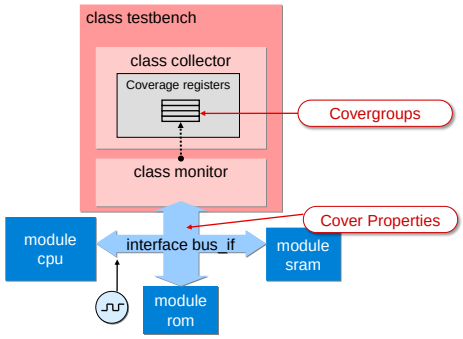
Cross with other covergroups

Practical Hacks for SystemVerilog Coverage

Hack #3 – How to Mix SVA Coverage

with Covergroups

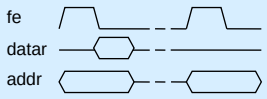
Example



Cover Property

- Uses sequence/property in interface

```
sequence cond_jump;
  logic [3:0]     opcode;
  logic [ww-1:0] jump_instr_pc;
  logic [ww-1:0] jump_target_pc;

  @(posedge bus_if.clk)
    first_match(
      fe
      ##1
      ((datar[15:12] == jzero_op ||
        datar[15:12] == jneg_op), opcode = datar[15:12],
                                  jump_instr_pc = addr)
      ##[1:$]
      fe
      ##1
      (1, jump_target_pc = addr,
        cover_jump(opcode, jump_instr_pc, jump_target_pc)
    ));

endsequence: cond_jump
cover property (cond_jump);
```



Note: sample local variables

Transfer Coverage to Monitor

```
typedef struct packed  {
  logic [3:0]     opcode;
  logic [ww-1:0] jump_instr_pc,
  logic [ww-1:0] jump_target_pc;
} jump_data_t;
```

```
interface basic_bus_if;
 jump_data_t jump_data;
 bit  jump_trig = 0;

 function void cover_jump( logic [3:0] op,
            logic [ww-1:0] jump_instr_pc, jump_target_pc );
   jump_data = {op, jump_instr_pc, jump_target_pc};
   jump_trig = ~jump_trig;
endfunction
...
```

Interface function

```
class monitor extends ...;
  task process_jump();
    forever begin
      jump_data_t t; jump_xact tr;
      @(m_bus_if.mon_xact.jump_trig)
      t = m_bus_if.mon_xact.jump_data;
      tr = new(t.opcode, t.jump_instr_pc, t.jump_target_pc);
      cov_collector.write(tr);    // Send to covergroup
    end
  endtask: process_jump
  ...
```

Bus monitor task

Record Coverage with Covergroup

```
class collector extends ...;
  ...
  virtual function void write( input jump_xact t );
    m_cov.opcode = t.opcode;
    m_cov.jump_instr_pc  = t.jump_instr_pc;
    m_cov.jump_target_pc = t.jump_target_pc;
    m_cov.jump_delta = m_cov.jump_target_pc - m_cov.jump_instr_pc - 1;
    m_cov.cov_jump.sample();
  endfunction
  ...
```

```
covergroup cov_jump;
   coverpoint jump_delta {
      bins no_jump         = { 0 };
      bins short_jump_fwd  = { [1:15] };
      bins long_jump_fwd   = { [16:2**ww-1] };
      bins short_jump_back = { [-15:-1] };
      bins long_jump_back  = { [-2**ww+1:-16] };
      option.at_least = 4;
   }
endgroup: cov_jump
```

Advantages

- Cover property ...
  - Protocol defined in the interface (everything kept together)
  - Protocol defined using temporal syntax--not a custom FSM
- Covergroup …
  - Provides additional coverage options
  - Provides cross coverage
  - Accessible by testbench or testcase (coverage feedback)

Practical Hacks for SystemVerilog Coverage

Hack #4 – How to Direct Stimulus with Coverage

Querying coverage

get_coverage() returns % covered (as a real number) on covergroups and coverpoints

```
initial
    repeat (100) @(posedge clk) begin
      cg_inst.sample;
      cov = cg_inst.get_coverage;     // Covergroup
      if ( cov > 90.0 ) cg_inst.stop;
    end
```

```
// Weight randomness to hit uncovered coverpoints
randcase
  (100 - $rtoi(cg_inst.a.get_coverage)) : ...;
  (100 - $rtoi(cg_inst.b.get_coverage)) : ...;
  (100 - $rtoi(cg_inst.c.get_coverage)) : ...;
  ...
endcase
```


Querying bin coverage

```
covergroup cg;
  coverpoint i {
    bins zero     = { 0 };
    bins tiny     = { [1:100] };
    bins hunds[3] = { 200,300,400,500,600,700,800,900 };
  }
endgroup
```

get_coverage() does not work on bins

```
cov = cg_inst.i.zero.get_coverage();
```

✗ Not allowed

Create coverpoints for each bin

```
covergroup instr_cg;
  op_nop :
  coverpoint instr_word[15:12] { bins op = { nop_op }; }

  op_load :
  coverpoint instr_word[15:12] { bins op = { load_op };}

  op_store :
  coverpoint instr_word[15:12] { bins op = { str_op }; }

  op_move :
  coverpoint instr_word[15:12] { bins op = { move_op }; }
  ...
endgroup
```

- Now `get_coverage()` can be used ...

```
cov = cg_inst.op_nop.get_coverage();
```


Randomize using dist

The dist operator accepts changing random weights

```
int    weight_nop    = 1,
   weight_load   = 1,
   weight_store  = 1,
   weight_add    = 1,
   ...;

constraint bias_opcodes {
opcode dist {
   nop_op     := weight_nop,
   load_op    := weight_load,
   store_op   := weight_store,
   add_op     := weight_add,
   ...
};
}
```

1800 LRM: **`dist`** accepts integral expressions

Some simulators only support variables

Use pre_randomize to set weights

pre_randomize() sets the weighting used by the dist

```
function int calc_weight( opcode_t op );
real cov;
case ( op )   // Grab coverage (see Trick #2)
   nop_op:   cov = covunit.cg.op_nop.get_coverage;
   load_op:  cov = covunit.cg.op_load.get_coverage;
   store_op: cov = covunit.cg.op_store.get_coverage;
   ...
endcase
calc_weight = 100 - $rtoi(cov) + 1;
endfunction : calc_weight

function void pre_randomize();// Set dist weighting
weight_nop    = calc_weight(nop_op);
weight_load   = calc_weight(load_op);
weight_store  = calc_weight(store_op);
weight_add    = calc_weight(add_op);
...
endfunction
```

Beware!! No longer truly random!

Practical Hacks for SystemVerilog Coverage

Hack #5 – How to Include Cover Properties

into Functional Coverage

Create a Coverage Counter

```
bit coverage[string];

c1: cover property ( a |=> b ) coverage["c1"]=1;
c2: cover property ( c |=> d ) coverage["c2"]=1;
```

Check tool support

```
covergroup cg @(posedge clk);
  type_option.strobe = 1;  // Sample end of cycle
  coverpoint coverage["c1"] {
     bins match = { 1 };
  }
  coverpoint coverage["c2"] {
     bins match = { 1 };
  }
endgroup
cg cg1 = new;

// Clear the coverage after it is sampled
always @(negedge clk)
  coverage = '{default:0};
```


Add Weights and Cross

```
covergroup cg @(posedge clk);
  type_option.strobe = 1;
  option.per_instance = 1;

  cp_c1: coverpoint coverage["c1"] {
    bins match = { 1 };
    option.weight = 1;
  }

  cp_c2: coverpoint coverage["c2"] {
    bins match = { 1 };
    option.weight = 0.5;
  }

  cross cp_c1, cp_c2;

endgroup
```


Practical Hacks for SystemVerilog Coverage

Hack #6 – Avoid Coverage Gotchas

illegal_bins

```
logic [2:0] opcode;
logic signed [15:0] jump_distance;

covergroup cg @(posedge clk iff decode);
  coverpoint opcode {
    bins move_op[] = { 3'b000, 3'b001 };
    bins ALU_op = {[3'b010:3'b011],[3'b101:3'b110]};
    bins jump_op = {3'b111};
    illegal_bins unused_op = {3'b100};
  }
endgroup
```

Opcode table	
000	load
001	store
010	add
011	sub
100	*unused*
101	and
110	shift
111	jump

Value not counted

Issues

illegal_bins

- excludes values from a covergroup--that's good
- throws errors--that's bad!

```
# ** Error: Illegal range bin value='b1011 got
covered. The bin counter for the bin
'\/covunit/cg_i.b.bad' is 362.
```

Questions to consider:

- Should something passive throw errors?
- If used for checking, what happens if coverage is turned off?

Better option:

- write assertions and checkers for checking
- **`ignore_bins`** for coverage

Using default bins

```
bit [15:0] i;
```

`default` catches unplanned or invalid values

```
covergroup cg @(posedge Clock);
  coverpoint i {
    bins zero     = { 0 };
    bins tiny     = { [1:100] };
    bins hunds[3] = { 200,300,400,500,600,700,800,900 };
    bins huge     = { [1000:$] };
    ignore_bins ignore = { [501:599] };
    bins others[] = default;
  }
endgroup
```

One bin for each other value

Issues (1)

```
int a;                    // 2^32 values
covergroup cg ...;
coverpoint a { bins other[] = default; }
endgroup
```

One bin for each value

- Use of default may crash your simulator:

  # ** Fatal: The number of singleton values exceeded the system limit of 2147483647 for unconstrained array bin 'other' in Coverpoint 'a' of Covergroup instance '\/covunit/cg_i'.

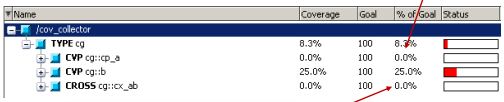
- Do you really want to look at 2147483647 bins?

Issues (2)

- default bins are not included in the coverage calculation!

```
covergroup cg @(posedge clk);
  cp_a : coverpoint a {
    bins a[4] = default;
  }
  cx_ab : cross cp_a, b;
endgroup
```

No coverage!



Therefore, no cross coverage!

Solution

Avoid [] with default, or use with smaller variables with fewer possible values

```
logic [7:0] a;                  // Fewer values
covergroup cg ...;
coverpoint a {
   bins other = default;        // One bin
}
endgroup
```

Use wildcard or min/max ($) to catch remaining values

```
bins huge     = { [1000:$] };   // Max values
wildcard bins a[4] = { 'b?0 };  // Even values
```


Practical Hacks for SystemVerilog Coverage

Summary

Hack #1 – Use Coverage Options Wisely

Hack #2 – How to Cover Non-Local Signals

Hack #3 – How to Mix SVA Coverage with Covergroups

Hack #4 – How to Direct Stimulus with Coverage

Hack #5 – How to Include Cover Properties into Functional Coverage

Hack #6 – Avoid Coverage Gotchas

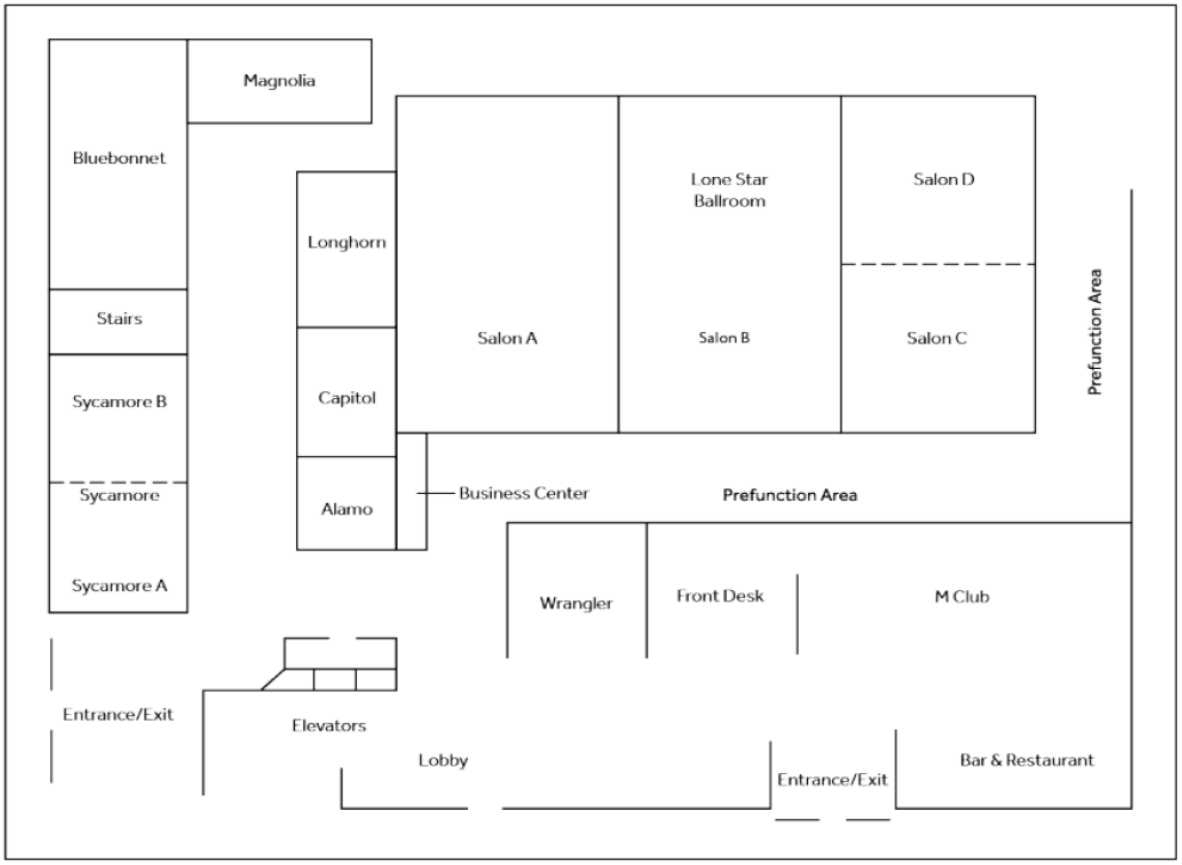
Q & A

Any questions?

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

Espen Tallaksen

EmLogic
Managing Director

A pragmatic approach to improving your FPGA VHDL verification

User Paper (Remote presentation)

Abstract
A good architecture is very important for FPGA design, but it is in fact equally important for verification of complex FPGA design. The verification architecture determines the verification efficiency and the product quality for complex designs.
The difference between a good and a "normal" verification architecture could be many hundred hours, and for medium to complex designs even as much as a couple of thousand hours. The only good thing about this - is that you can easily do something about it. UVVM was made exactly for this and is free and open source - and used by 35-40% of all FPGA VHDL designers world-wide. A new ESA (European Space Agency) project has just been initiated to extend UVVM even further.
In this presentation, we will show you how to make a simple, well-structured, and efficient testbench using the open-source Universal VHDL Verification Methodology (UVVM) architecture. We will also discuss the importance of testbench sequencer simplicity and how it can be used to control multiple VHDL Verification Components simultaneously. We will show testbench examples for a simple interrupt controller, for AXI, for Avalon and more - to illustrate how UVVM will help allow pragmatic and simple verification of both simple and complex DUTs.

Biography
Espen Tallaksen is the CEO of EmLogic, in Norway.
Espen is also the author and architect of the Open Source UVVM (Universal VHDL Verification Methodology).
He has a strong interest in methodology cultivation and pragmatic efficiency and quality improvement, and he has given lots of presentations at various international conferences with great feedback. He is also giving courses on FPGA Design and Verification.

Notes

EmLogic

A pragmatic approach to improving your FPGA VHDL verification

Verification Futures Conference, US
12 September, 2024

(by Espen Tallaksen, CEO EmLogic)

The leading FPGA design centre in Norway and Scandinavia (www.emlogic.no/leading)

EmLogic.no | The Norwegian Embedded Systems and FPGA Design Centre

---

EmLogic

- Independent Design Centre for Embedded Systems and FPGA
- Established 1st of January 2021. **Extreme ramp up**
  - January 2021: 1 person
  - September 2024: → 43 persons (SW:19, HW:4, FPGA:18, DSP:1+)
- Continues the legacy from bitvis
  - All previous Bitvis technical managers are now in EmLogic
- Verification IP and Methodology provider UVVM
- Course provider within FPGA Design and Verification
  - Accelerating FPGA Design (Architecture, Clocking, Timing, Coding, Quality, Design for Reuse, …)
  - Advanced VHDL Verification – Made simple (Modern efficient verification using UVVM)
- A potential partner for ESA projects for European companies
  - Increased opportunities due to Norway's low geo return

What is UVVM?

UVVM

UVVM = Universal VHDL Verification Methodology
- VHDL Verification Library & Methodology
- Free and Open Source

FPGA ASIC CPLD

- Very structured infrastructure and architecture
- Significantly improves Verification Efficiency
- Assures a far better Design Quality

DOULOS

- Recommended by Doulos for Testbench architecture
- ESA projects to extend the functionality

esa

- IEEE Standards Association Open source project

IEEE SA OPEN

- Runs on any VHDL-2008 compliant simulator

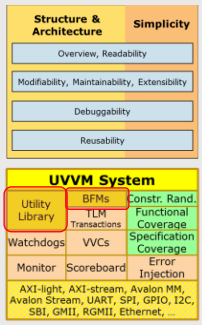
SIEMENS Mentor A Siemens Business | ALDEC | GHDL

Testbenches in the industry: Far worse than design

- Architecture is normally far worse
  - Far worse structured (architecture)
  - Far worse documented
  - Far worse coded
- We waste far more time on verification
  - Saving-potentials from 20% to 75% on average…
    - 75% means 4 times the "possible" time.
    - Average? Maybe a factor of 2 (50%)
- TB quality could have "double quality improvement"
  - The testbench in itself could be much better for finding bugs
  - Faster verification allows more verification

Applying the design solution?

Same issues:



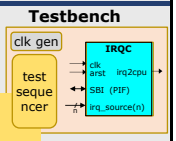
Same approach:
- Basic infrastructure
- Divide and conquer at all stages
- Prioritise the reader at all times

Same solution:
- Architecture, Architecture, Architecture
- Simplicity where needed the most

Same challenge:
Awareness, Awareness, Awareness

Example on test sequencer code and transcript/log

```
clock_generator(clk, GC_CLK_PERIOD);
```




```
log(ID_LOG_HDR, "Check Interrupt trigger clear mechanism");
check_value(irq2cpu, '0', "irq2cpu default inactive");
check_stable(irq2cpu, now - v_reset_time, "Stable irq2cpu");
gen_pulse(irq_source(3), '1', C_CLK_PERIOD, "Set IRQ 3 for clock period");
await_value(irq2cpu, '1', 0 ns, 2* C_CLK_PERIOD, "Interrupt expected");
sbi_write(C_ADDR_ITR, x"AA", "ITR : Set interrupts");
```

```
2000.0 ns    Check Interrupt trigger clear mechanism
-------------------------------------------------------------------------------
 110.0 ns    check_value() => OK, for std_logic '0'. irq2cpu default
 727.5 ns    check_stable() => OK. Stable at 0. Stable irq2cpu
1060.0 ns    Pulsed to '1'. Set IRQ source for clock period
1117.5 ns    await_value(std_logic 1, 0 ns, 20 ns) => OK. Interrupt expected
2020.0 ns    SBI write(A:x"2", x"AA") completed. ITR : Set interrupts
```

All procedures with:
- Positive acknowledge If wanted
- Alert message and mismatch report
- Alert count and ctrl

UVVM Utility Library for simple **and** advanced testbenches

- check_stable(), await_stable()
- clock_generator(), adjustable_clock_generator()
- random(), randomize()
- gen_pulse()
- block_flag(), unblock_flag(), await_unblock_flag()
- await_barrier()
- enable_log_msg(), disable_log_msg()
- to_string(), fill_string(), to_upper(), replace(), etc…
- normalize_and_check()
- set_log_file_name(), set_alert_file_name()
- wait_until_given_time_after_rising_edge()
- etc…

Basic Randomisation in "old" UVVM

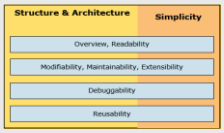
- Under UVVM Utility library : methods_pkg
- Simple functions - using shared variable seeds:
  - my_int := random(VOID);
  - **my_int := random(4, 245);**
  - my_slv8 := random(8);
  - my_byte_array := random(1, 16);
  - my_time := random(1 ns, 15 ns);
- Also provides support for fixed random sequence
  - Needs to control seeds locally
  - random(4, 245, seed1, seed2, my_int);

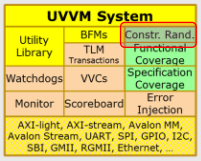
Still the simplest possible solution for simple Randomisation

UVVM Enhanced Randomisation

- Well integrated with UVVM
  - Alert handling and logging in particular
- Strong focus on Overview & Readability
  - Adding keywords to ease understanding

Quality & Efficiency enablers





Single Method approach

- **"Standard" approach: Randomisation in one single command**
  - Simple randomisation is always easy to understand

```
addr <= my_addr.rand(0, 18);
```

  - More complex randomisation is normally more difficult to understand
    BUT – there are ways to significantly improve this

```
addr <= my_addr.rand(0, 18, EXCL, (7));
addr <= my_addr.rand(0, 18, ADD, (30,31));
addr <= my_addr.rand(0, 18, ADD, (30,31), EXCL, (7));
```

  - S

But – what does this mean???

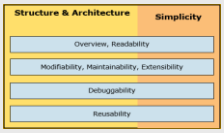
```
addr <= my_addr.rand(0, 18, (30,31), (7));
```

```
addr <= my_addr.rand_range_weight((0,18,4),(19,31,1));
```


Coding for readability

- Well integrated with UVVM
  - Alert handling and logging in particular
- Strong focus on Overview & Readability
  - Adding keywords to ease understanding
- Easy to Maintain and Extend

Quality & Efficiency enablers



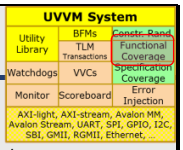
Typing code consumes is an insignificant part of the development time.

Reading and understanding code is repeated over and over again, and is thus a significant part of the development time

```
addr <= my_addr.rand(0, 18, ADD, (30,31), EXCL, (7));
```

➔ **Investing in better code yields a huge return on investment**

Functional Coverage



- Functional Coverage is
  'a user-defined metric that measures how much of the design specification has been exercised in verification'
  - Has various functional scenarios been tested.
  - **A manual process is required to set up all wanted scenarios**

➔ **Time consuming, but**
Great to check that various specific scenarios have been verified

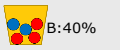
E.g. For a UART
- Has payload started and ended with both 0 and 1 (4 permutations)
- Has odd and even parity been tested
  - AND with the payload variation above
- Has selected cases of baud rate vs clock rate been tested

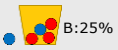
Some FC reports – out of many

```
# UVVM:  ================================================================================================
# UVVM:  0 ns *** COVERAGE SUMMARY REPORT (NON VERBOSE): TB seq. ***
# UVVM:  ================================================================================================
# UVVM:  Coverpoint:                Covpt_1
# UVVM:  Coverage (for goal 100): Bins: 60.00%,   Hits: 76.47%
# UVVM:  ------------------------------------------------------------------------------------------------
# UVVM:              BINS          HITS     MIN HITS    HIT COVERAGE          NAME          ILLEGAL/IGNORE
# UVVM:          (256 to 511)         1          N/A          N/A        illegal_addr          ILLEGAL
# UVVM:           (0 to 125)          6          8          75.00%       mem_addr_low             -
# UVVM:        (126, 127, 128)        3          1         100.00%       mem_addr_mid             -
# UVVM:         (129 to 255)         14          4         100.00%       mem_addr_high            -
# UVVM:         (0->1->2->3)          0          2           0.00%       transition_1             -
# UVVM:          transition_2         2          2         100.00%       transition_2             -
# UVVM:  ------------------------------------------------------------------------------------------------
# UVVM:  transition_2: (0->15->127->248->249->250->251->252->253->254)
# UVVM:  ================================================================================================

# UVVM:  ================================================================================================
# UVVM:  0 ns *** OVERALL COVERAGE REPORT (VERBOSE): TB seq. ***
# UVVM:  Coverage (for goal 100): Covpts: 50.00%,   Bins: 73.68%,   Hits: 76.00%
# UVVM:  ================================================================================================
# UVVM:     COVERPOINT     COVERAGE WEIGHT    COVERED BINS     COVERAGE(BINS|HITS)    GOAL(BINS|HITS)    % OF GOAL(BINS|HITS)
# UVVM:  ------------------------------------------------------------------------------------------------
# UVVM:      Covpt_1             1              3 / 5          60.00% | 76.47%       50% | 100%         100.00% | 76.47%
# UVVM:      Covpt_2             1              3 / 3         100.00% | 100.00%     100% | 100%         100.00% | 100.00%
# UVVM:      Covpt_3             1              6 / 6         100.00% | 100.00%     100% | 100%         100.00% | 100.00%
# UVVM:      Covpt_4             1              0 / 4           0.00% | 0.00%       100% | 100%           0.00% | 0.00%
# UVVM:      Covpt_5             1              0 / 1           0.00% | 0.00%       100% | 100%           0.00% | 0.00%
# UVVM:      Covpt_6             1              4 / 4         100.00% | 100.00%     100% | 100%         100.00% | 100.00%
# UVVM:      Covpt_7             1              0 / 3           0.00% | 0.00%       100% | 100%           0.00% | 0.00%
# UVVM:      Covpt_8             1             12 / 12        100.00% | 100.00%     100% | 100%         100.00% | 100.00%
# UVVM:  ================================================================================================
```

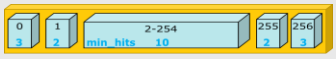

Optimised randomisation

- Optimised Randomisation is
  - Randomisation without replacement
  - Weighted according to target distribution AND previous events.





Thus for the protocol payload of 0-256 bytes:



→ Optimised randomisation yields the lowest number of randomisations for a given target

→ Major reduction in # packets and thus simulation time

Again: Imagine a payload between 0 to 65.536 bytes…

Specification Coverage

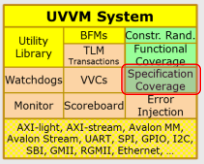
- Assure that all requirements have been verified
  1. Specify all requirements
  2. Report coverage from test sequencer(s) (or other TB parts)
  3. Generate summary report
     - Coverage per requirement
     - Test cases covering each requirement
     - Requirements covered by each Test case
     - Accumulate over multiple Test cases

Requirements Traceability Matrix

Mandatory for Safety and Mission Critical
(Strictly required by ESA)
Strongly recommended for good quality assurance
Expensive tools exist…

Solutions exist to report that a testcase finished successfully

BUT - reporting that a testcase has finished is not sufficient



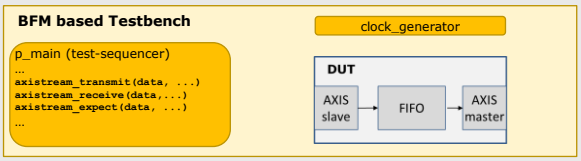
Specification Coverage (2)

Requirement Label	Description
MOTOR_R1	The acceleration shall be ***
MOTOR_R2	The top speed shall be given by ***
MOTOR_R3	The deceleration shall be ***
MOTOR_R4	The final position shall be ***

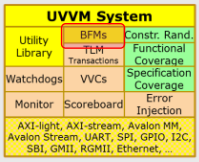
- What if multiple requirements are covered by the same testcase?
  - E.g. Moving/turning something to a to a given position
    R1: Acceleration R2: Speed R3: Deceleration 4: Position etc..

- Generates various types of reports
  - Coverage per requirement
  - Test cases covering each requirement
  - Requirements covered by each Test case
- Accumulated over multiple Test cases

AXI-stream - BFM based TB



- No test harness (for simplicity)
- Sequencer has direct access to DUT signals
  - Thus BFMs from p_main can also see the DUT signals
- BFMs are sequential procedures running in sequence in p_main.



Resulting transcript +Debug

Note: Removed Prefix and Scope to show on a single line.

```
axistream_transmit(v_byte_array, msg, clk, m_axis);
  ID_BFM              106.0 ns  axistream_transmit(3B)=> Tx DONE.
axistream_expect(v_exp_array(0 to 2), "", clk, s_axis);
  ID_BFM              122.0 ns  axistream_expect(3B)=> OK, received 3B.
```

May add more info for debugging

```
enable_log_msg(ID_PACKET_INITIATE);        enable_log_msg(ID_PACKET_DATA);

ID_PACKET_INITIATE      52.0 ns  axistream_transmit(3B)=>
ID_PACKET_DATA          52.0 ns  axistream_transmit(3B)=> Tx x"00", byte# 0.
ID_PACKET_DATA          68.0 ns  axistream_transmit(3B)=> Tx x"01", byte# 1.
ID_PACKET_DATA          82.0 ns  axistream_transmit(3B)=> Tx x"02", byte# 2.
ID_PACKET_COMPLETE     106.0 ns  axistream_transmit(3B)=> Tx DONE.
```

May add similar debugging info for data reception

EmLogic

Advanced BFM usage - in simple TB

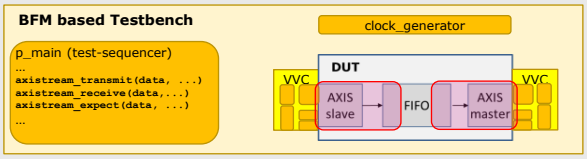
- May utilise more of the protocol: `tkeep, tuser, tlast, tstrb, tid, tdest`
- May define different widths
- May configure behaviour:
  - Set maximum wait cycles
  - May set to match data exact or std_match
  - May set byte endianness (for SLV larger than data width)
  - May set to de-assert tvalid some cycles (randomly or fixed)
  - May set to de-assert tready some cycles (randomly or fixed)
  - And more…

Have enabled lots of bug detection in users' AXI stream interfaces

`valid_low_at_word_num`	`Word index during which the Master BFM shall deassert valid while sending a packet.`
`valid_low_duration`	`Number of clock cycles to deassert valid.`
`valid_low_multiple_random_prob` `valid_low_max_random_duration`	Similar for 'ready'

EmLogic

What if we need to check the DUT for simultaneous activity?



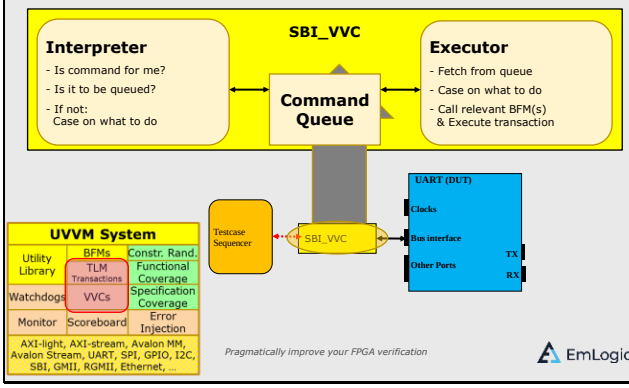
- BFMs are great for simple testbenches
  Dedicated procedures in a simple package
  Just reference and call from a process
- BUT
  A process can only do one thing at a time
  - Either execute that BFM
  - **Or** execute another BFM
  - **Or** do something else

- To do more than one thing:
  → Need multiple "threads"
  → Could use multiple processes
  Need inter process communication
  → Leads to chaos – as for design

→ Need an entity (or component)
(VC = Verification Component)
→ Need a defined protocol

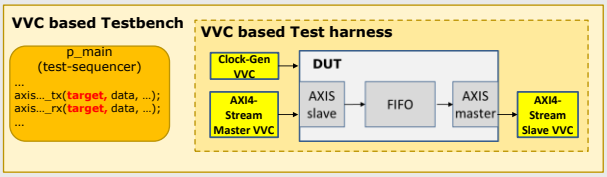
EmLogic

VVC: VHDL Verification Component

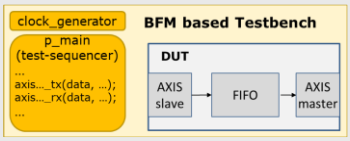


EmLogic

AXI-stream - VVC based TB (1)



axistream_transmit(**target**, data, ...);
axistream_expect(**target**, data, ...);



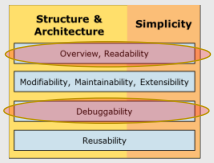
EmLogic

Debugging is equally important

According to Wilson research: On average…

- Verification is 50% of total development time
- Debug is 50% of verification. I.e. 25% of total development

Same issues:



- Mismatch reports are important
- Progress report is critical
- Detailed progress report is sometimes needed.

EmLogic

Resulting transcript +Debug

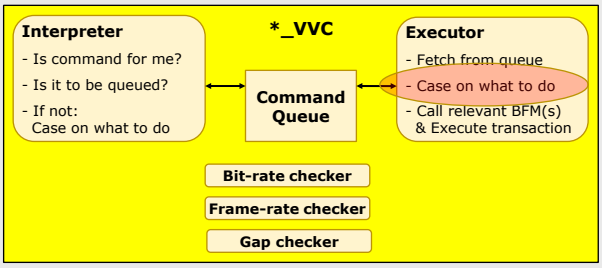
Note the changing scope

```
axistream_transmit(AXISTREAM_VVCT,0, v_data_array, msg);
ID_UVVM_SEND_CMD         50.0 ns   TB seq.(uvvm)
    ->axistream_transmit(AXISTREAM_VVC,0, 512 bytes): 'TX 512B'   [6]
ID_PACKET_DATA       24202.0 ns  AXISTREAM_VVC,0
      axistream_transmit(512B)=> Tx x"ED", byte# 493. 'TX 512B'  [6]
ID_PACKET_COMPLETE   24346.0 ns  AXISTREAM_VVC,0
      axistream_transmit(512B)=> Tx DONE. 'TX 512B'  [6]

axistream_expect(AXISTREAM_VVCT,1, v_exp_array, "Expecting **** ");
ID_UVVM_SEND_CMD       50.0 ns   TB seq.(uvvm)
    ->axistream_expect_bytes(AXISTREAM_VVC,1, 512b): 'Expecting 512b'  [7]
-  Plus similar additional verbosity as for Transmit
-  Plus for both: Debug-messages when command reaches Interpreter and Executor
```


EmLogic

VVC: Easy to extend & modify

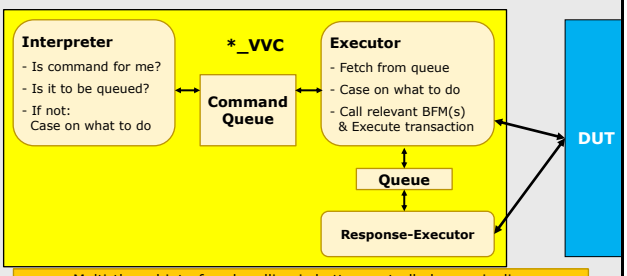
- Easy to add local sequencers
- Easy to add checkers/monitors/etc



Checkers are better included as parallel processes.
VVC architecture allows simple inclusion and control

VVCs: Extended

- Easy to handle split transactions
- Easy to handle out of order execution



Multi-thread interface handling is better controlled as a pipeline
VVC architecture allows simple inclusion – with queuing and control
→ Allows overview to be kept – for something that normally creates chaos…

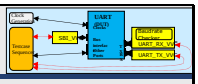
VVC Advantages

Due to:
- VVC architecture
- TB architecture
- Command structure

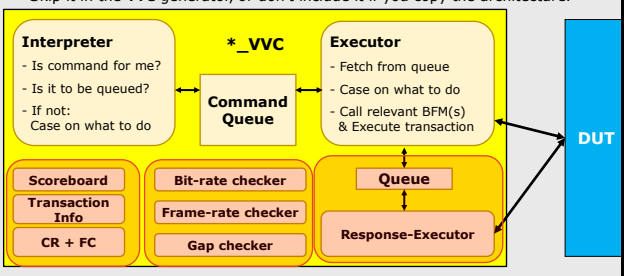
- Simultaneous activity on multiple interfaces
- Encapsulated → Reuse at all levels
- Queue → May initiate multiple high level commands
- Local Sequencers for predefined higher level commands
- **Only in UVVM VVCs:**
  - UNIQUE: Control all VVCs from a single sequencer!
  - May insert delay between commands – from sequencer
    → The only system to target cycle related corner cases
  - Simple handling of split transactions and out of order protocols
  - Common commands to control VVC behaviour
  - Simple synchronization of interface actions – from sequencer
  - May use Broadcast and Multicast

Better Overview, Maintenance, Extensibility and Reuse

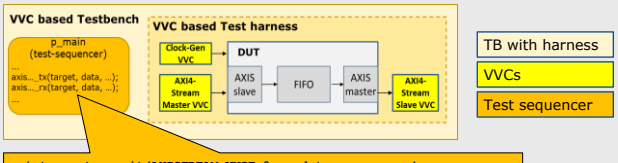
VVCs: Too advanced???



Advanced functionality is great when needed, but what if not???
- If using an existing VVC, just ignore it. Use it out of the box – without the extras.
- If making your own VVC, don't include the advanced stuff
  Skip it in the VVC generator, or don't include it if you copy the architecture.



Simplicity where needed the most



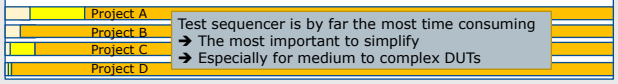
TB with harness

VVCs

Test sequencer

```
axistream_transmit(AXISTREAM_VVCT,0, v_data_array, msg);
axistream_expect(AXISTREAM_VVCT,1, v_exp_array, "Expecting **** ");
```

Totalt Workload percentages for various scenarios

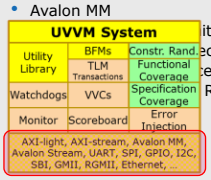


Test sequencer is by far the most time consuming
➔ The most important to simplify
➔ Especially for medium to complex DUTs

Not investing in a good architecture ➔ Test sequencer time may increase a lot

Lots of free UVVM BFMs and VVCs

- AXI4-lite
- AXI4 Full
- AXI-Stream Transmit and Receive
- UART Transmit and Receive
- SBI
- SPI Transmit and Receive
- I2C Transmit and Receive
- GPIO
- Avalon MM



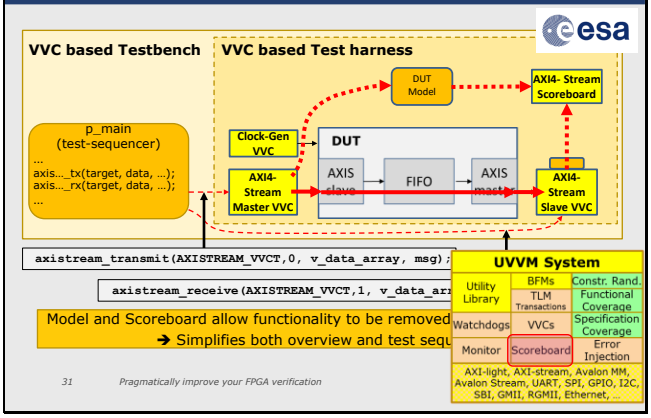
All:
- Free
- Open Source
- Well documented
- Example Testbenches

The largest collection of VHDL Interface Models

VVC: VHDL Verif. Comps.
- Includes the corresponding BFM

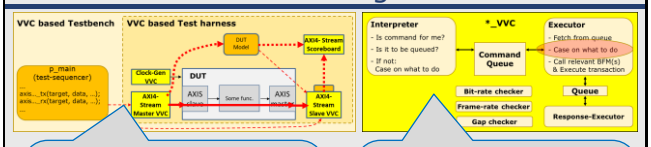
Allows:
- Simultaneous interface handling
- Synchronization of interfaces
- Skewing between interfaces
- Additional protocol checkers
- Local sequencers
- Activity detection
- Simple reuse between projects

Advanced scoreboard-based TB



Model and Scoreboard allow functionality to be removed
➔ Simplifies both overview and test sequ

A good architecture allows better understanding at all levels



- Standard Interface
- Standard Protocol
- Standard common commands
- Standard Status interface
- Standard Config interface
- Standard handling of multiple VVCs
- Standard VVC synchronization
- Standard multicast/broadcast

- Standard VVC internal architecture
- Standard VVC control of checkers
- Standard queuing system
- Standard handling of multi-threaded interfaces
- Standard debug support

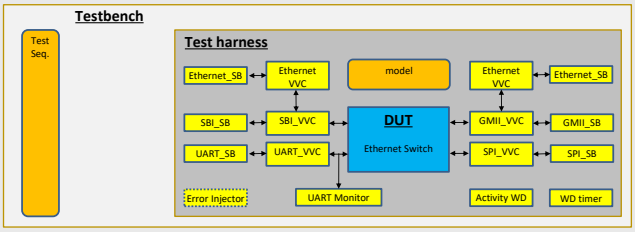
Simplification

VVCs from different users will work together

Users know how VVCs behave and how any test harness will work

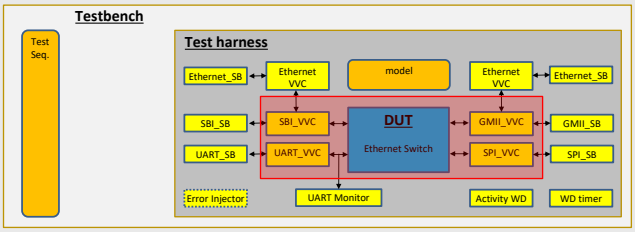
The full picture (1)

Is this too much? Too many VIP components? Too much structure?



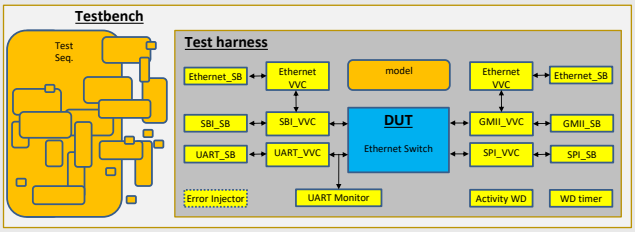
The full picture (2)

VVCs are needed to check multiple interfaces



The full picture (3)

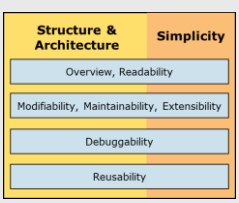
Removing VIPs ➔
Putting What if we simply write a sequencer
- only very unstructured, less overview, not maintainable, not very reusable



Debugging is much easier with a good architecture
(25% of development, acc. to Wilson Research 2020 survey)

Improving verification in general?

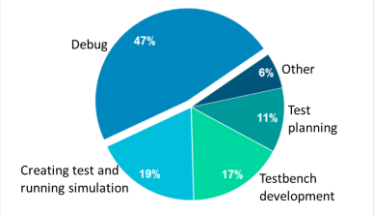
- Awareness, awareness, awareness
- Allow time up front to structure
- Discuss your testbench architecture before coding
  - Sparring partner
  - Walkthrough
- Same applies to
  - Control & status of all stimuli & checkers
  - Reaching Corner cases
    - Value and Cycle related



➔ Divide and conquer at all stages
➔ **Prioritise the reader at all times**

Recap:
The 2022 Wilson Research Group Functional Verification Study (2)

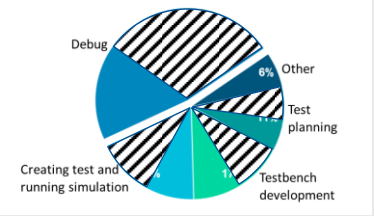
Half the verification time is spent on debugging



2022 WILSON RESEARCH GROUP, FUNCTIONAL VERIFICATION STUDY
FPGA FUNCTIONAL VERIFICATION TREND REPORT

We can definitely be more efficient! - structured!

Huge improvement potential



- Assume a 6000 hour project → 3000 hour verification
- 50% improvement → 1500 hours saved

Courses

- FPGA (and ASIC) Verification:
  'Advanced VHDL Verification – Made simple'
- FPGA (and ASIC) Design:
  'Accellerating FPGA and Digital ASIC Design'

Design
- Design Architecture & Structure
- Clock Domain Crossing
- Coding and General Digital Design
- Reuse and Design for Reuse
- Timing Closure
- Quality Assurance - at the right level
- Faster and safer design

Verification
- Verification Architecture & Structure
- Self checking testbenches
- BFMs – How to use and make
- Checking values, time aspects, etc
- Verification components
- Advanced Verif: Scoreboard, Models, etc
- State-of-the-art verification methodology

Next courses Online in November.
More courses on demand: On-site, Online, Public, or Hybrid

https://emlogic.no/courses/

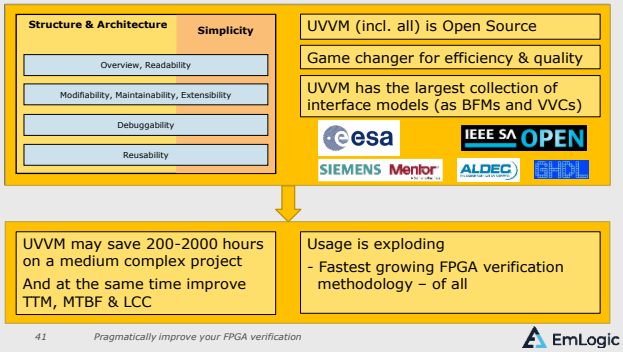
New **ESA UVVM** extension project

esa UVVM FPGA & ASIC & CPLD

- Started up in March
- First new functionality to be published (in June)
  - Detection of unexpected / unwanted transactions or interface activity
  - Completion detection

UVVM in a nutshell

- Huge improvement potential for more structured FPGA verification



Notes

TESSOLVE
A HERO ELECTRONIX VENTURE

Tessolve (a Hero Electronix Venture) 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 Center of excellence initiatives such as 5G, RISC-V subsystems, 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 Center/ Enterprise, Industrial/IoT and Semiconductor 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

180nm
3nm↕
Advanced Process Development

950+
Engineers executing Specs to GDSII solutions

Test/Product Engineering

900+
Test Programs Released

30+
Automated Test Equipment supported

Hardware Development

130+
PCB Designs Per Month

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

Embedded Systems

40+
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

TESSOLVE
A HERO ELECTRONIX VENTURE

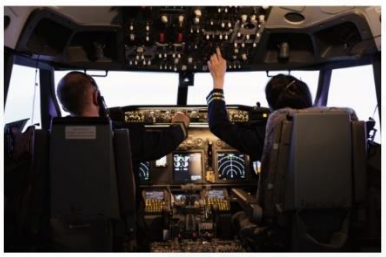
Notes

TESSOLVE
A **HERO ELECTRONIX** VENTURE

SILICON AND SYSTEMS SOLUTION PARTNER

INDUSTRY FOCUS

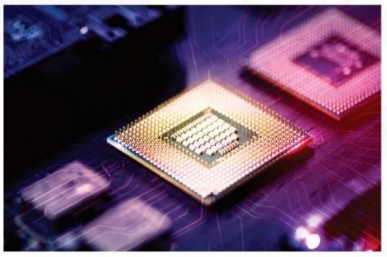
Automotive



Avionics

Data Center/Enterprise

Industrial/IoT



Semiconductor

SCAN ME

Notes

CALL FOR PAPERS

Whatever your specialty, Verification Futures provides an excellent opportunity to share your experiences and insights on the key technical and industry challenges we face in verification.

Submit an Abstract for VF2025

We are now seeking submissions for presentations and papers describing interesting and innovative experiences related to challenges faced in hardware verification. These submissions should include a brief description of the ASIC, SoC and FPGA verification challenges that need to be addresses and/or innovative solutions to improving verification. Abstracts should be targeted toward a technical audience of hardware verification engineers. Abstracts may also address the safety and security issues relating to verification.

Submission Dates

Call for Papers Opens UK & USA	12 September 2024
Final Presentations Required UK	31 March 2025
Final Presentations Required USA	30 June 2025

Abstract Submissions

Abstract submissions should be no more than 2,500 characters and include a short biography of the speaker. All abstracts will be reviewed and notice of acceptance will be sent via email.

To submit your abstract please sent to – mike.bartley@tessolve.com and for guidance please visit https://www.tessolve.com/verification-futures/

About Verification Futures 2025

Verification Futures is a unique one-day conference, exhibition and industry networking event organized by Tessolve to discuss the challenges faced in hardware verification. The event gives the opportunity for end users to define their current and future verification challenges and collaborate with the vendors to create solutions. It's also an excellent opportunity to network and catch up with other verification engineers across Europe & USA.

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/vf2024-austin-usa/