2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY
OCTOBER 15-16, 2024

Calling All Engines - Faster Coverage Closure with Simulation, Formal, and Emulation

Yassine Eben Aimine, Product Architect
Dirk Hansen, Application Engineer

accellera SYSTEMS INITIATIVE

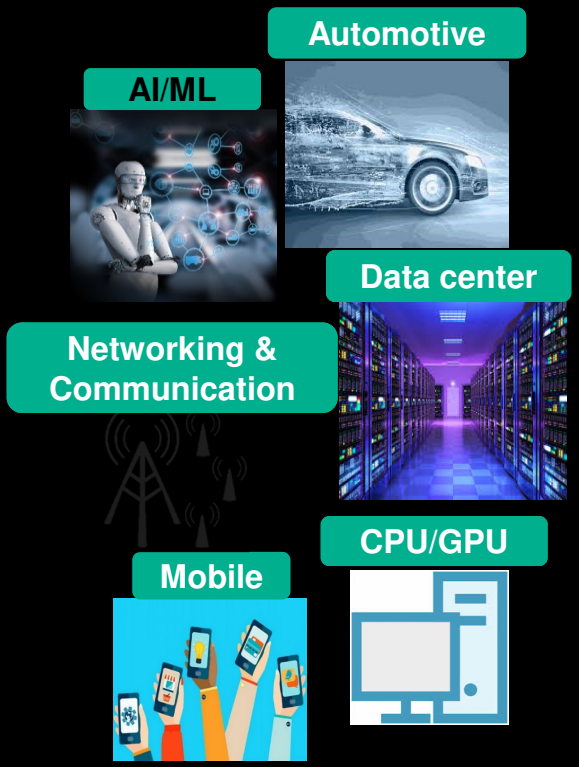
Agenda

- The Case for Unified Coverage
- Simulation & Emulation Coverage
- Formal Coverage
- Coverage Merge
- Unified Coverage Analysis
- Conclusion and Q&A

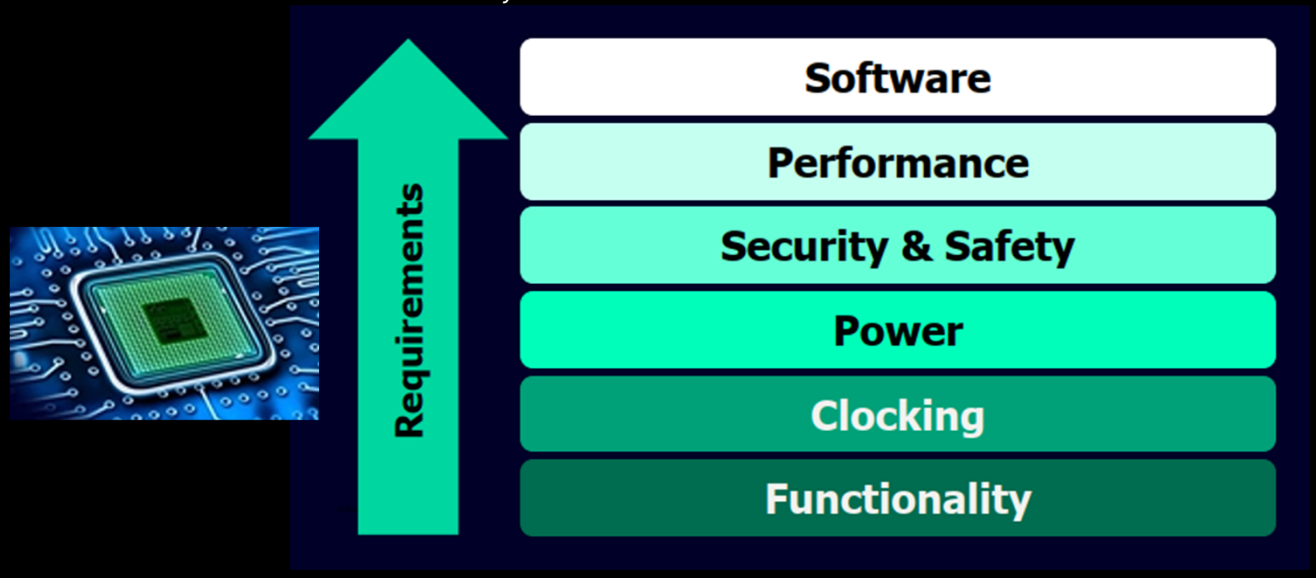
Agenda

- **The Case for Unified Coverage**
- Simulation & Emulation Coverage
- Formal Coverage
- Coverage Merge
- Unified Coverage Analysis
- Conclusion and Q&A

Market dynamics drive new use-cases and requirements



Source: Wilson Research Group and Mentor, A Siemens Business, 2022 Functional Verification Study



Complexity in terms of functional verification and HW-SW integration is predominantly due to these increasing levels of requirements

Project impact

Decline in first silicon success and cost increase significantly affect TTM and business success

66 %

Projects behind schedule

50-60 %

Median project time spent in verification

76 %

ASICs require 2 or more respins

13 %

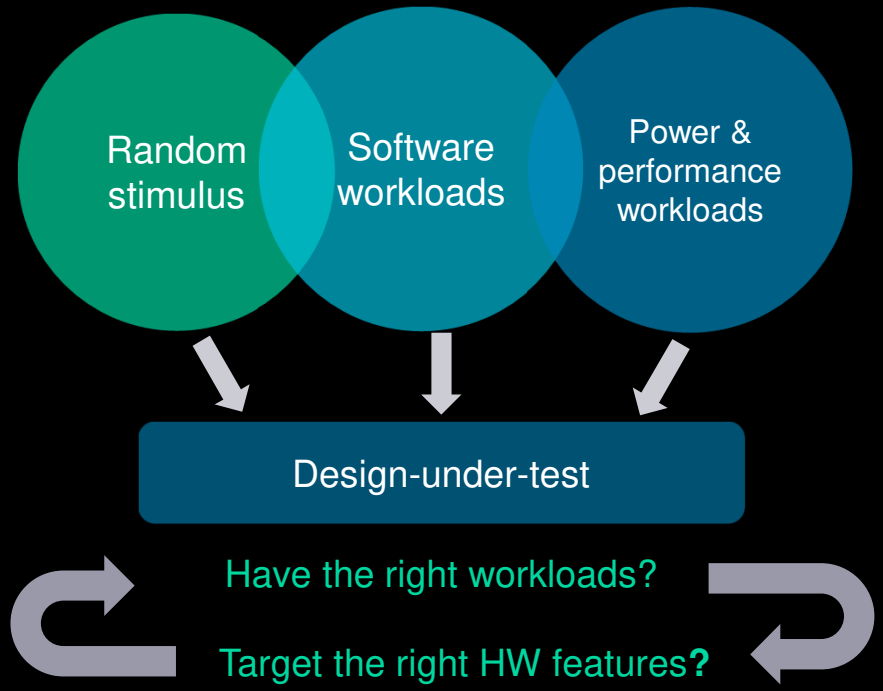
First silicon success on 10M-1B gates designs

62 %

Logical/Functional flaws caused respins in design greater then 1B gates

Source: Wilson Research Group and Mentor, A Siemens Business, 2022 Functional Verification Study

Understand the challenge to create the opportunity



Source: Wilson Research Group and Mentor, A Siemens Business, 2022 Functional Verification Study

Every project benefits from a robust coverage and assertion methodology

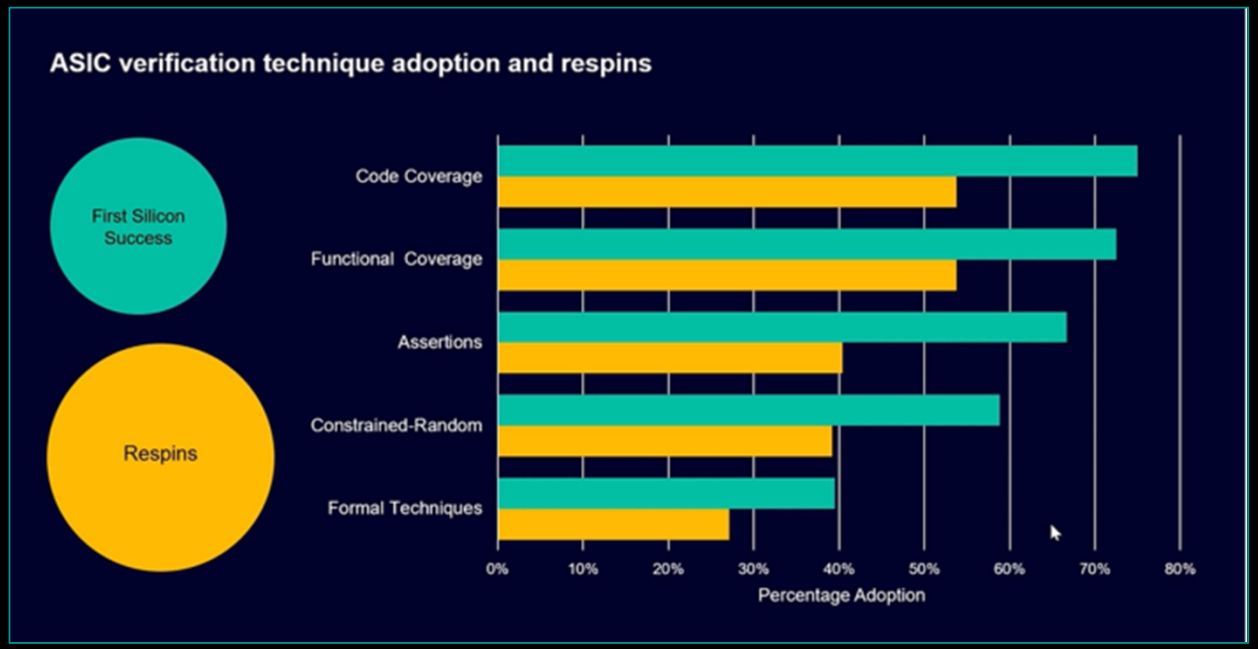
Why the need for coverage and assertion ?
Metrics to measure verification progress and quality and productivity

Code coverage
- Design structure or implementation is covered

Functional coverage
- Intent of the design or specification is covered

Assertion
- A given temporal condition holds or is covered
- Improve debug productivity

Source: Wilson Research Group and Mentor, A Siemens Business, 2022 Functional Verification Study



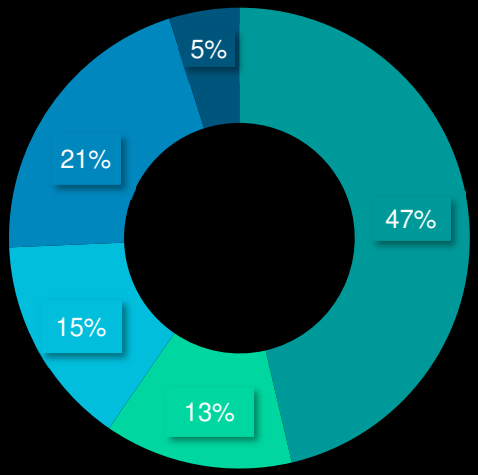
Leverage coverage & assertions to avoid bug escapes from block to system level

ASIC/FPGA verification effort

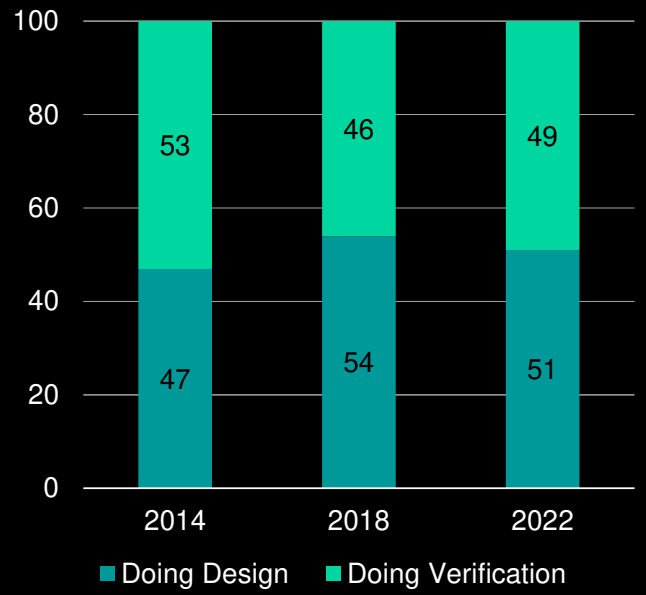
Coverage & assertion-based verification are growing requirements in emulation

2024 DVCON EUROPE MUNICH, GERMANY OCTOBER 15-16, 2024

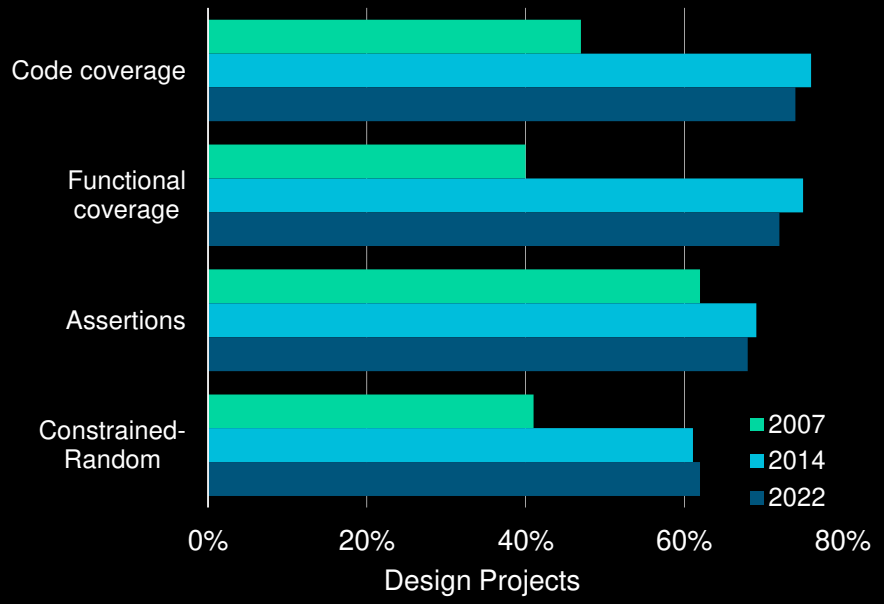
Where IC/ASIC **verification engineers** spend their time



Where IC/ASIC **design engineers** spend their time



ASIC adoption of dynamic techniques



Source: Wilson Research Group and Mentor, A Siemens Business, 2022 Functional Verification Study

An SoC level multi-verification phase and tool challenge

Adopt the right verification engine for effective verification sign-off

2024 DVCON EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024

Block-level verification plan

- Improve block quality with exhaustive testing
- Simulation and formal tools for 100% verification closure

IP/sub-system verification plan

- HW functional, power & performance verification
- Simulation and emulation regressions with coverage & assertions

System-level verification plan

- SW, power/perf targeted workloads
- Verify SoC HW/SW integration with coverage and assertions

FORMAL

Property checking
- Catch bugs early
- Identify more assertions

Explore reachability of design

Collect exhaustive formal coverage

SIMULATION

Goal-driven stimulus

Test saturation
- Run constrained random stimulus

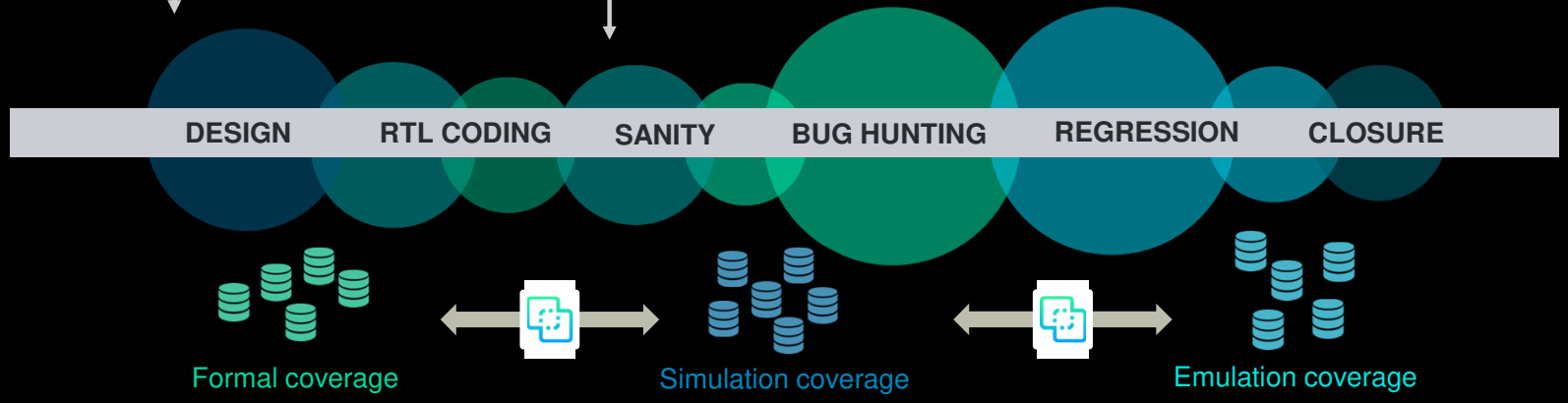
Primary coverage collection engine

EMULATION

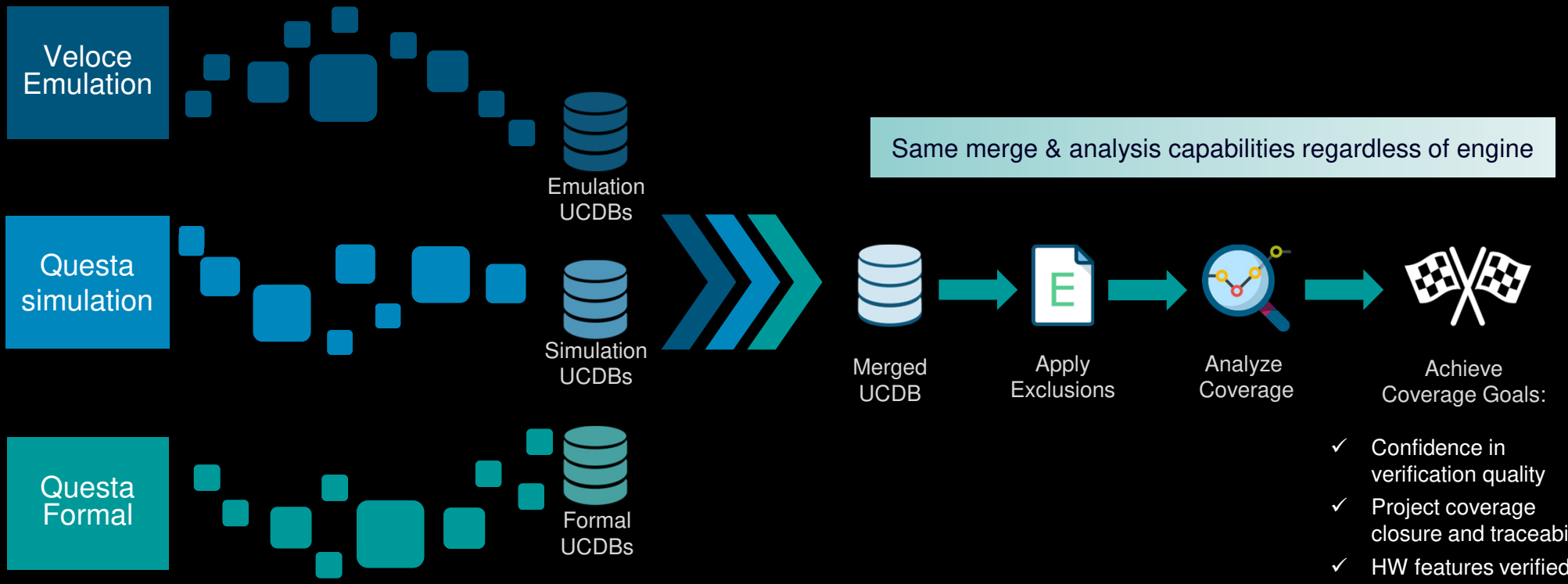
Acceleration and long workloads

Close difficult coverage holes

System level testing



Combining Questa Simulation, Questa Formal and Veloce emulation coverage



Agenda

- The Case for Unified Coverage
- **Simulation and Emulation Coverage**
- Formal Coverage
- Coverage Merge
- Unified Coverage Analysis
- Conclusion and Q&A

Purpose of Code Coverage

Determine the effectiveness of your testbench

- Does it exercise the HDL model (source code)
- Helps automate design process.

Use to improvement / gain confidence in the test suite

- Use in a manufacturing functional test

Shorten Lab Debug Time

Expose corner case bugs

Purpose of Functional Coverage / Assertion covers

- The role of the functional coverage model is to ensure that the tests that the DUT passes have checked the design features for all of the relevant conditions.
- The testbench will exercise the features of the design. The role of the functional coverage model is to check that the different variants of those features have been observed to work correctly.
- Features may also be referred to as requirements or in some situations as stories.

Purpose of using both types of coverage

100% code coverage means

- All nodes inside the design have been stimulated

100% functional coverage means

- All possible combinations of input stimuli have been applied to the design, and the output is proven correct

100% code coverage and 100% functional coverage ➔ DONE TESTING

Major Types of Coverage

Code Coverage

- Did all statements, branches of code get exercised?
- Did all signals toggle at least once?
- Automated in the simulation environment
- A basic measure with little correlation to functionality

FSM Coverage

- Did all the states and transitions get exercised?
- Automated in the simulation environment
- Typically included with code coverage
- Indicates transitions exercised, not functionally correct

Functional Coverage

- Have all the important values/ranges of a signal been covered?
- Have sequences / transitions between signals been covered?
- Verification engineer must write covergroups, bind into RTL
- Are the covergroups written correctly? Do they cover enough?

Assertion Coverage

- How many times did the assertion get evaluated and pass, and fail?
- Designer must write assertions and cover properties
- Is the assertion implemented correctly? Check anything of value? Are there enough assertions?

Veloce Coverage & Assertion App – Addressing Coverage Closure Needs

Assertions and SVA Coverage

Unexpected scenarios/error conditions?

Occurrence of interesting conditions?

Code Coverage

All statements/branches exercised?

All signals toggled?

All FSM states reached? All state transitions taken?

Functional Coverage

Have signals taken specific set of values?

Have signals taken specific sequence of values?

Did interesting value combinations occur?

Coverage Metric Holes

Code / FSM / Assertion Coverage

- Functional dead code and unreachable FSM states/transitions
- Modes of the design that create dead code
- Time can be wasted trying to hit these holes!

Transaction/Structural Coverage

- Testbench doesn't stress the design enough
- Incomplete models don't exercise all transactions

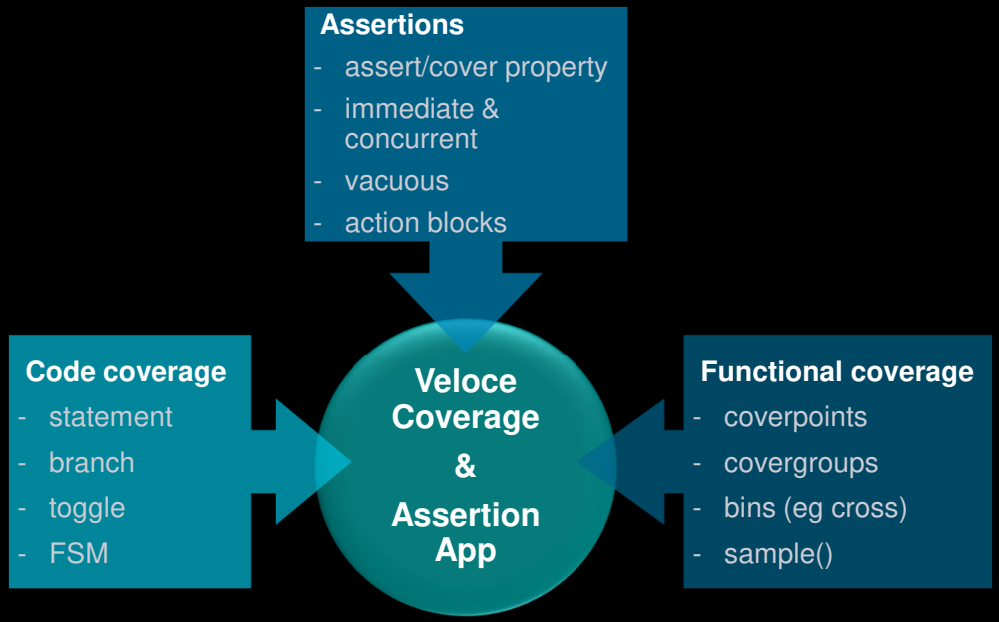
Functional Coverage

- Incomplete spec or planning, lack of knowledge/time

Proper test planning can mitigate some of these challenges

Static techniques such as Questa CoverCheck can minimize time to closure

Accelerate coverage closure and assertion-based debug



Re-use simulation coverage and assertions to achieve metric-driven verification goals in emulation

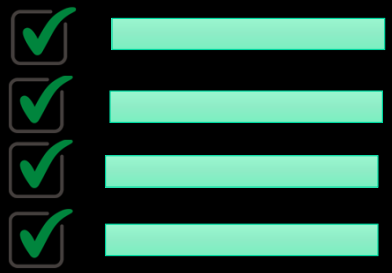
Run real workloads that effectively cover the system features and stress scenarios

Fast coverage closure & efficient assertion-based debug for complex system level bugs

Effectively merge coverage with simulation and emulation results for project level view

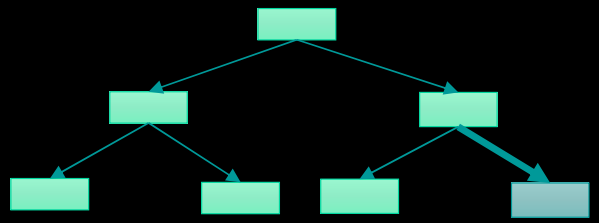
Code Coverage Metrics Supported by Veloce Coverage App

Statement Coverage



Check if all procedural statements in always blocks (clocked/combinational) are exercised

Branch Coverage

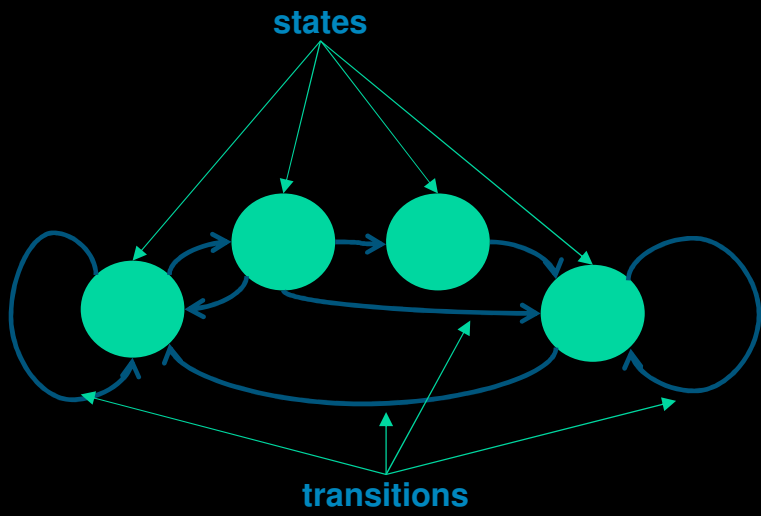


Check if all branches in always blocks (clocked/combinational) are exercised

- if/case/conditional operator

Also check if none of the branches are taken (missing else)

FSM Coverage



State Coverage – Check if all states of the FSM have been covered

Transition Coverage – Check if all **possible** transitions between states have been covered.

Can also be determined by branch coverage, but a more intuitive representation

Toggle Coverage

Check if all signals have toggled from 0->1 and 1->0

Methodology Considerations

- Once enabled, code coverage is automatic. No changes / additions to RTL required.
- While easy to "turn on everything," doing so can significantly impact capacity.
- Counter width settings and thresholds are global.
  - Be mindful of capacity increases and adjust counter widths accordingly.

Coverage Database: UCDB

- The **U**nified **C**overage **I**nteroperability **S**tandard (UCIS) from Accelera defines the schema/format for storing Coverage information, known as UCISDB (or UCDB)
  - Manipulation via standard APIs, implementation is tool-specific
  - Questa fully supports this standard
- Integration in Veloce to generate Questa-compatible UCDB
  - All actions/manipulations possible with Questa-generated UCDB also possible with Veloce-generated UCDB
    - Reporting
    - Merging (Veloce-Veloce or Questa-Veloce)
    - Ranking
  - Questa Coverage ecosystem available to Veloce as well
    - vcover utility
    - Visualizer
    - Coverage Analyzer

Unified coverage and assertion metrics

	Statement	Branch	Block	Expr/Cond (FEC)	FSM (State/Trans)	Toggle	Covergroup	Assert / Cover
Questa	✓	✓	✓	✓	✓	✓	✓	✓
Veloce	✓	✓	🕒	🕒	✓	✓	✓	✓
Formal	✓	✓	✓	✓	✓	✓	✓	✓

✓ Supported

🕒 Future Support

Agenda

- The Case for Unified Coverage
- Simulation & Emulation Coverage
- **Formal Coverage**
- Coverage Merge
- Unified Coverage Analysis
- Conclusion and Q&A

Adding formal coverage is critical to success

Complementary with simulation, deliver unique abilities and acceleration

Increase simulation efficiency

Target blocks for complete verification using formal techniques measuring success with coverage

Get answers now/early

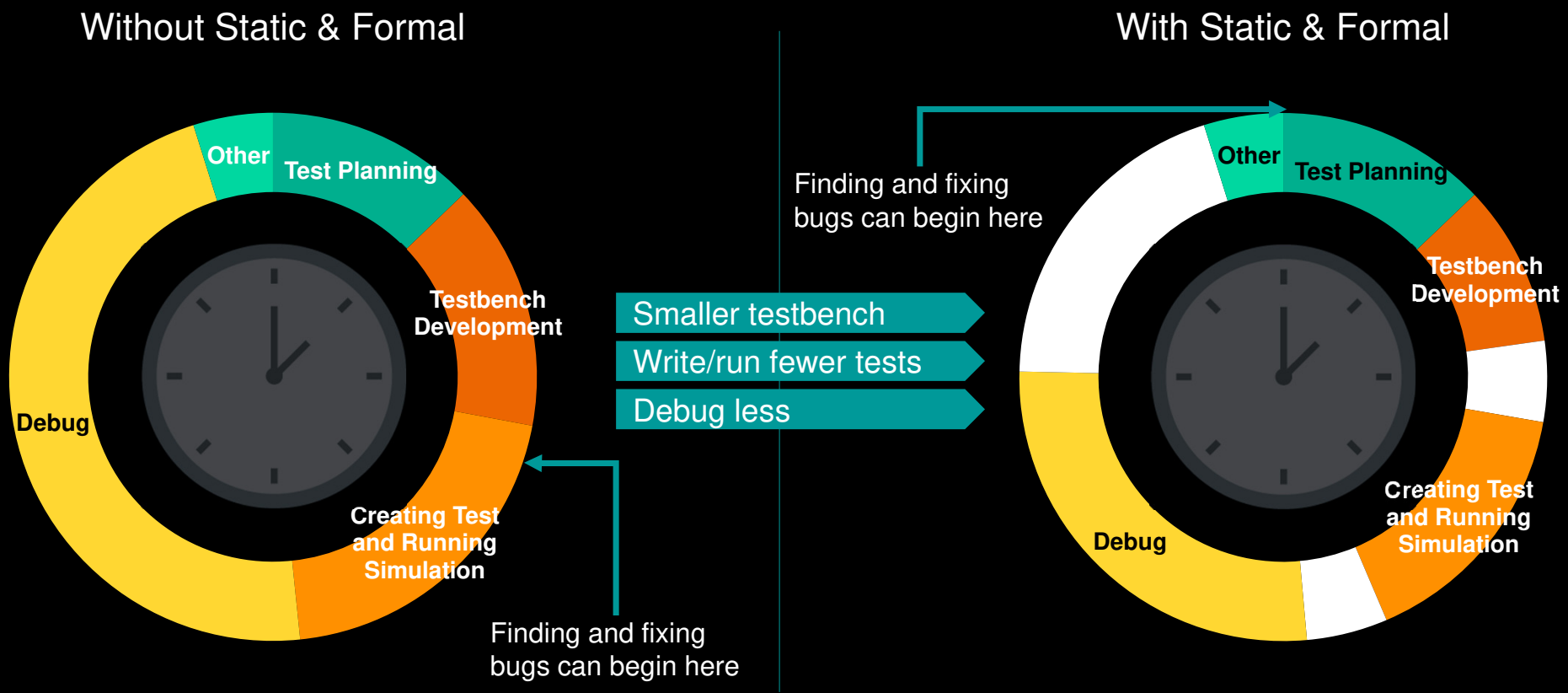
Exhaustive analysis focused on intent and expectations to find bugs and optimize simulation efforts

Find what simulation won't

Cover areas hard to reach with simulation – vectorless coverage

Formal adoption does more than increase simulation efficiency

Finding and fixing bugs before testbench increases team agility



Source: Wilson Research Group and Mentor, A Siemens Business, 2022 Functional Verification Study

Reachability – simulation coverage closure

Questa Increase Coverage performs a reachability analysis for all code coverage items

Reachability provides design insights in form of:

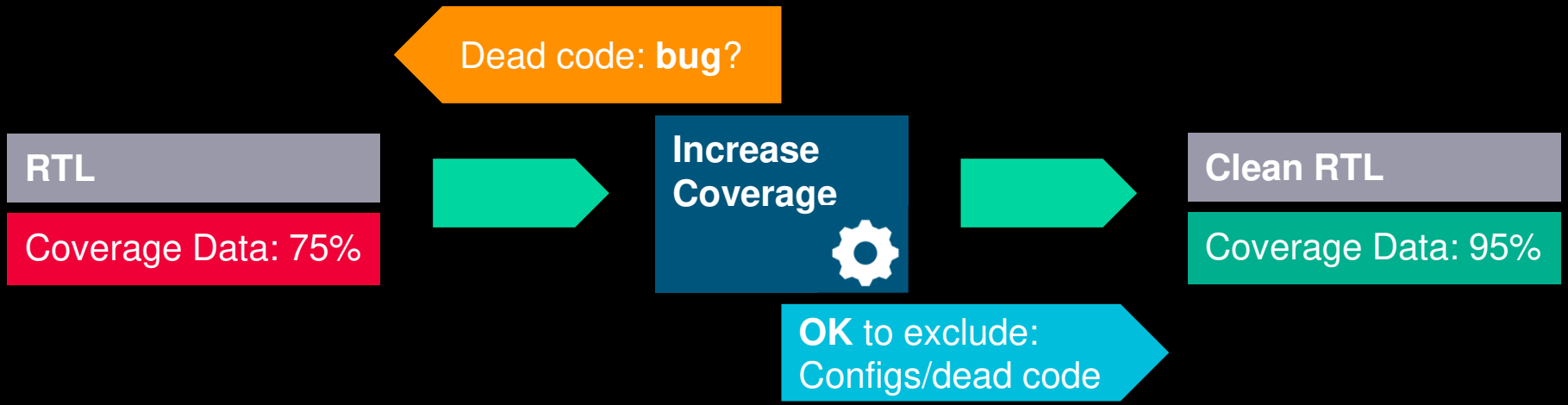
- Dead code analysis
- FSM state and arc analysis
- Signal stuck at value analysis
- Witness trace
- Supports same 7 coverage types Questa Sim
  - bcefst and bin

Used to accelerate simulation coverage closure

- Tight integration with Questa Sim and UCDB
- Integrated with Coverage Analyzer

```
always @*
 if (a)       R = 2'b00;
 else if (b)  R = 2'b01;
 else         R = 2'b11;
…
always @*
 case (R)
  2'b00:       T = 1'b0;
  2'b01:       T = 1'b1;
  2'b10:       T = 1'b1;
  2'b11:       T = 1'b0;
 endcase
```

Questa Increase Coverage
Statically identifies unreachable code



Increase Coverage advantages

- Save man-months of effort closing coverage
- Tight integration with UCDB
- Automation of exclusions
- Identifies unreachable coverage targets
- Multi-vendor solution

Reachability and observability – formal code coverage

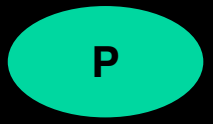
Questa Verify Property performs a reachability and observability analysis for all code coverage items

Reachability provides design insights in form of:

- Dead code analysis
- FSM state and arc analysis
- Signal stuck at value analysis
- Over-constraint analysis
- States and signals contributing to property proof

```
always @*
 case (sel)
  2'b00:    en = 1'b0;
  2'b01:    en = 1'b1;
  2'b10:    en = 1'b1;
  2'b11:    en = 1'b0;
  default:  en = 1'b0;
 endcase
```

Observability provides design insight in form of structural coverage contributing to property proof

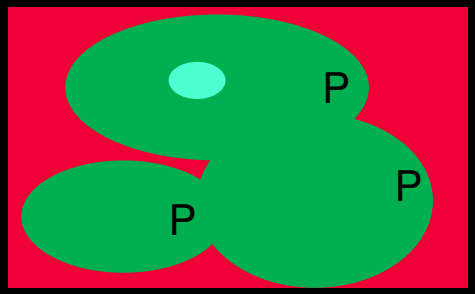


Formal signoff code coverage

Based on property proof coverage

Familiar code coverage types with simulation

- Branch
- Condition
- Expression
- FSM
- Statement
- Toggle
- Covergroup bin



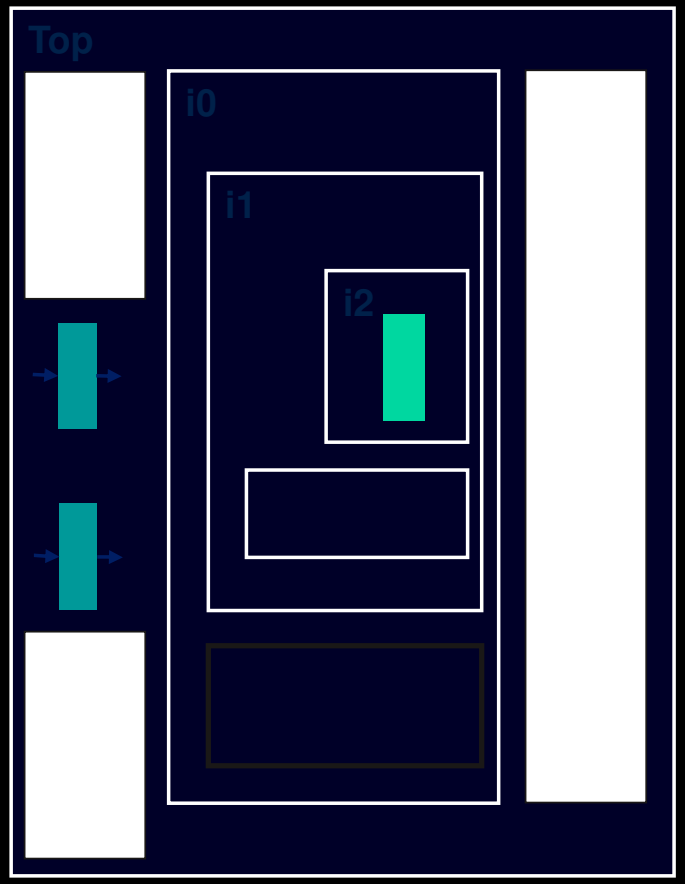
Based on two concepts:

- Reachability
  - For a given property and its cone of influence, what logic is reachable from the inputs?
- Observability
  - For a given property, what logic was used to prove the property (is observable by a property)?
- 100% code coverage doesn't equal bug free RTL
  - Focus on what isn't covered, improve TB there

Reachable	Observable	Coverage Result
Yes	Yes	Covered
Yes	No	Uncovered
No	-	Excluded
Inconclusive	-	**Uncovered**

Questa Verify Property

Enables scalable formal coverage for sub blocks verified with property checking



Specification (property) is equivalent to the DUT?

PASS

Yes!
Record in UCDB

FAIL

Waveform that shows violation

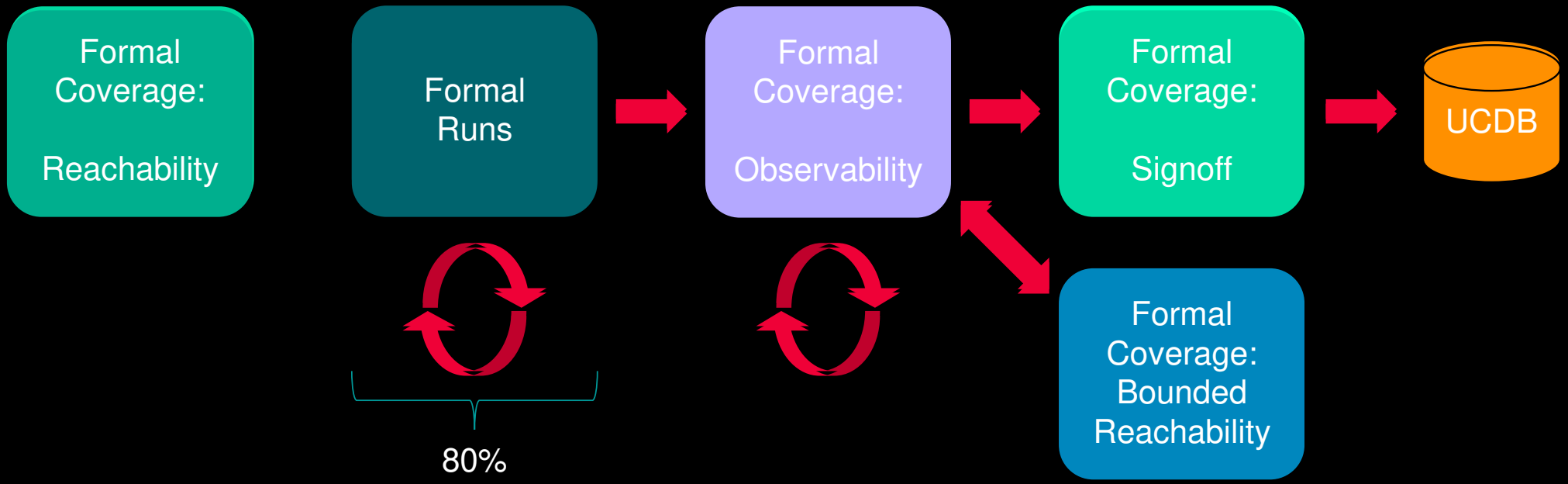
Formal UCDB 1

Formal UCDB 2

Merged all engine coverage database

Verify Property formal coverage methodology

The 4 types of formal coverage: reachability, observability, bounded reachability, and signoff



Formal feasibility	Develop formal tb	Run formal coverage	Final coverage run	Test planning
Find bugs	Run normal formal	Add asserts	Generate UCDB	Trends / analysis
Over-constraint	Refine assert/assume	Add exclusions	Bounded Proof chk	Merging of data

Analysis: Reachability

Reachability analysis shows if a cover item (sbceft bin) is reachable from the design inputs

- Can be used for insight into the complexity of the design
- Can be used with or without properties
- Unreachable logic is a bug or can be excluded

Example FSM design:

- Initial run has unreachable FSM state and transitions, bug?

Initial run

```
case(cstate)
3'b001: if (dir)
                nstate <= 3'b010;
        else
                nstate <= 3'b001;
3'b010: if (dir)
                nstate <= 3'b010;
        else
                nstate <= 3'b001;
3'b100: if (dir)
                nstate <= 3'b001;
        else
                nstate <= 3'b010;
default: nstate <= 3'b001;
endcase
```

Fixed run

```
case(cstate)
3'b001: if (dir)
                nstate <= 3'b010;
        else
                nstate <= 3'b100;
3'b010: if (dir)
                nstate <= 3'b010;
        else
                nstate <= 3'b001;
3'b100: if (dir)
                nstate <= 3'b001;
        else
                nstate <= 3'b010;
default: nstate <= 3'b001;
endcase
```

Reachability

Summary (Recursive)

Coverage	Total	Unreachable	Inconclusive	Reachable
Branch	13	1	0	12 (92.3%)
Condition	OFF	OFF	OFF	OFF (N/A)
Expression	OFF	OFF	OFF	OFF (N/A)
FSM State	3	0	0	3 (100.0%)
FSM Transition	5	0	0	5 (100.0%)
Statement	12	1	0	11 (91.7%)
Toggle	OFF	OFF	OFF	OFF (N/A)
Covergroup Bin	0	0	0	0 (N/A)
Total	33	2	0	31 (93.9%)

Analysis: Observability

Observability analysis shows if a cover item (sbceft bin) is observable by a property

- Is used to determine where in the design to add more properties
- Is calculated from proofs or bounded proofs during the formal run

Example FSM design:

- Add properties that exercise the uncovered logic, repeat till covered

Observability

Summary For 1 Property (Recursive)

Coverage	Total	Unobservable	Observable (Proofs)
Branch	13	0	13 (100.0%)
Condition	OFF	OFF	OFF (N/A)
Expression	OFF	OFF	OFF (N/A)
FSM State	3	0	3 (100.0%)
FSM Transition	5	0	5 (100.0%)
Statement	12	2	10 (83.3%)
Toggle	OFF	OFF	OFF (N/A)
Covergroup Bin	0	0	0 (N/A)
Total	33	2	31 (93.9%)

Uncovered logic

```
3'b010: if (dir)
                nstate <= 3'b010;
        else
                nstate <= 3'b001;
3'b100: if (dir)
                nstate <= 3'b001;
        else
                nstate <= 3'b010;
default: nstate <= 3'b001;
endcase

always @(posedge clk) dir_r <= dir;

assign pul = cstate == 3'b001 & dir_r;
```

Property added to cover it

```
3'b010: if (dir)
                nstate <= 3'b010;
        else
                nstate <= 3'b001;
3'b100: if (dir)
                nstate <= 3'b001;
        else
                nstate <= 3'b010;
default: nstate <= 3'b001;
endcase

always @(posedge clk) dir_r <= dir;

assign pul = cstate == 3'b001 & dir_r;
```

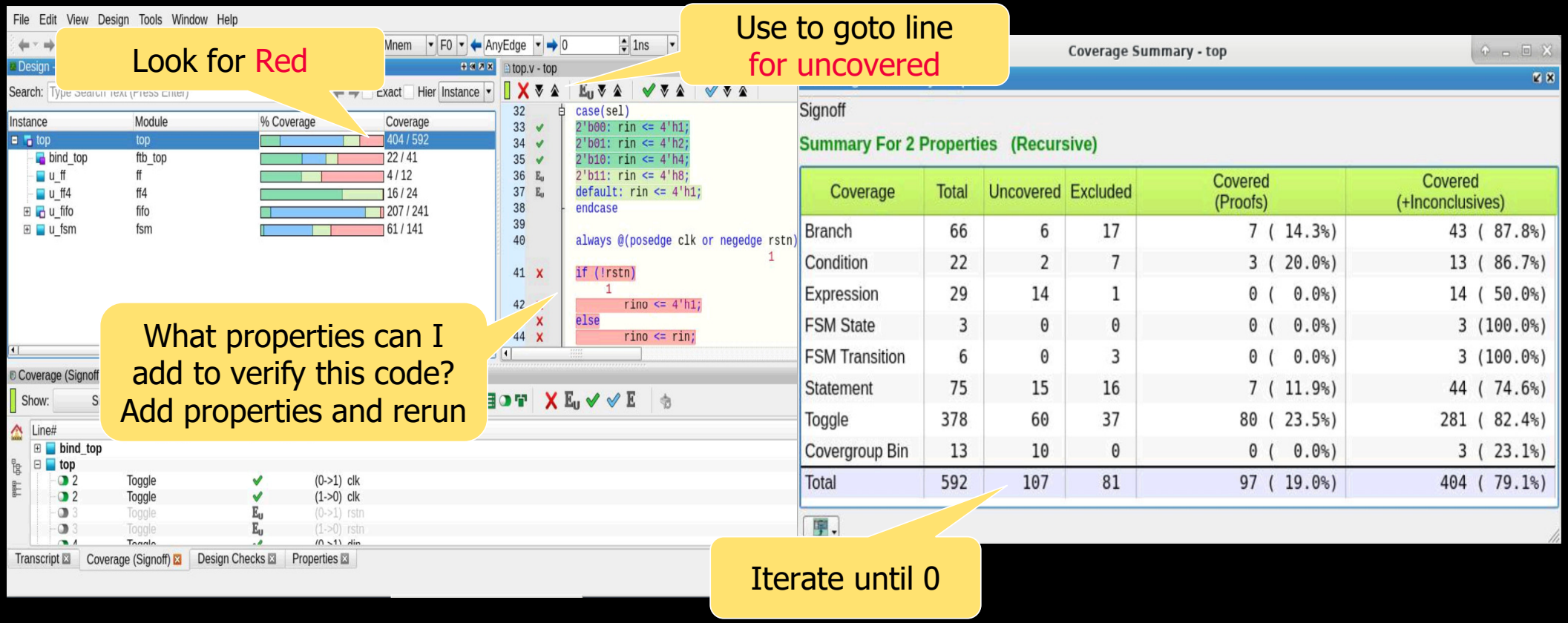
Signoff coverage: Obs + Rea

```
✔  3'b010: if (dir)
✔                  nstate <= 3'b010;
✔          else
✔                  nstate <= 3'b001;
✔  3'b100: if (dir)
✔                  nstate <= 3'b001;
✔          else
✔                  nstate <= 3'b010;
Eu default: nstate <= 3'b001;
   endcase

✔  always @(posedge clk) dir_r <= dir;

✔  assign pul = cstate == 3'b001 & dir_r;
```

Verify Property debug: formal signoff coverage

19% signoff coverage with proofs, still a lot uncovered, need to add properties to uncovered areas

- Add properties that verify to the spec the logic that is uncovered (highlighted in red)



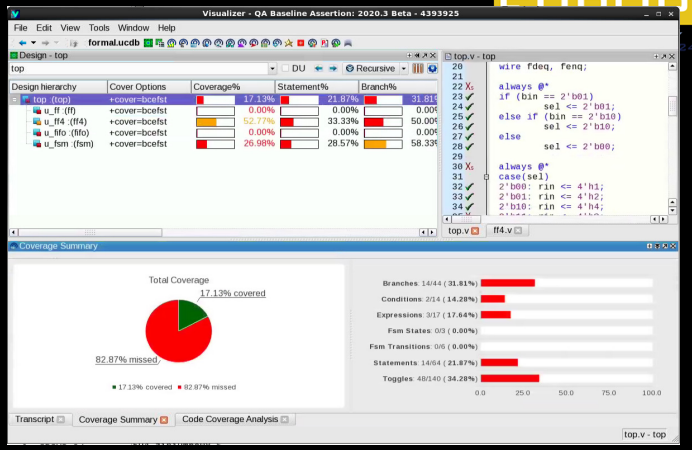
Coverage tracking and analysis
Questa Verify Property

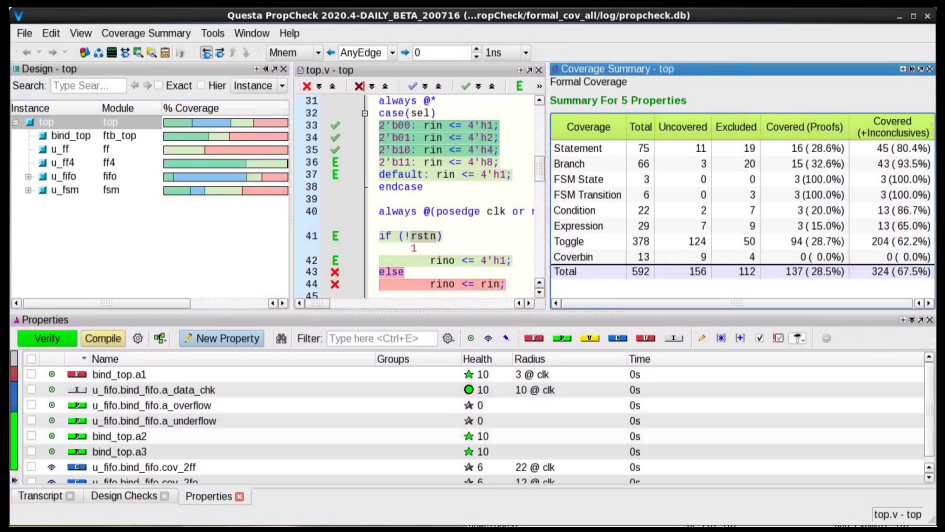
Coverage Analyzer and Visualizer supporting views

- Will include higher-level tracking management systems (Polarion, etc.)

Formal coverage workflow

- Debug done in Verify Property
- Tracking/Management same as simulation/emulation using the UCDB

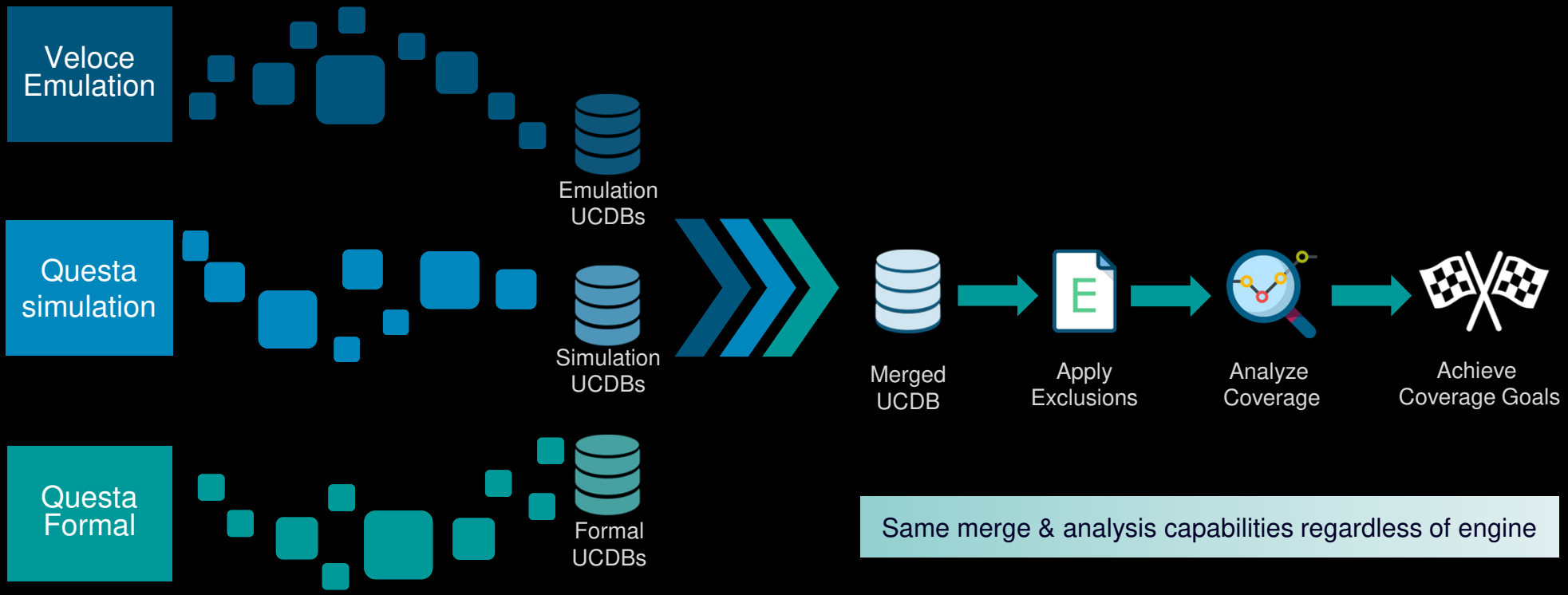




Agenda

- The Case for Unified Coverage
- Simulation & Emulation Coverage
- Formal Coverage
- **Coverage Merge**
- Unified Coverage Analysis
- Conclusion and Q&A

Combining Questa Simulation, Questa Formal and Veloce emulation coverage



Same merge & analysis capabilities regardless of engine

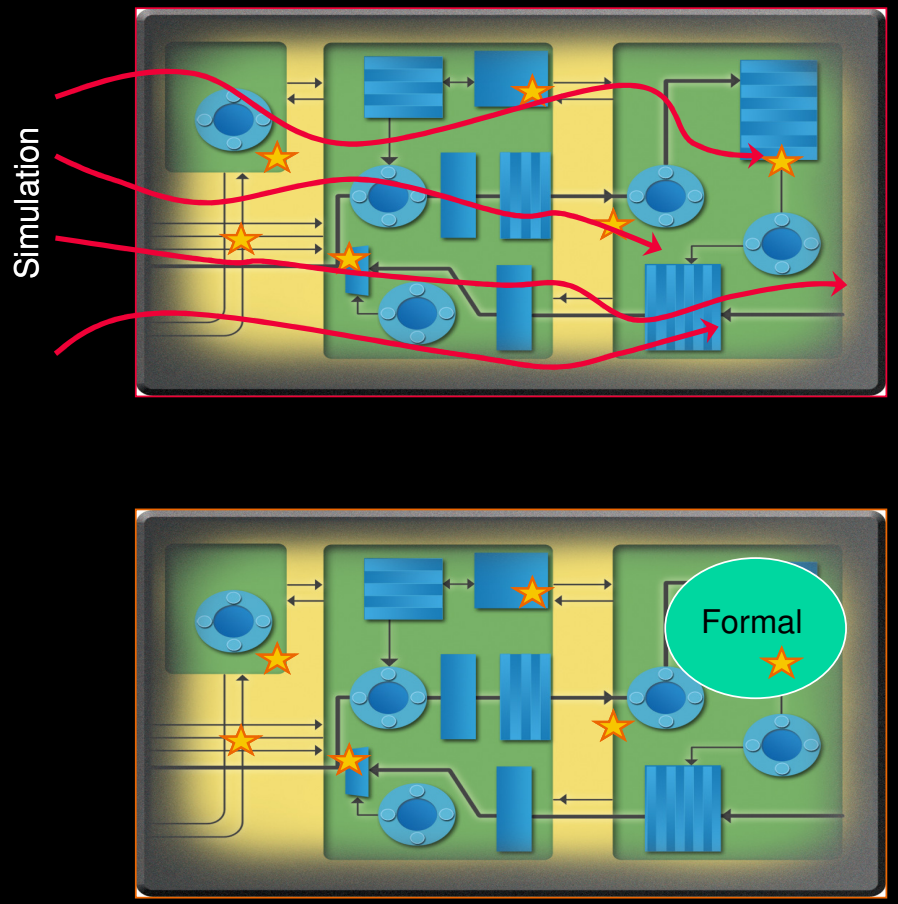
Comparing formal coverage to simulation/emulation coverage

Simulation Code Coverage = Vector Coverage

- Is based on vectors run through the design
- Whatever a vector hits is covered
- Can be logic related to / unrelated to what is being tested
- Property coverage = pass/fail
- For all vectors – looking for what isn't covered
- Action: Run more vectors to hit what isn't covered

Formal Code Coverage = Property Coverage

- Is based on proven properties in the design
- What logic is used to prove a property is covered
- Only related to a given property
- Property coverage = proven/fired/inconclusive
- For all properties – looking for what isn't covered
- Action: Add more properties to hit what isn't covered
- Is a combination of reachability and observability



Don’t cross the streams
Example comparing simulation and formal coverage

Each tool's coverage is there to point to holes in each tool's respective flows

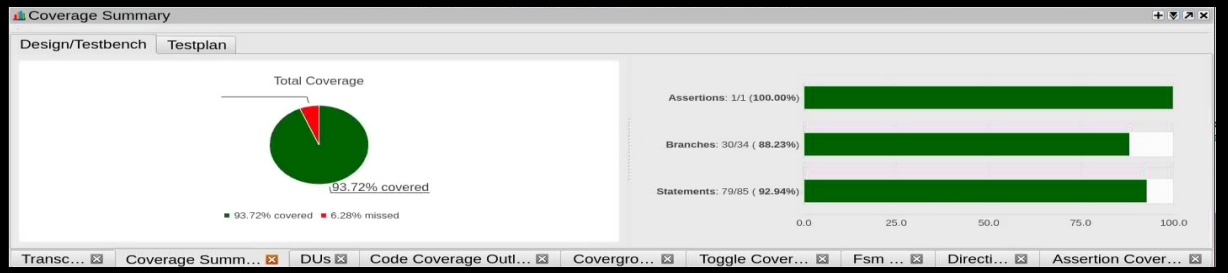
- **Mixing coverage masks this data (customers will still want to do it)**
- **The semantics of each is different, how it is calculated is different**
  - Sim coverage is more optimistic when compared to formal coverage
- **Example: Design has A,B,C outputs, 1 hot 0. Formal proves 1 hot property, sim does vectors for all 3 outputs**

Formal coverage from UCDB (s,b)

- Statement – 5.45%
- Branch – 0%

Sim coverage from UCDB (s,b)

- Statement – 92.94%
- Branch – 88.23%



Recommendations for merging formal with simulation coverage

Using both formal and simulation coverage to ensure robustness and correctness

Include formal in testplanning

- Complete testplanning for the whole project includes sim, formal, and emulation and which engine to use for verification and coverage

Close coverage per engine

- Focus on improving the testbench robustness by closing coverage in each verification engine before merging

Keep coverage data separate

- Knowing where your coverage data comes from is important for closing coverage and only merging it for final analysis

Meaningful property closure

- Add properties to close code coverage holes that test features and requirements of the design will give you the highest quality of verification

Do make use of all types of formal coverage

- Functional for requirements closure
- Reachability for sim and formal closure
- Observability / mutation for improving formal tb
- Sign-off for integration

Why you want to use the various types of formal coverage
Success requires these flows to be deployed

Functional

- Assertions check design behavior
- Cover statements show parts of the design covered
- Both tied to test-planning and requirements

Cone of Influence

- Structural COI based
- Very early code coverage
  Where to add properties
- Not generally used

Mutation

- Design mutation based
  Is property impacted?
- Very accurate code coverage
  Where to add properties
- Computationally intensive

Reachability

- Unreachable logic checking
  Bugs and configuration
- Used to increase sim code coverage
- Over-constraint checking

Observability

- Property based code coverage
- Logic used to prove the property is observable by the property
- Early guide on where to add properties

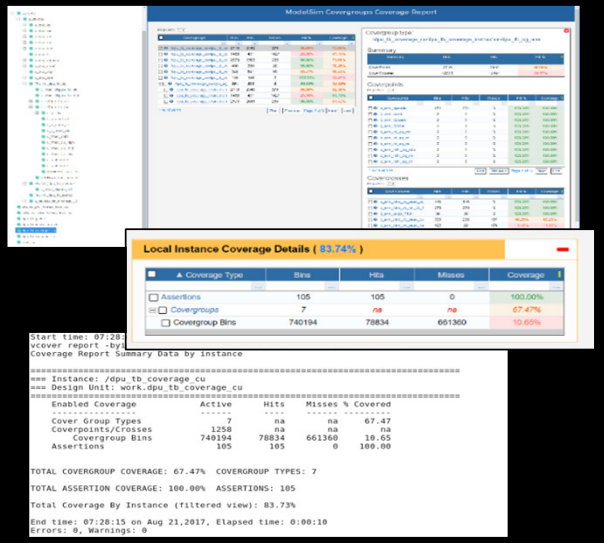
Signoff

- Combination of observability and reachability
- Final coverage data to UCDB
- Integration with other verification engine data and coverage closure tools

Agenda

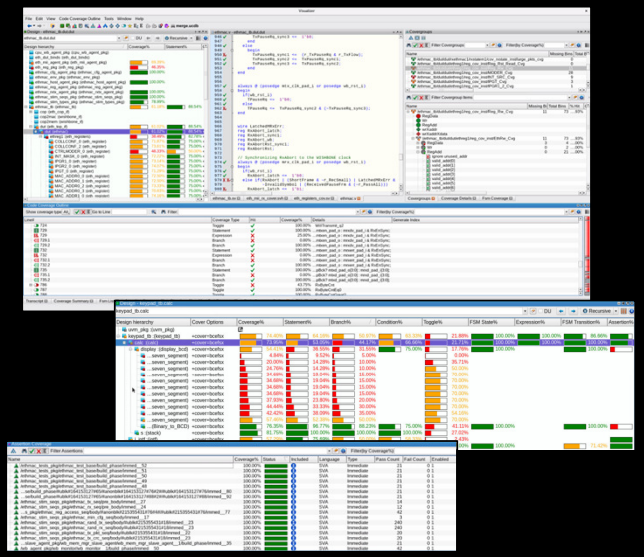
- The Case for Unified Coverage
- Simulation & Emulation Coverage
- Formal Coverage
- Coverage Merge
- **Unified Coverage Analysis**
- Conclusion and Q&A

Unified coverage and assertion analysis – Stand-alone to cloud-based collaborative workflows



Text / HTML

- Generate coverage reports to track verification closure
- Fast adoption but limited visualization and collaboration



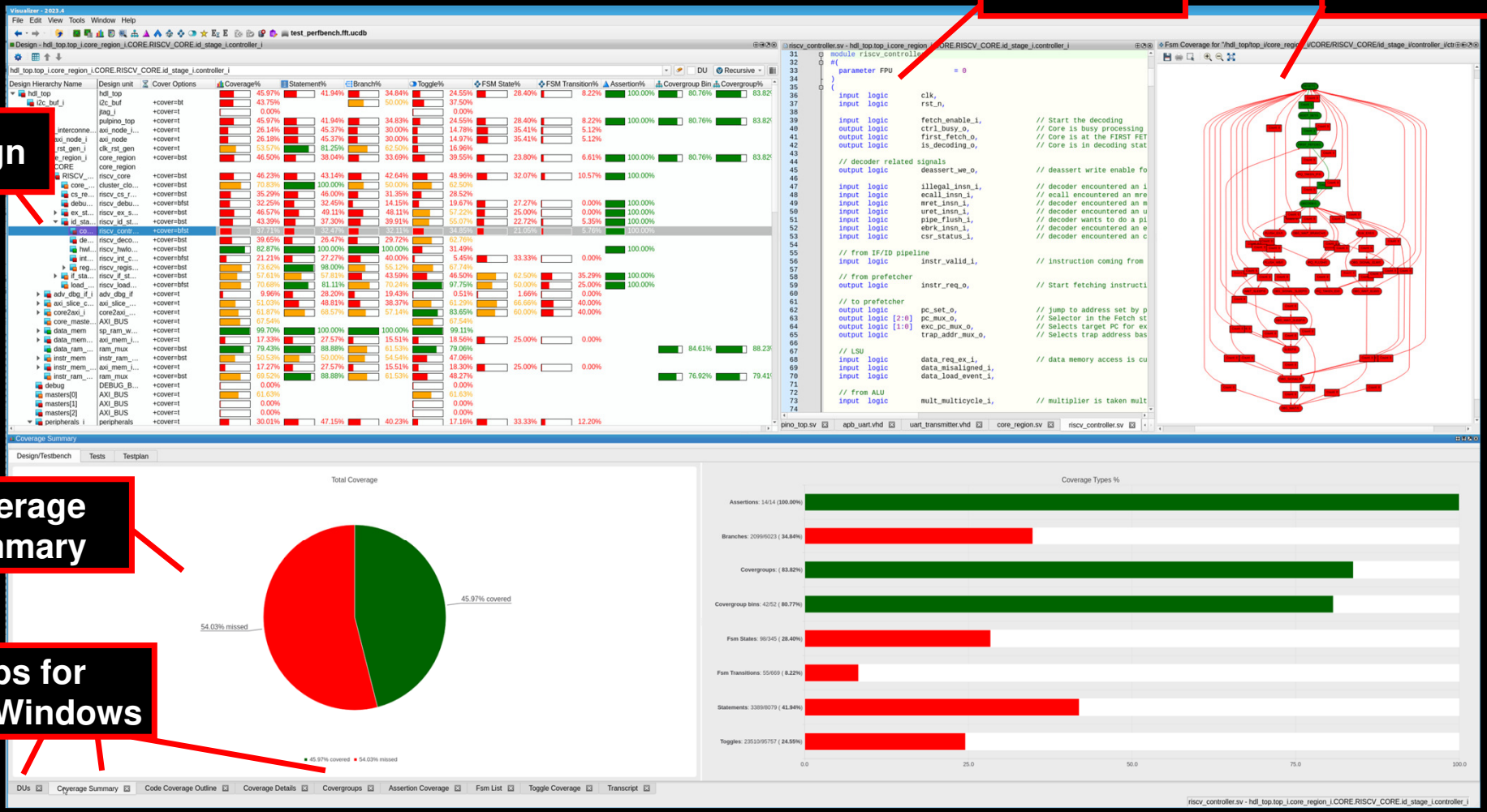
Visualizer

- Unified debug & coverage analysis tool
- View source code and apply exclusions
- A standalone product flow

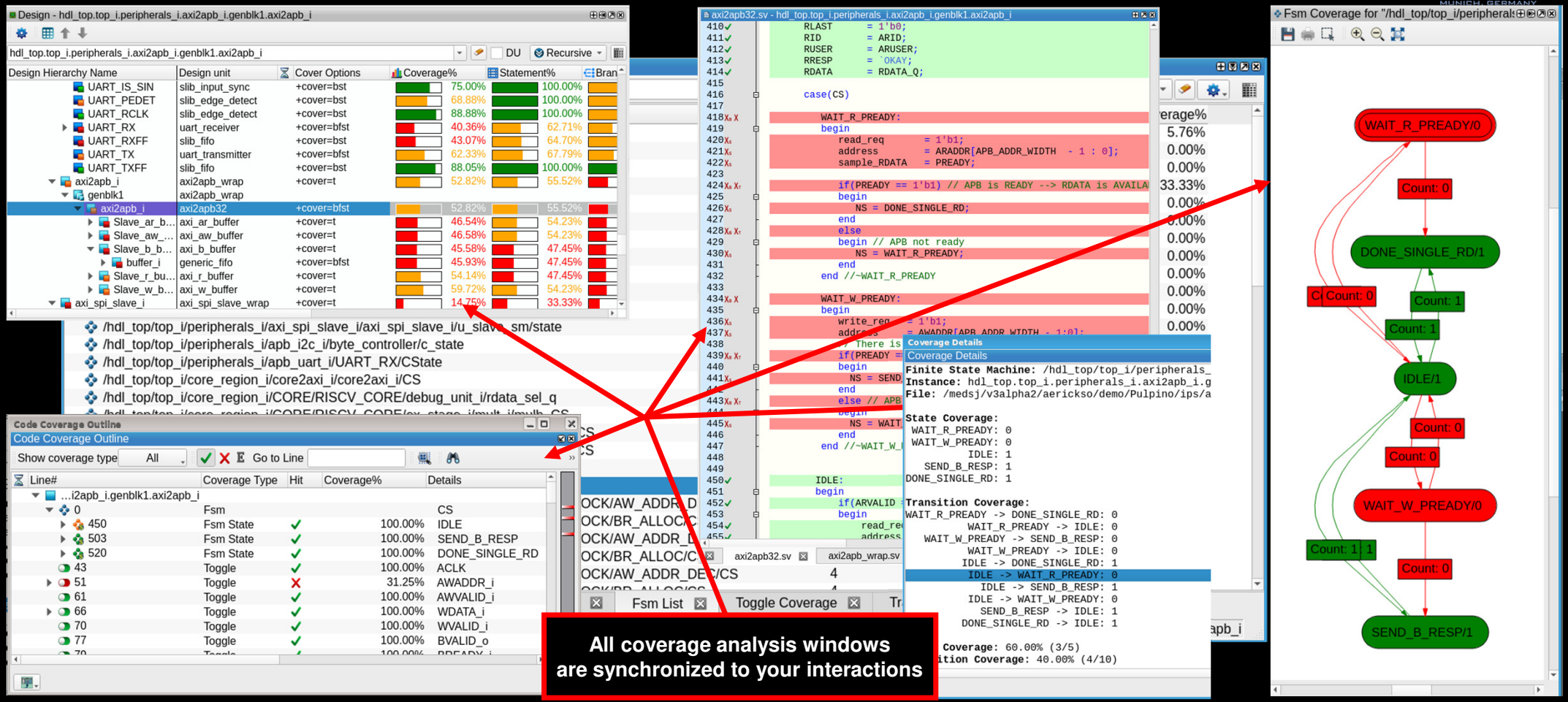
Coverage Analyzer

- Collaborative, web-ready coverage analysis flow
- Provide critical insights with coverage heat-map and network graphs

Visualizer – Unified coverage and assertion analysis



Visualizer – Linked windows update automatically with interactivity



Unified coverage is more than just a merge of databases …

Confidence in verification quality while meeting schedule targets

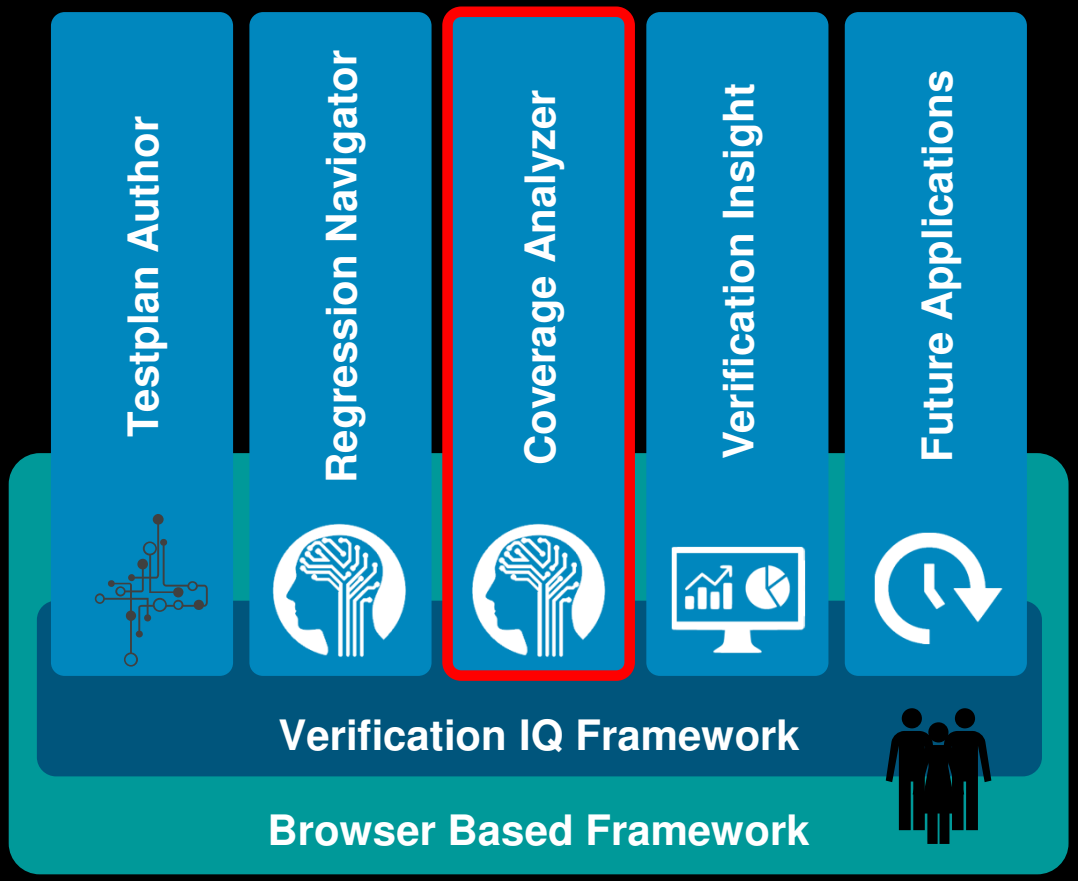
Project-level coverage analysis and traceability for robust verification closure and compliance

HW features verified from IP to system level to minimizing bug escapes into silicon

Real-world SW workloads for comprehensive validation against power/performance specs

Verification IQ – Collaborative Browser Based Solution

Enabling Team-based Data Driven Verification



Process Guide

- Plan & Requirements Driven
- Safety Critical Flows
- Lifecycle Management

Regression Running

- Creation, Execution & Visibility
- Historical Analysis
- ML Acceleration and Efficiency

Coverage Analysis

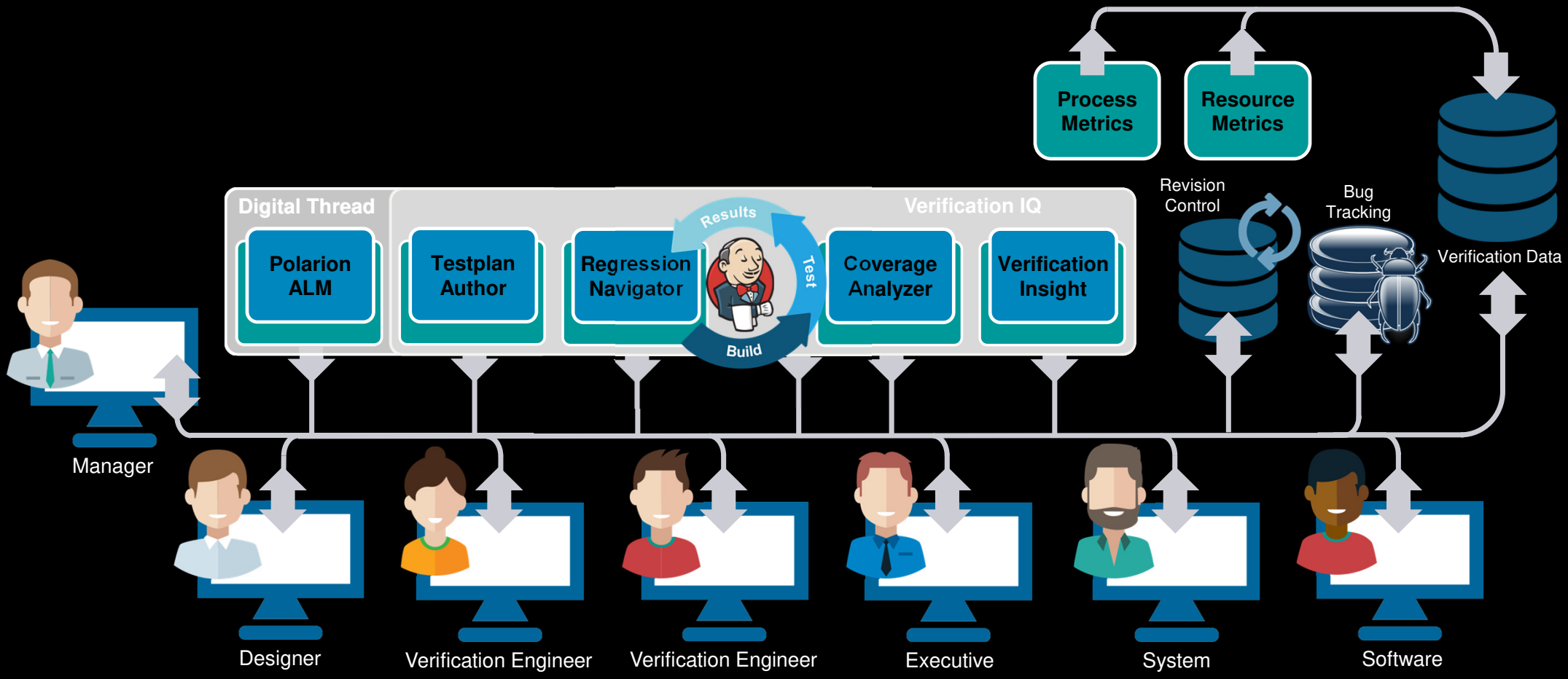
- Code/Functional/Testplan
- Hole, Test & Time Domains
- ML Powered Analytics

Data Analytics

- Metric Platform
- Project Dashboards
- Cross Analytics

Extending Collaboration to Verification Management

Solution Connects all Stakeholders and Domains via the Browser

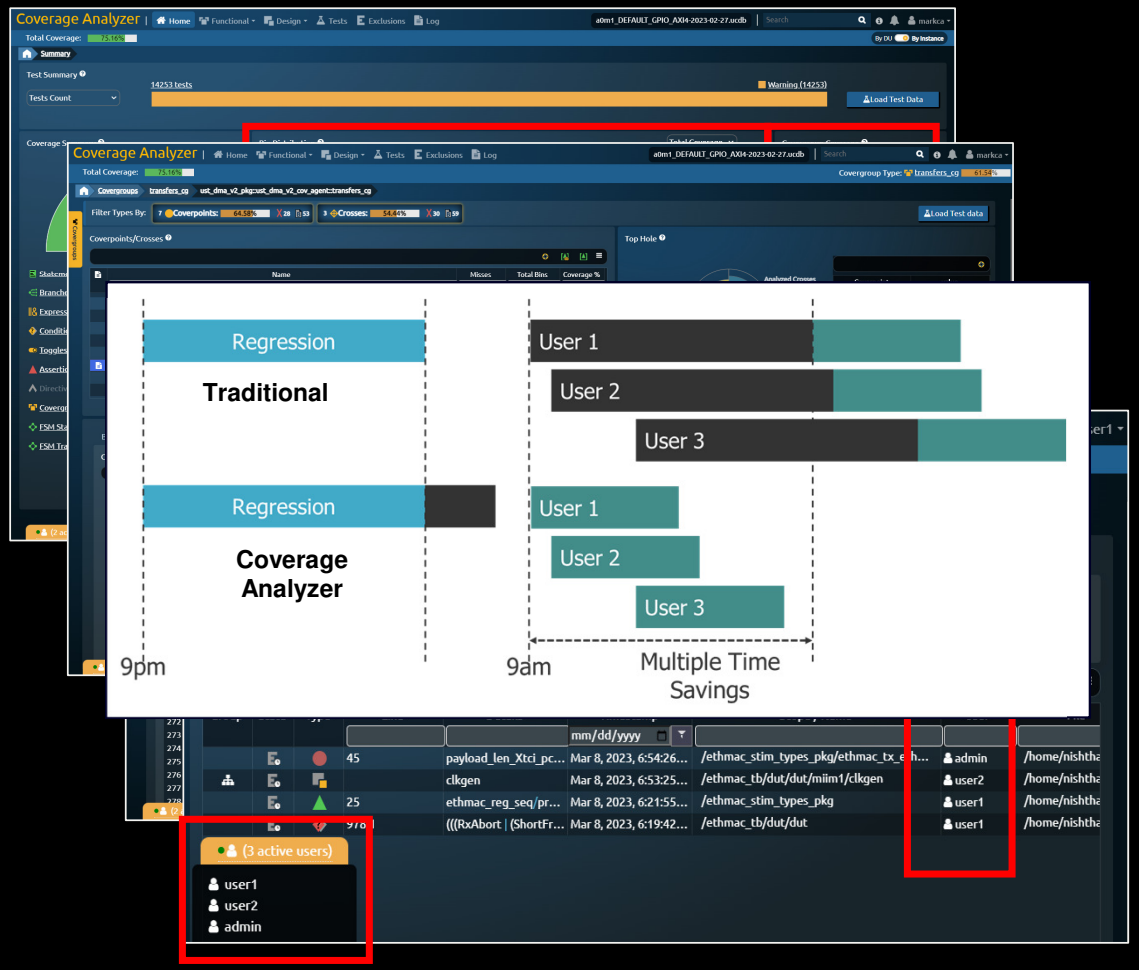


Verification IQ Coverage Analyzer
Collaborative Data-Driven Verification Closure

Coverage Closure Acceleration
- Browser-based Coverage Closure
- Analyze Code, Functional and Testplan Coverage
- Unified support of Multiple Engines

Power of Collaborative Data Analytics
- Improved Coverage Model & Closure Understanding
- Analytical Navigation and Smart Visualization
- Adds User awareness with Questa Exclusion Flows

Application Benefits
- Brings Team Based Analysis to the Questa UCDB
- Unique Analytics for Accelerated Understanding
- ML Accelerated Coverage Closure

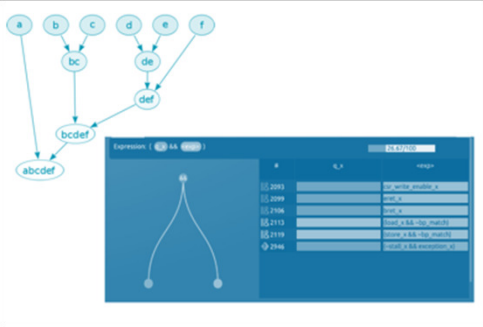


Pattern and Hole Analysis

Guiding Closure with Analytical Visualizations Powered by ML

Expression Coverage

Agglomerative Clustering

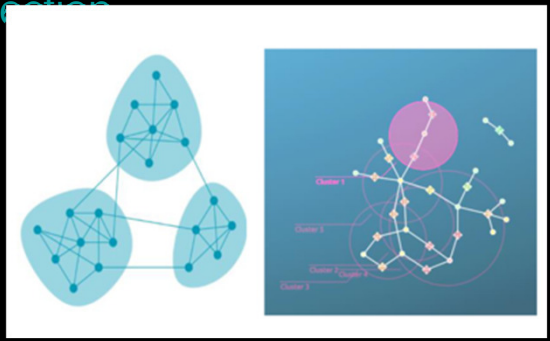


- Pattern Expression Matching
- Common Pattern Relationships
- Powerful Visualization

Benefit: Closure Acceleration

Functional Coverage

Graph Clustering - Louvain detection

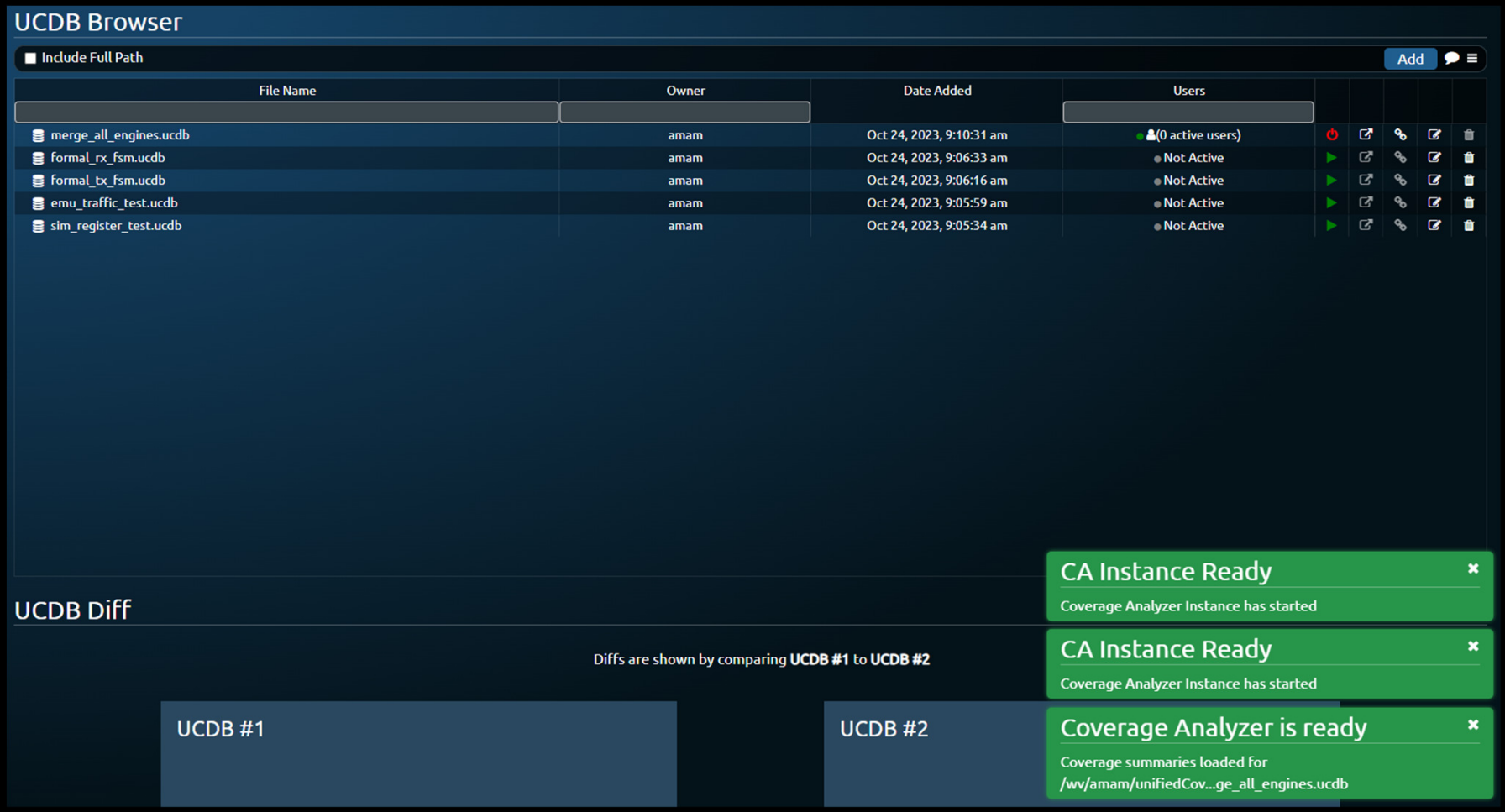


- Cross Multi-Hole Analysis
- Auto-Detects Subcommunities
- Clear Guidance to Holes

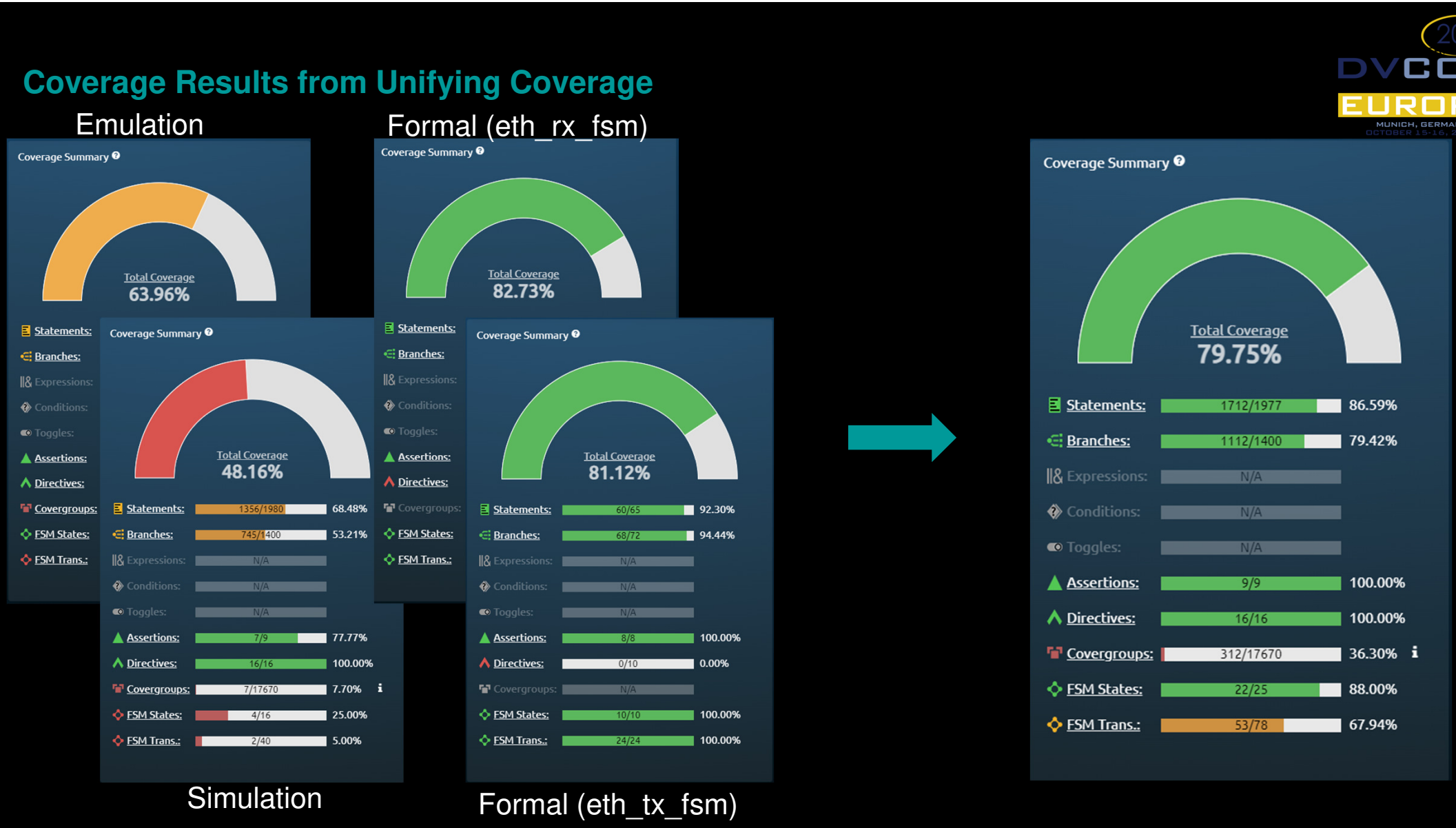
Benefit: Closure Acceleration

Viewing UCDB's in Coverage Analyzer

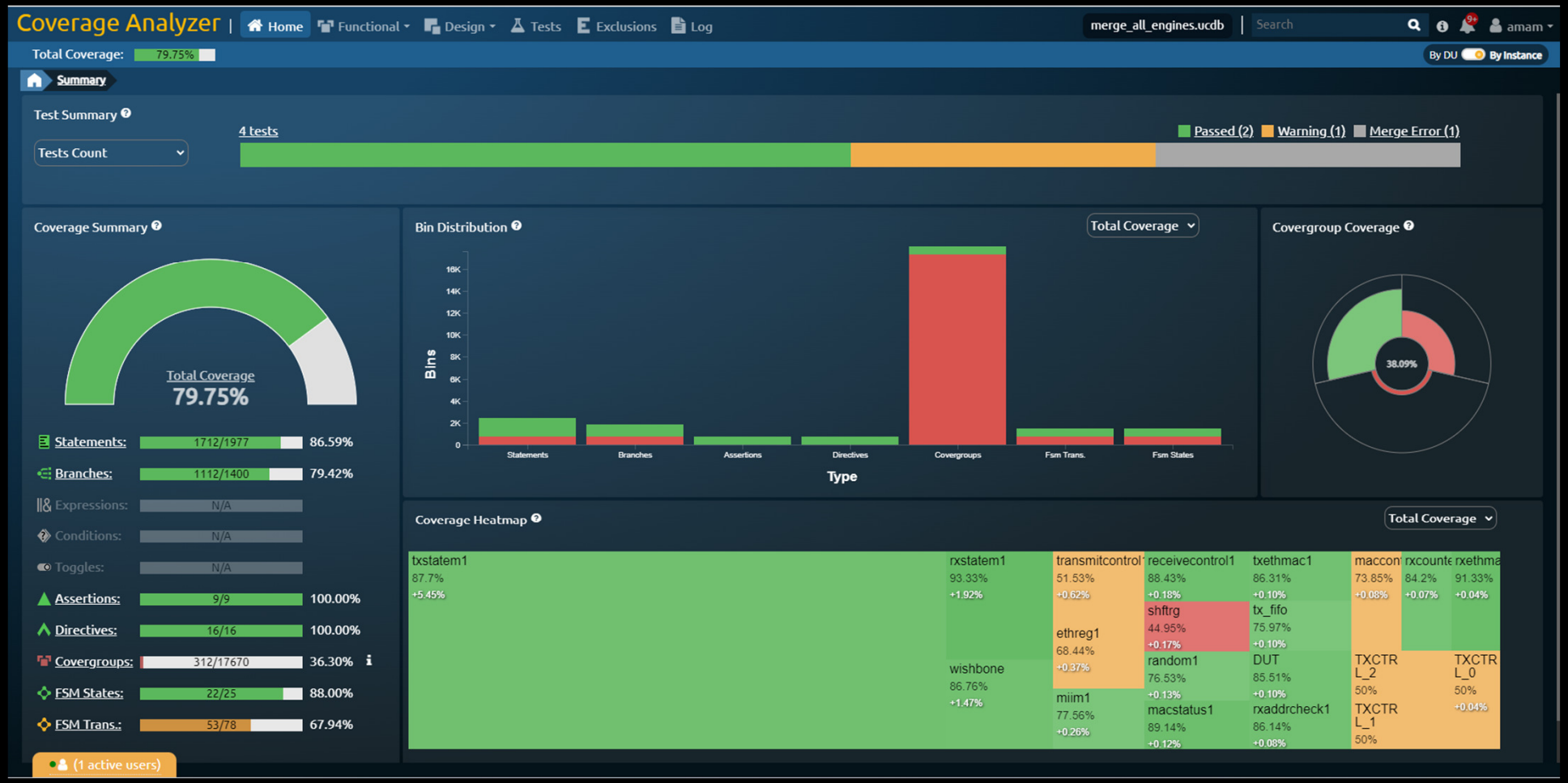
Perform analysis on individual UCDB's or a merged UCDB with unified coverage



Coverage Results from Unifying Coverage

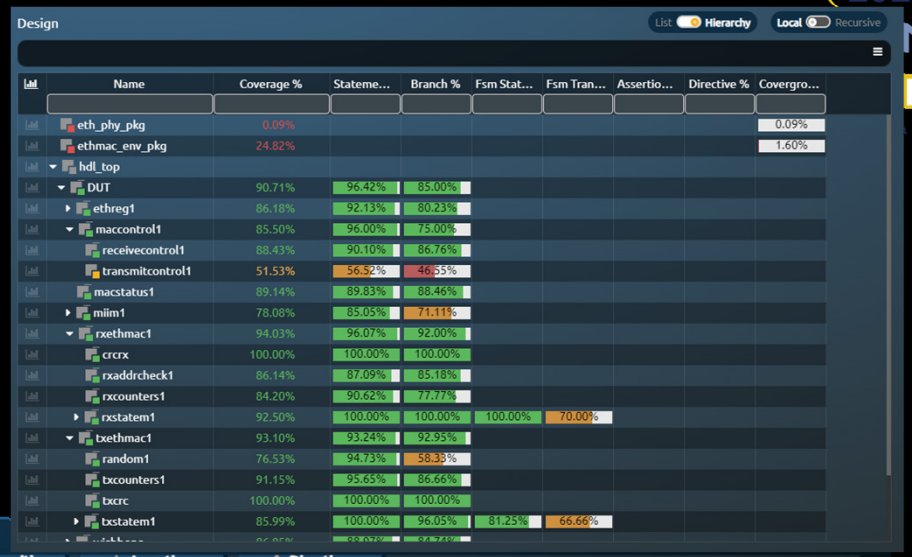


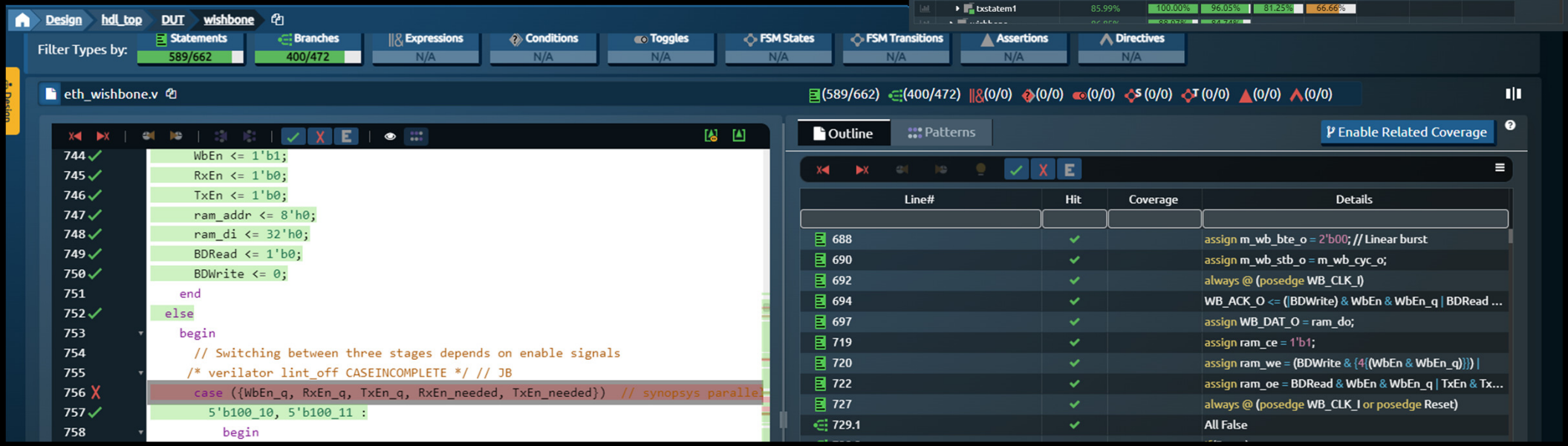
Coverage Visualization and Navigation



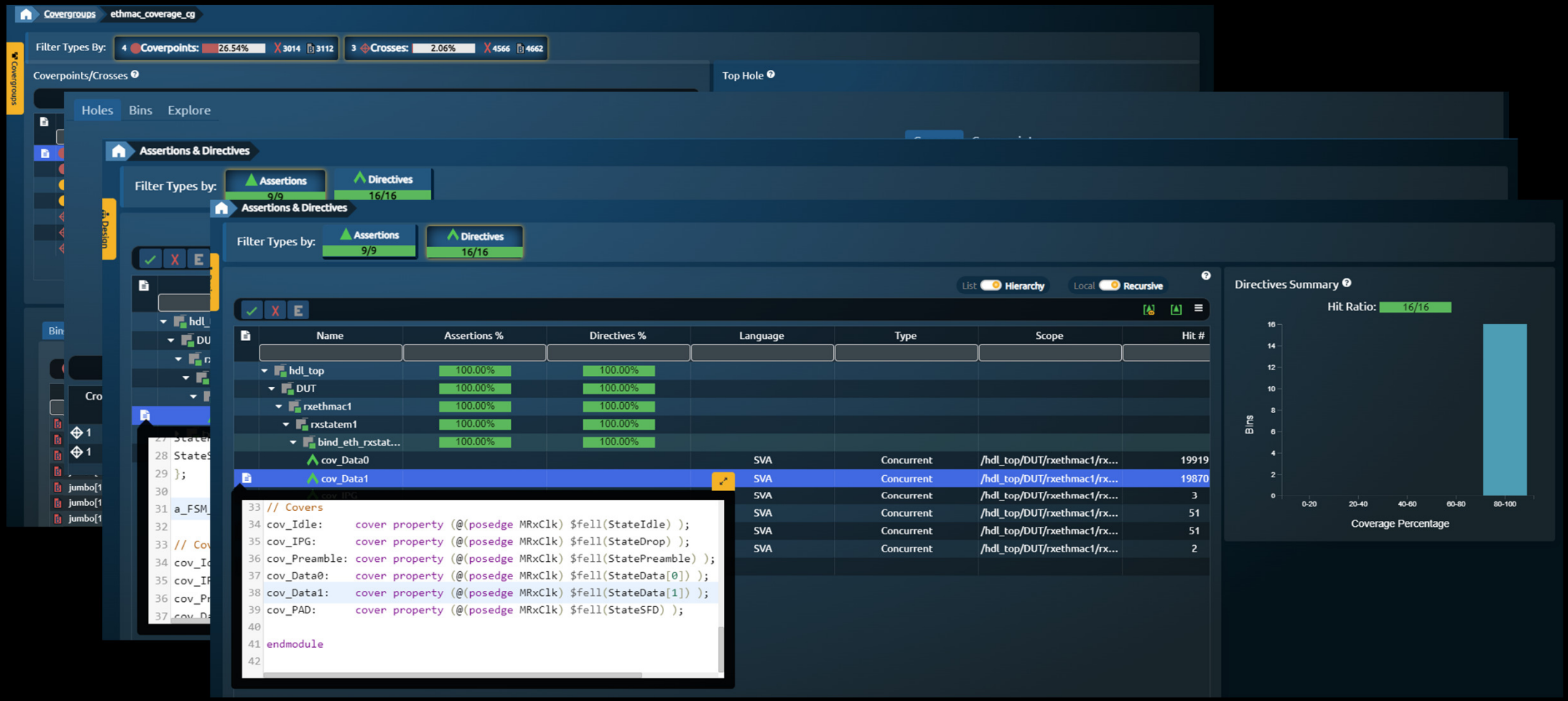
Code Coverage

- View RTL and code coverage that was hit/missed on a line or bin basis
- Filter out different code coverages
- View coverages for each instance/design unit



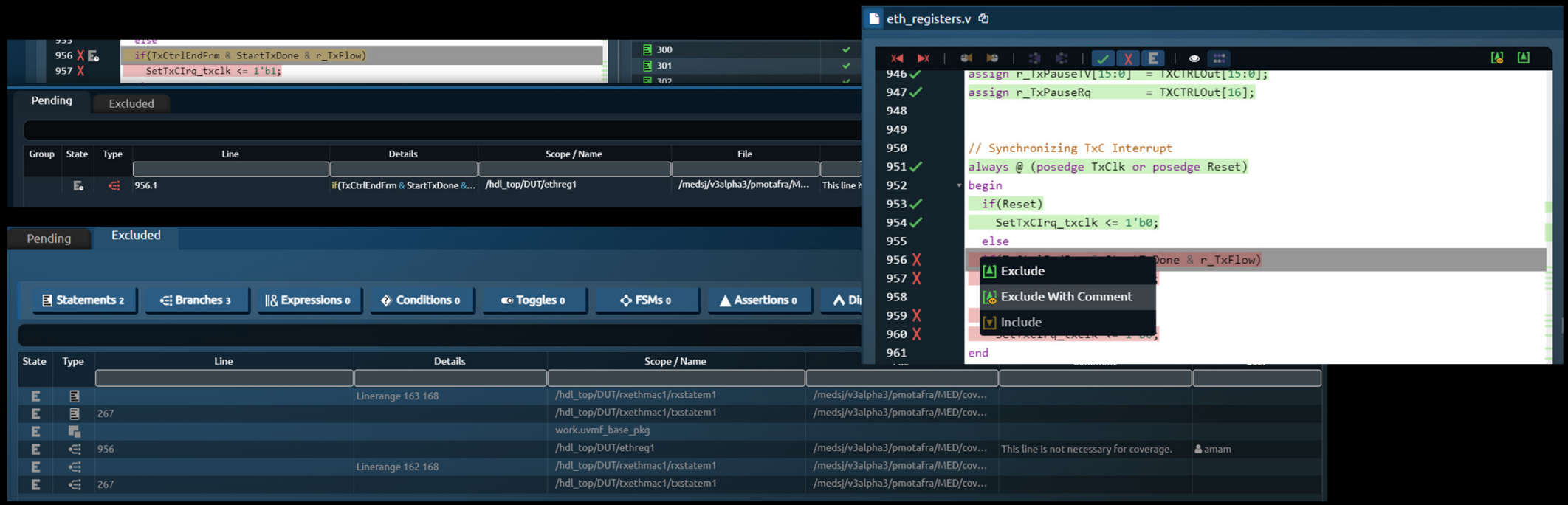


Functional Coverage
Covergroups, Assertions, Directives (Covers)



Coverage Exclusions

- Add exclusions through GUI to any code, functional, or design units/instances
- 2 step exclusion flow provides a way to help prevent invalid exclusions from being applied
- Export and import existing exclusions for future use

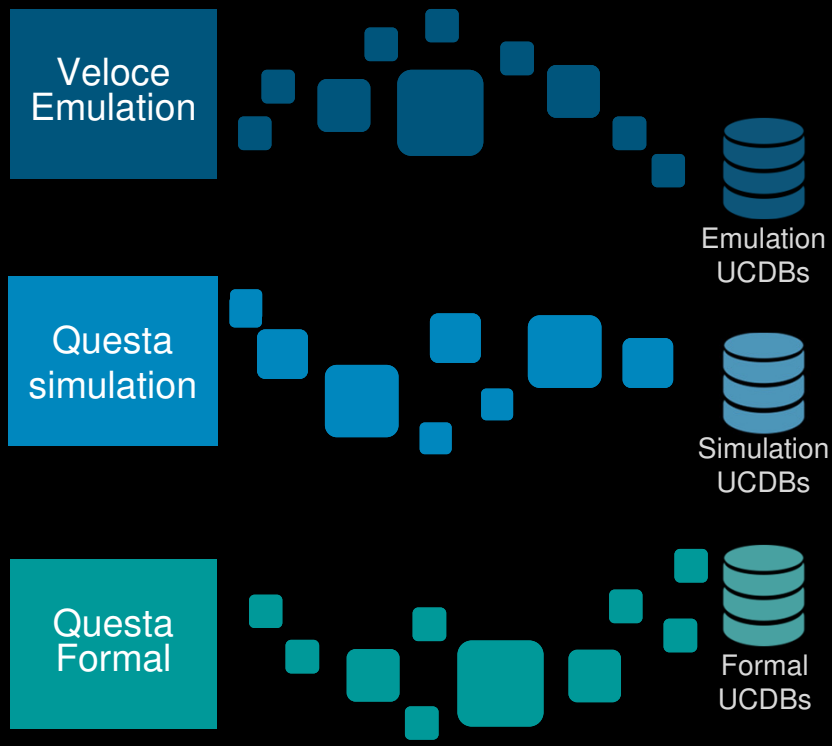


Agenda

- The Case for Unified Coverage
- Simulation & Emulation Coverage
- Formal Coverage
- Coverage Merge
- Unified Coverage Analysis
- Conclusion and Q&A

Coverage - Call On Your Engines

- Coverage is not only simulation:
  - Take coverage credit for your formal proofs
  - Take coverage credit for emulation long / software payloads
  - Generate the same database from any source of coverage
  - Merge seamlessly within a unified environment



Thank You