Using UVM-*e* Testbenches Utilizing Metric-Driven-Verification and Advanced Debug Capabilities

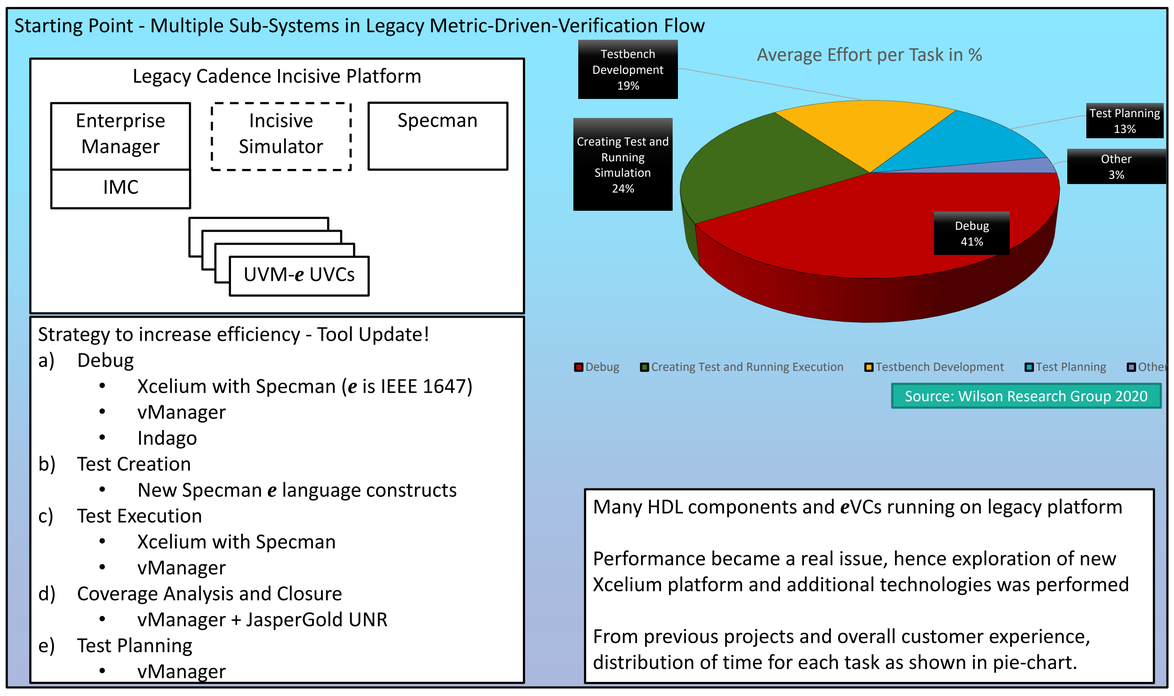
Himanshu Rawal
Intel - India
himanshu.v.rawal@intel.com

Vijay Kumar Birange
Cadence - India
bvijay@cadence.com

Daniel Bayer
Cadence Germany
danielb@cadence.com

Problem Introduction

Multi-Facetted Verification of SoC with Multiple Tools



Strategy to increase efficiency - Tool Update!

a) Debug
   - Xcelium with Specman (***e*** is IEEE 1647)
   - vManager
   - Indago
b) Test Creation
   - New Specman ***e*** language constructs
c) Test Execution
   - Xcelium with Specman
   - vManager
d) Coverage Analysis and Closure
   - vManager + JasperGold UNR
e) Test Planning
   - vManager

Many HDL components and ***e***VCs running on legacy platform

Performance became a real issue, hence exploration of new Xcelium platform and additional technologies was performed

From previous projects and overall customer experience, distribution of time for each task as shown in pie-chart.

Proposed Methodology

Increasing Verification Productivity through Tool Integration

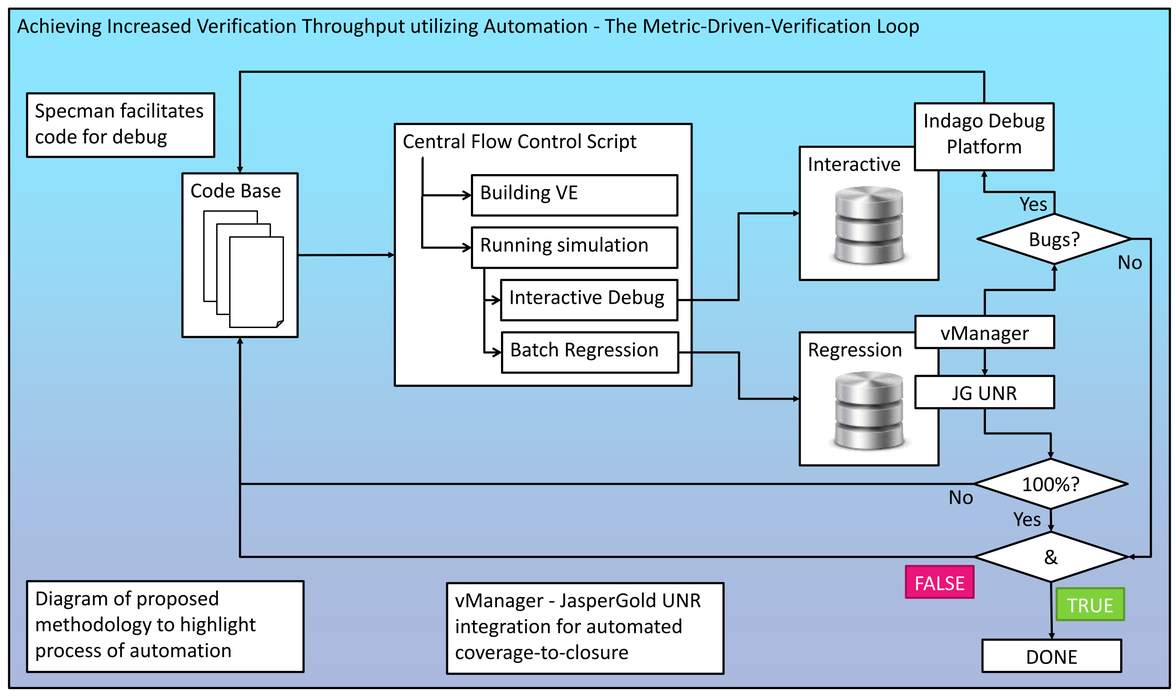


Diagram of proposed methodology to highlight process of automation

vManager - JasperGold UNR integration for automated coverage-to-closure

Implementation Details

Central Script to Coordinate Tools

Analysis Phase
- Detailed environment analysis showed minimal effort moving from Incisive to Xcelium.
- Reuse of scripting infrastructure facilitated rapid migration

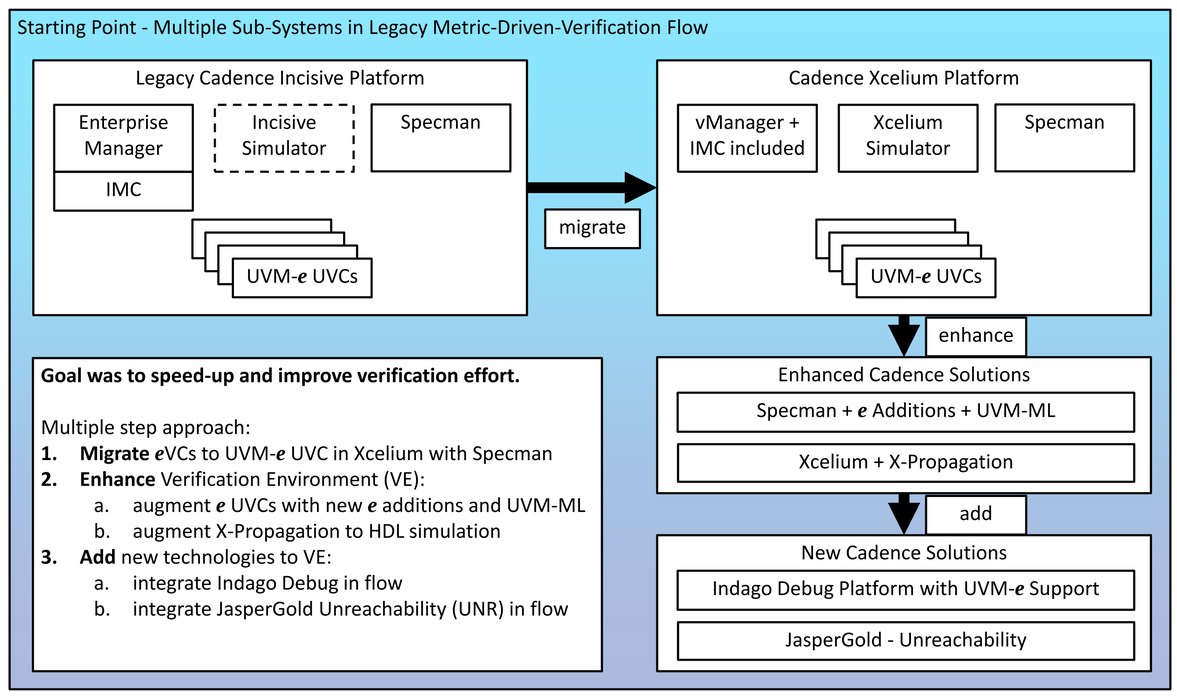
Migration Phase
- Main effort was to adjust legacy constraints to work with Specman's IntelliGen generator
- Deprecated ***e*** language features were slight adjustments in code
- Adjustment of ***e*** Tesflow library was required, and these changes were backported into official Testflow library
- Migration of ***e*** code and HDL code were separate tasks
- Migrating ***e*** code through loading ***e*** files interpreted mode to avoid compilation and linking during migration
- Final migration step is to integrate migrated code into Intel's central script
- Adjusting vManager automation scripts to accommodate failure triage, coverage analysis and reporting

Enhancement Phase
- Partly done during Migration Phase for new and improved language constructs
- Adding performance optimization and X-Prop options to xrun call in central script

Adding New Technology Phase
- **JasperGold UNR integrated in vManager** alleviates engineers from manually analyzing code for unreachable sections
- Adding **Indago through `xrun` options** and getting powerful debug automation

Each step resulted in script recipes that can be easily maintained and applied to any environment.

Implementation Flow Chart



Goal was to speed-up and improve verification effort.

Multiple step approach:
1. **Migrate** ***e***VCs to UVM-***e*** UVC in Xcelium with Specman
2. **Enhance** Verification Environment (VE):
   a. augment ***e*** UVCs with new ***e*** additions and UVM-ML
   b. augment X-Propagation to HDL simulation
3. **Add** new technologies to VE:
   a. integrate Indago Debug in flow
   b. integrate JasperGold Unreachability (UNR) in flow

Results Table

Verification Progress	Effort Saved	Performance Improvement over previous Tools/Methodologies	Quality Enhancement
Conversion of over 10 testbenches to latest tools and methodology	~ 4 weeks	~ 20% - 30%	Critical bugs found and reported
Regression Management	~ 2 - 3 weeks	~ 30% on overall regression	-
Coverage Unreachability	The **added technologies could not be quantified against** the **previous flow**, to avoid false comparison methodology. **Generally**, each of the **technologies saves** anywhere **between 10% to 30%** of the overall project schedule and **improve predictability** on each **subsequent project**. The **overall Quality Enhancements** for **Coverage Unreachability** is that engineers spend **significantly less time** on finding unreachable code, due to the automated process of UNR, which is generally a **low-effort** with **high-impact** technology. **Debug Productivity** is measured in **time-to-root-cause**. In other projects, there is **typically** an **improvement of 20%** and more, depending on the root-cause issue.		
Debug Productivity (X-Prop and Indago)			
Coverage Closure			

Conclusion

- **Verification consumes** a significant portion of **time and resources** of the SoC development process.
- Hence, it is **necessary to** tightly **integrate and automate**
  - project planning
  - test execution
  - regression analysis
  - debug process
  - coverage closure

  to minimize resources and **achieve** a predictable **reduction of the project schedule**.
- **All phases** must be **governed by processes** that facilitate to **minimize project time**, **engineering effort** and **compute resources**.
- By choosing Specman with its built-in UVM-***e*** methodology, **migrating legacy code-bases and develop new testbenches**, as well as **integrating both architectures** together seamlessly is **facilitated by the language itself**.
- This **enabled rapid tool-migration**, which achieved **receiving out-of-the-box performance boost** through **updated tool versions** and **additionally augmenting** the **planning and debug** process at the same time.
- The result comparison show excellent time-savings throughout all verification areas.