

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



Agenda

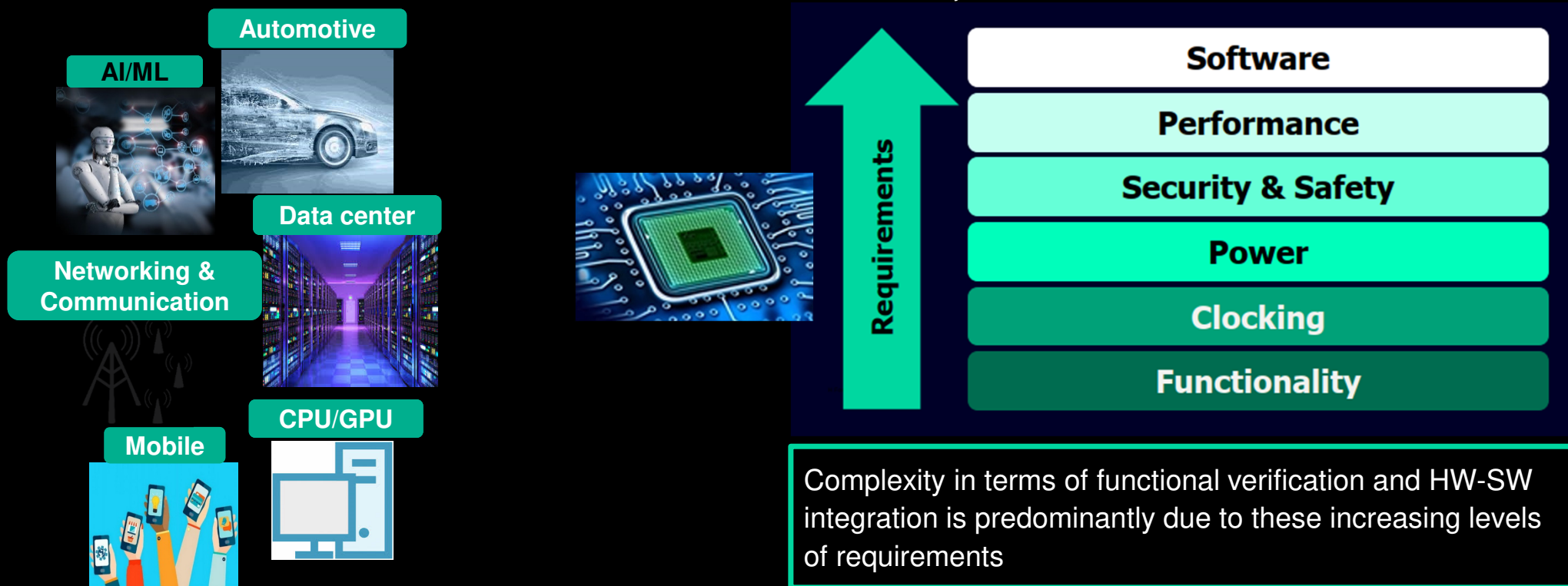
- **The Case for Unified Coverage**
- **Simulation & Emulation Coverage**
- **Formal Coverage**
- **Coverage Merge**
- **Unified Coverage Analysis**
- **Conclusion and Q&A**

Agenda

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

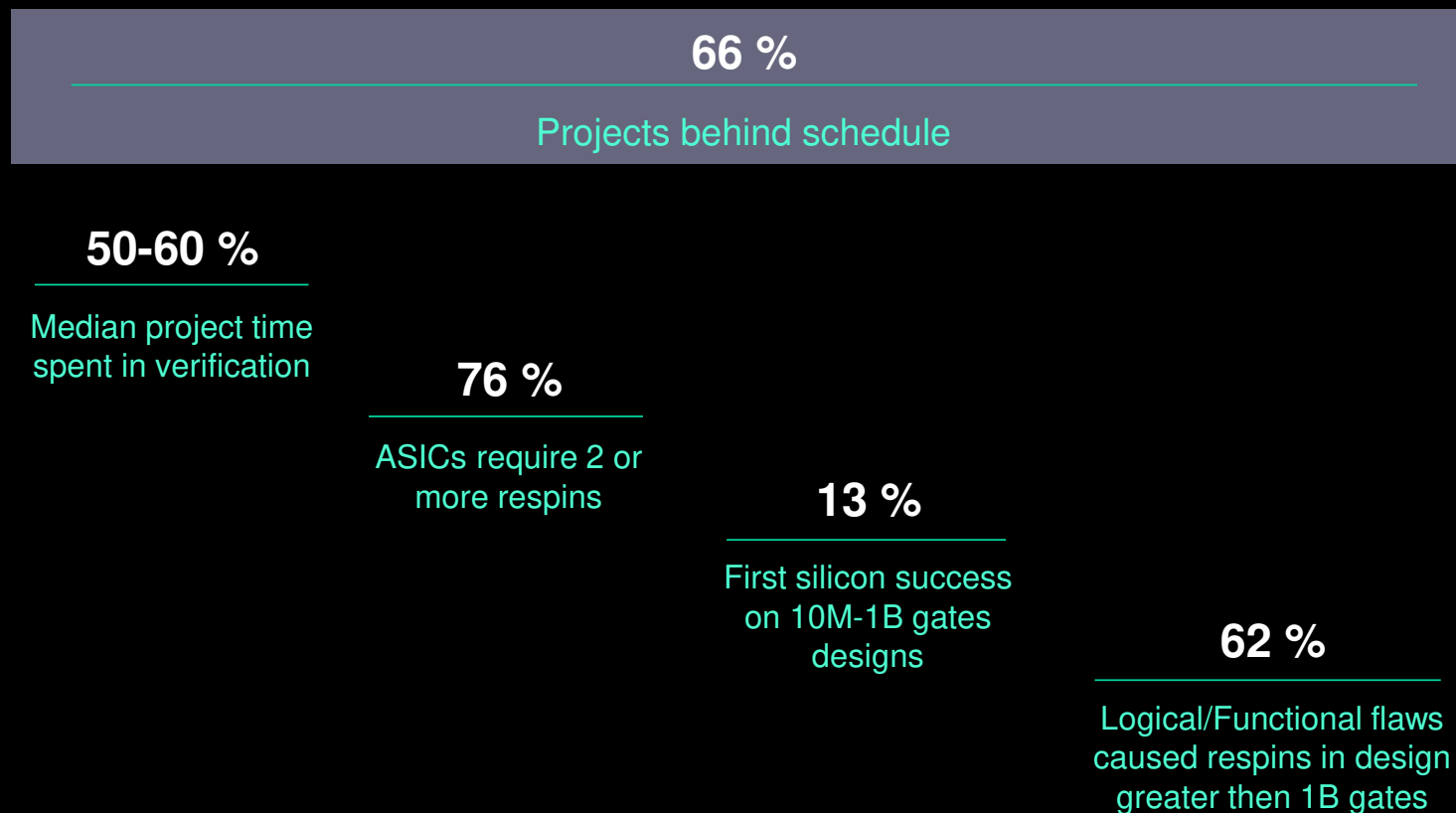
Market dynamics drive new use-cases and requirements

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



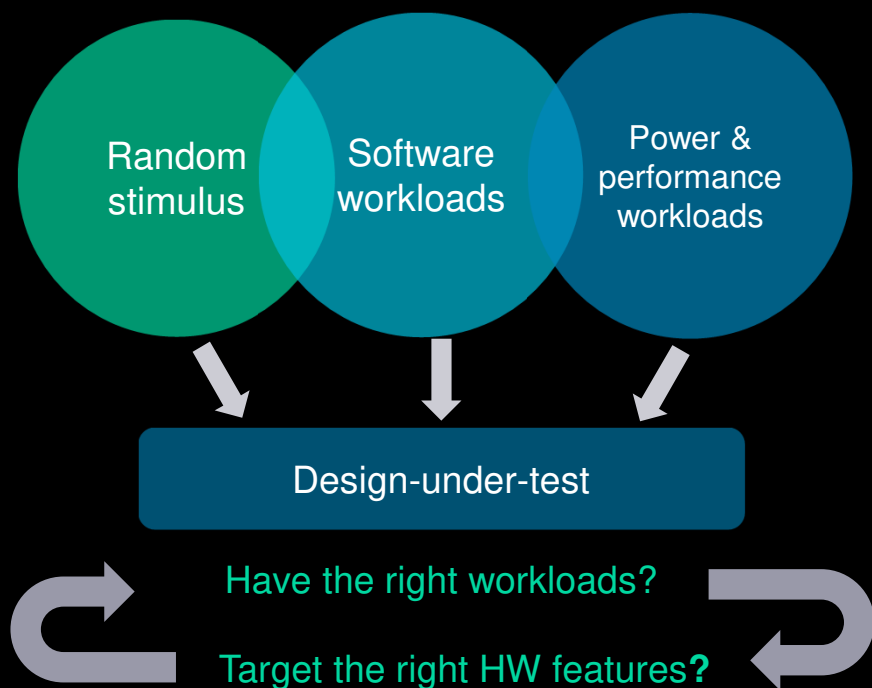
Project impact

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



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

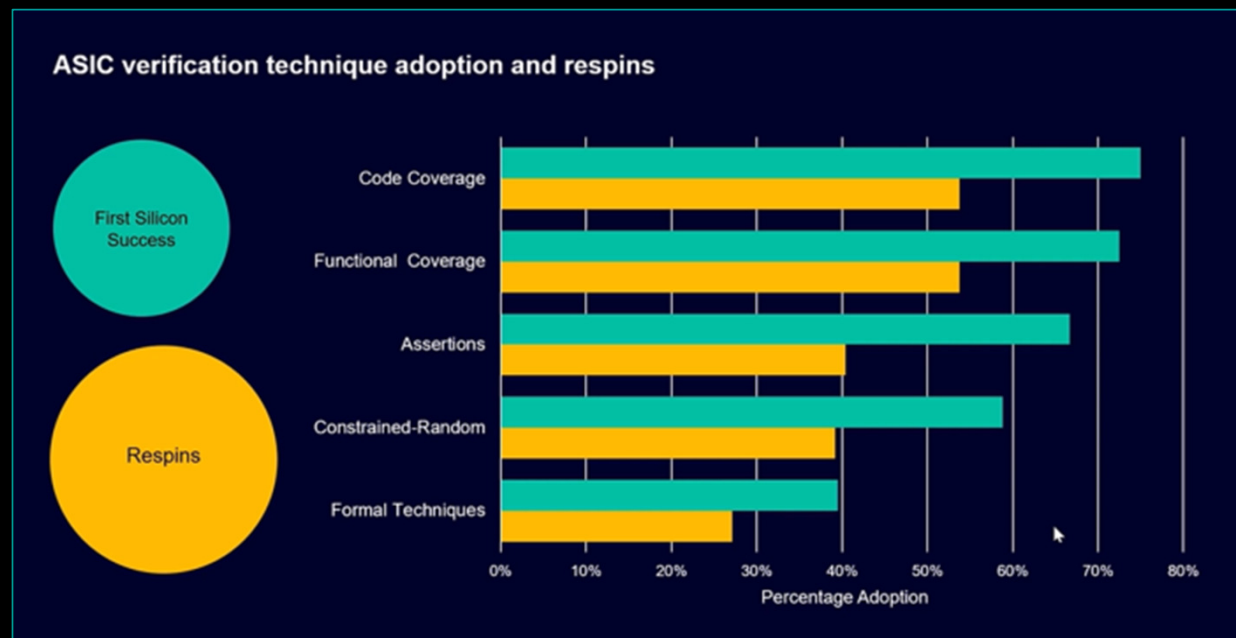
Functional coverage

- 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

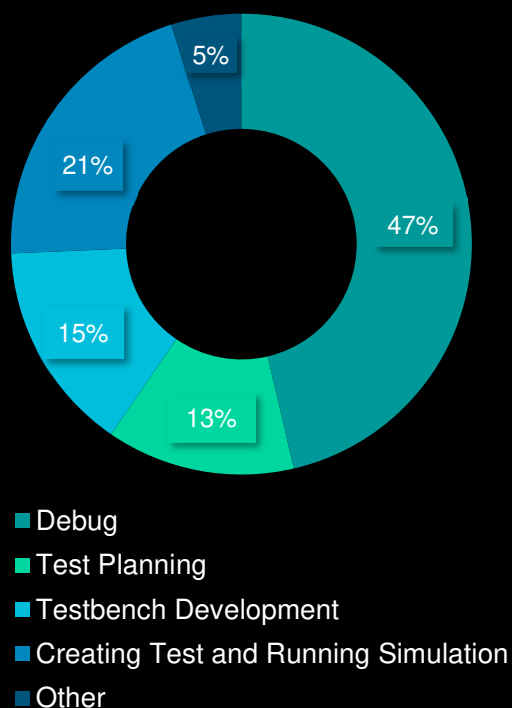


Leverage coverage & assertions to avoid bug escapes from block to system level

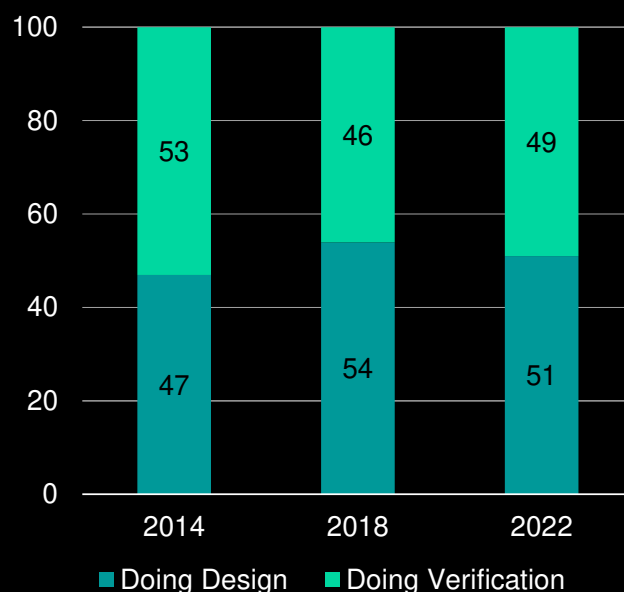
ASIC/FPGA verification effort

Coverage & assertion-based verification are growing requirements in emulation

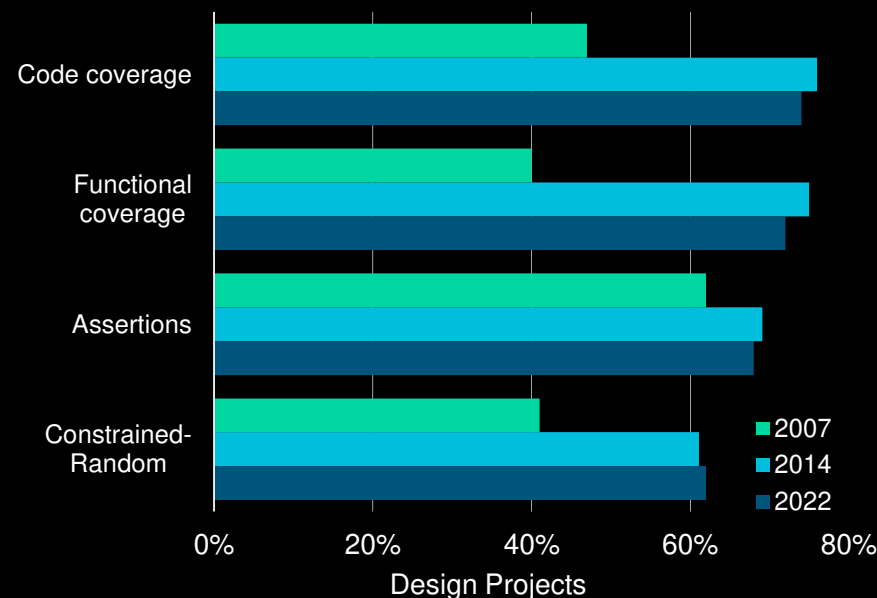
Where IC/ASIC **verification engineers** spend their time



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



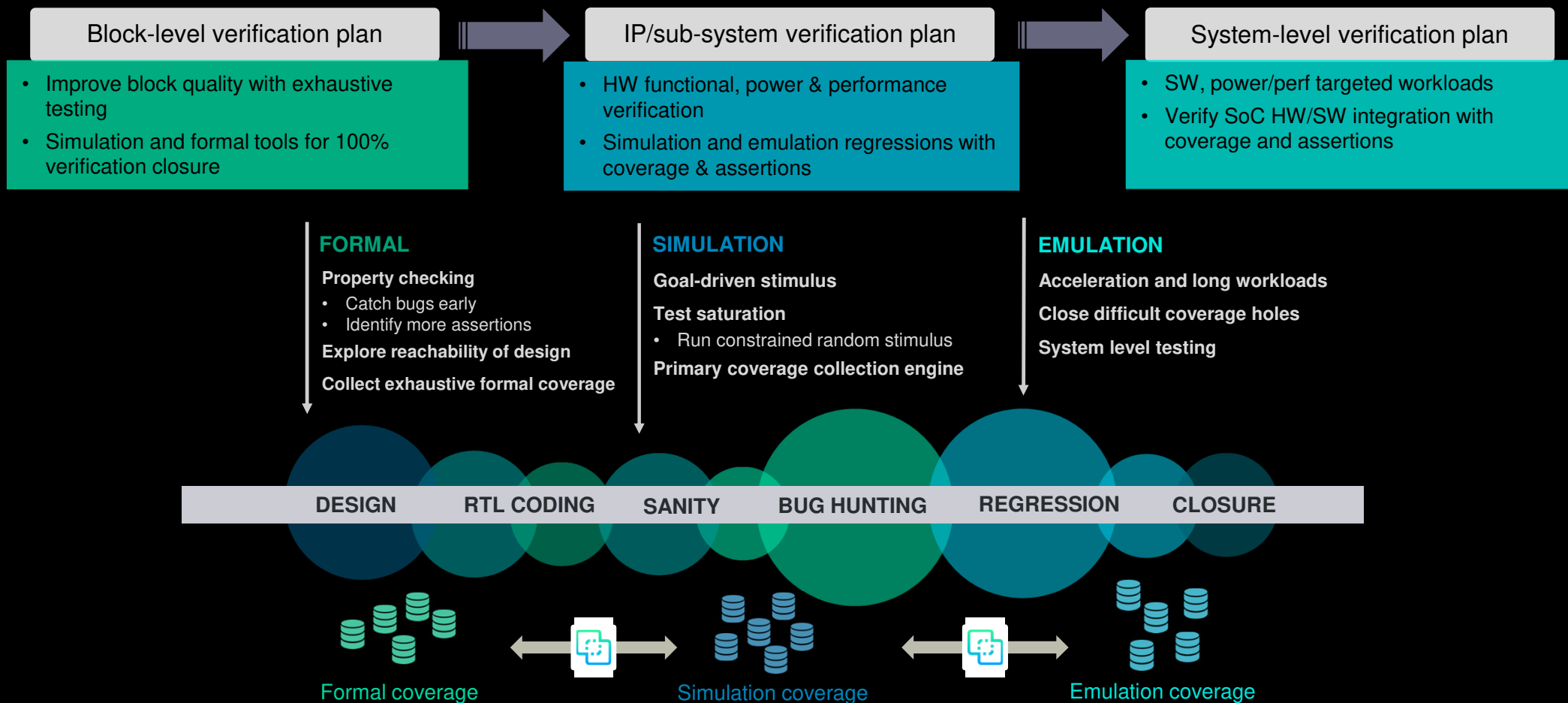
ASIC adoption of dynamic techniques



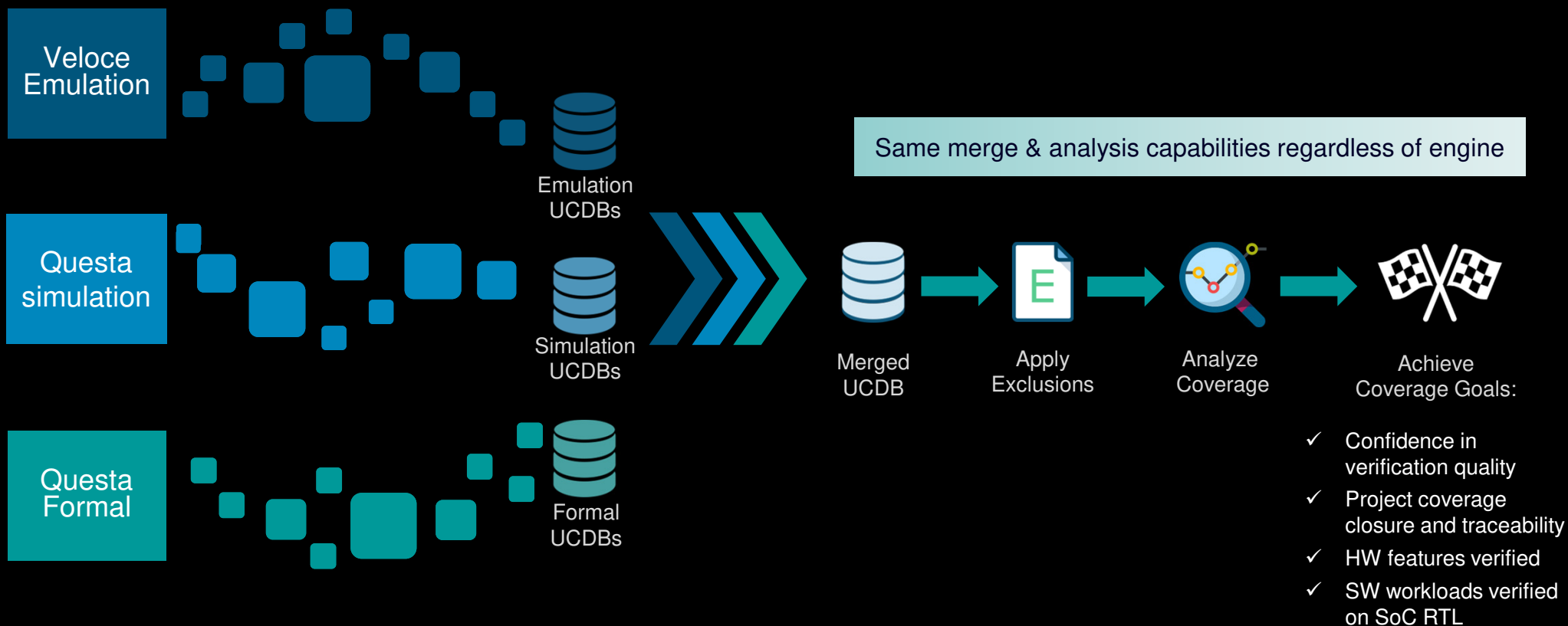
Source: Wilson Research Group and Mentor, A Siemens Business, 2022 Functional Verification Study

An SoC level multi-verification phase and tool challenge

Adopt the right verification engine for effective verification sign-off



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

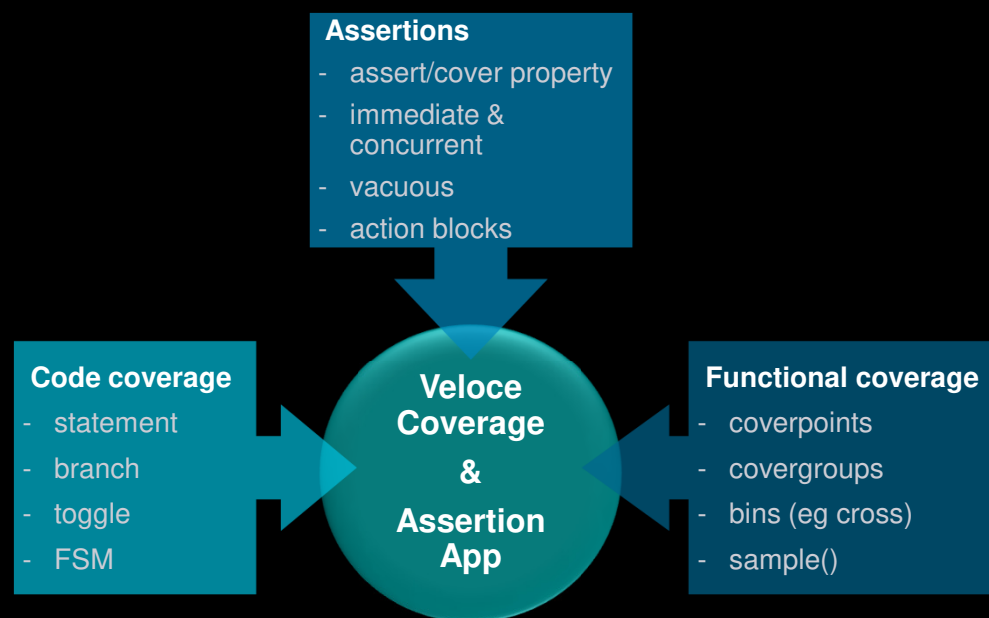
Functional Coverage

- 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

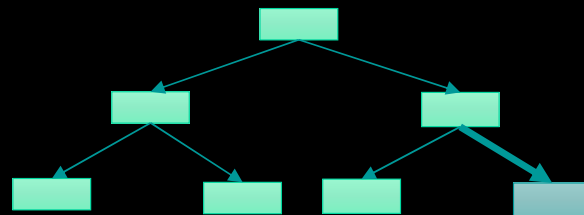
Code Coverage Metrics Supported by Veloce Coverage App

Statement Coverage



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

Branch Coverage

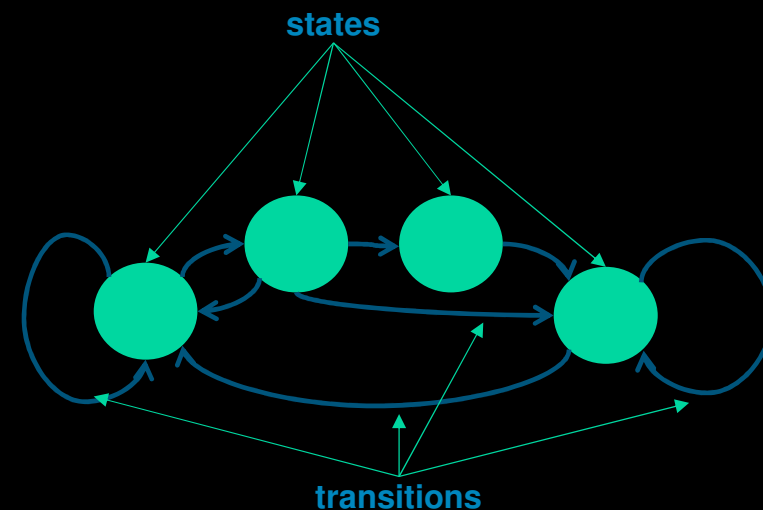


Check if all branches in always blocks (clocked/combinational) are exercised

- if/case/conditional operator

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

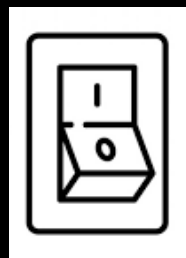
FSM Coverage



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 **Unified Coverage Interoperability Standard (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	✓	✓	✓	✓	✓	✓	✓	✓

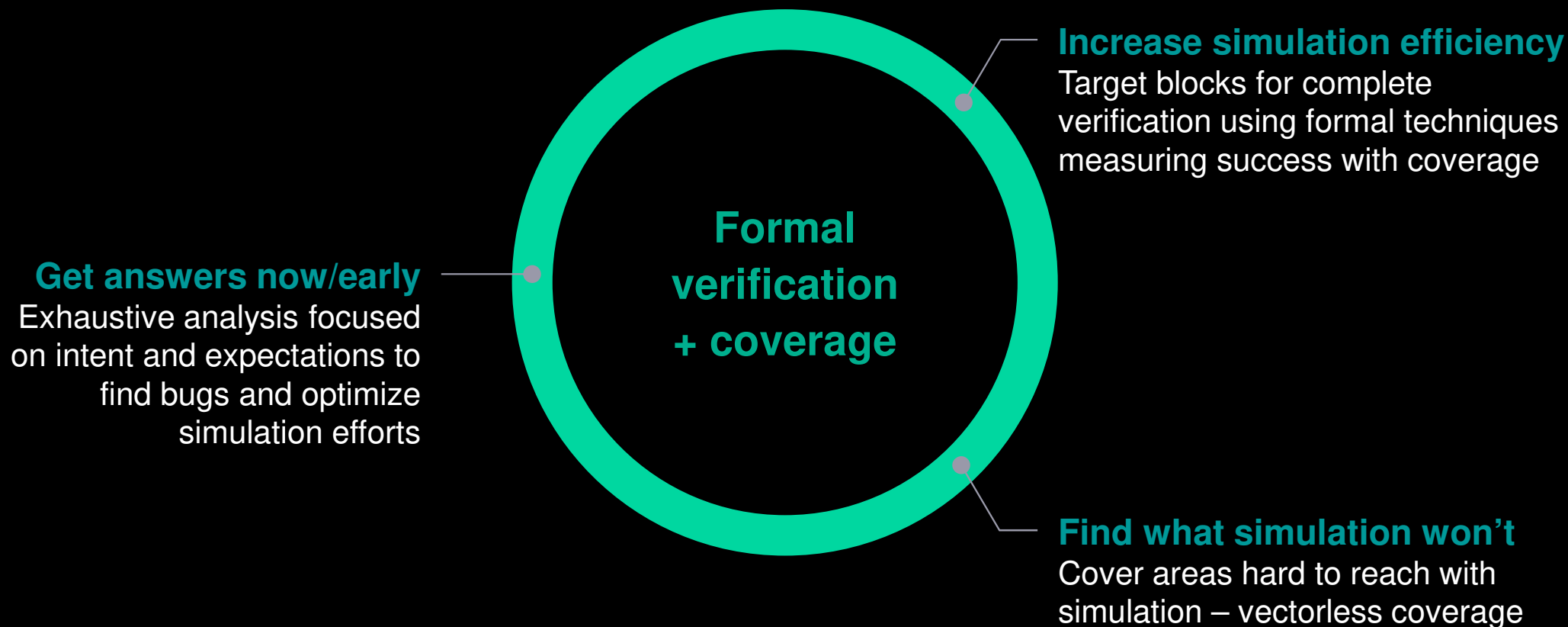
- ✓ Supported
- ⌚ Future Support

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

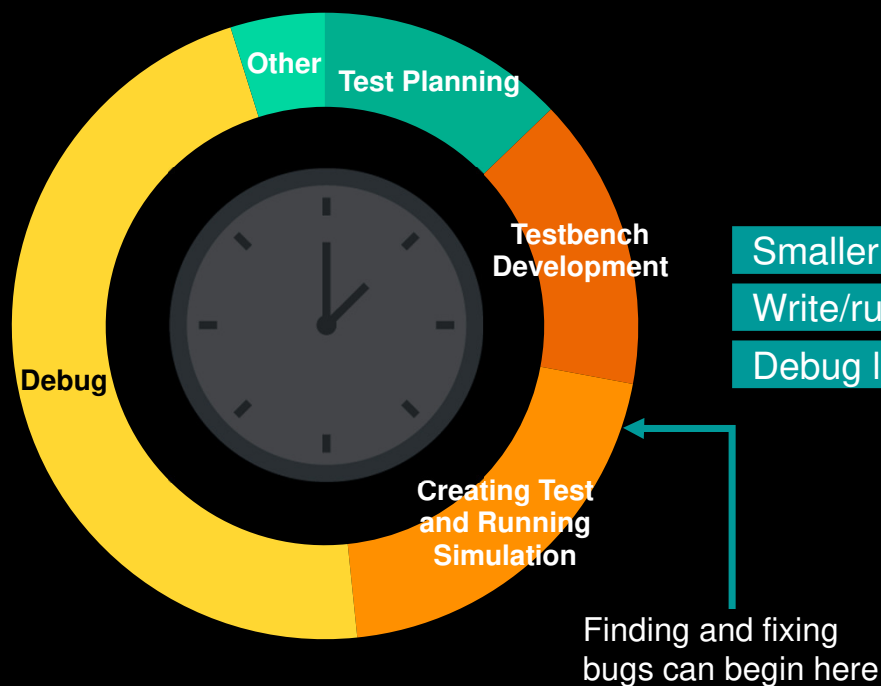
Complementary with simulation, deliver unique abilities and acceleration



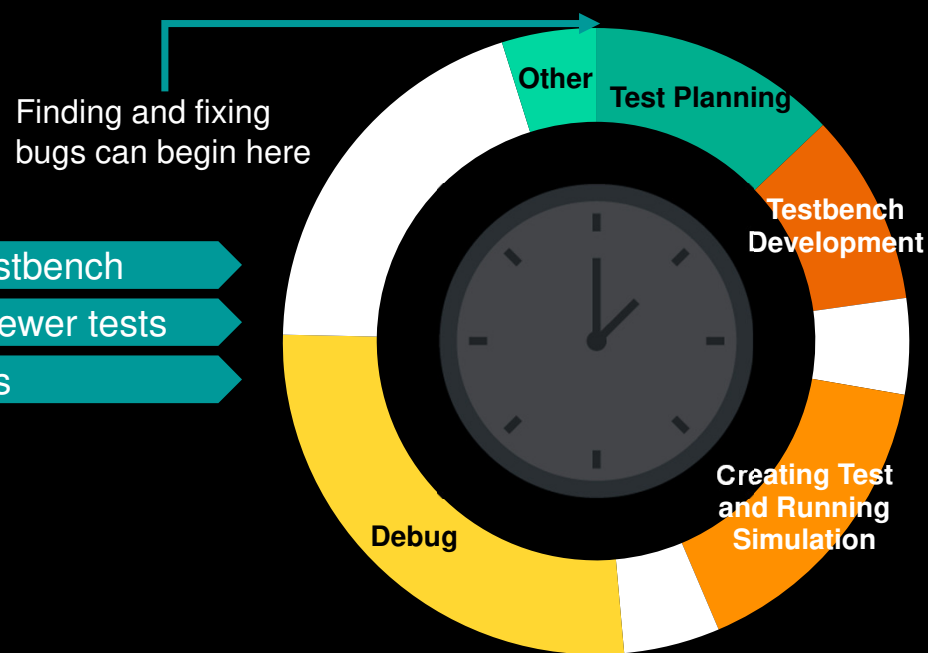
Formal adoption does more than increase simulation efficiency

Finding and fixing bugs before testbench increases team agility

Without Static & Formal



With Static & Formal



Smaller testbench
 Write/run fewer tests
 Debug less

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

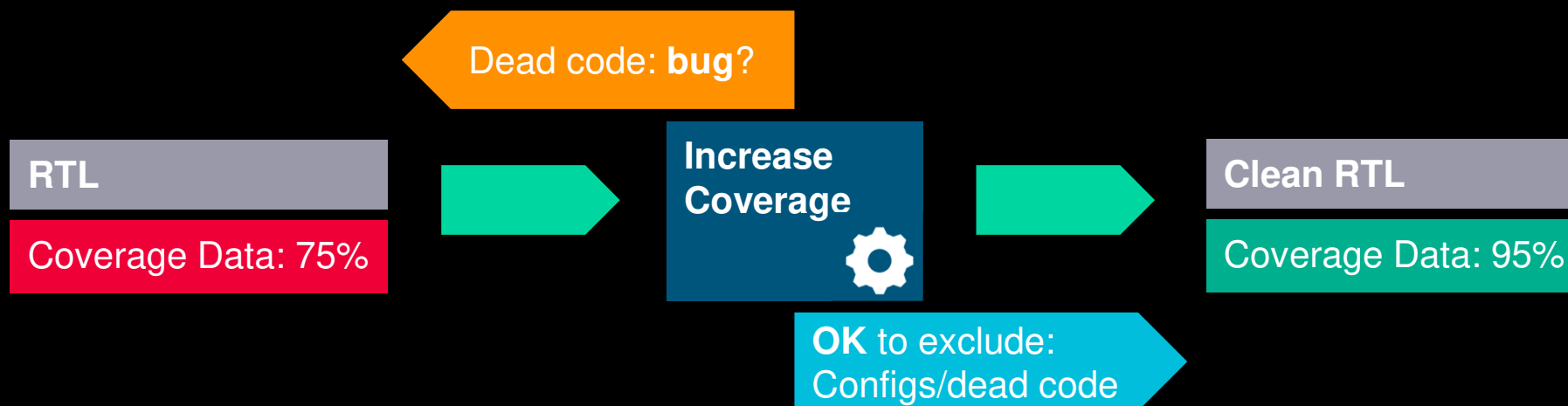
Used to accelerate simulation coverage closure

- 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

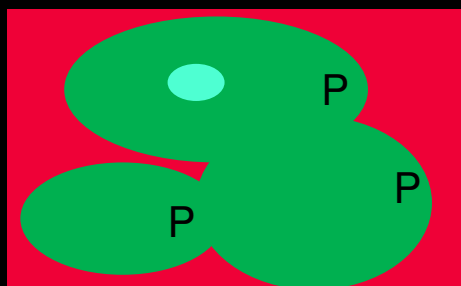
P

Formal signoff code coverage

Based on property proof coverage

Familiar code coverage types with simulation

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



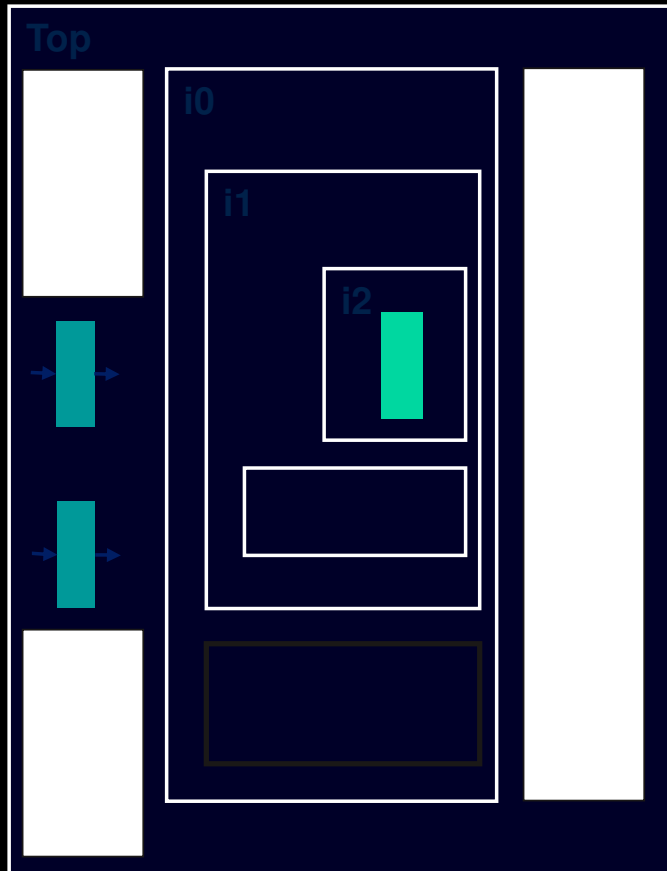
Based on two concepts:

- 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

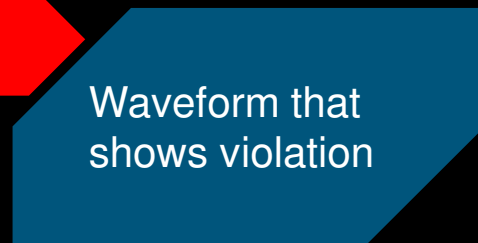
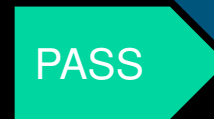
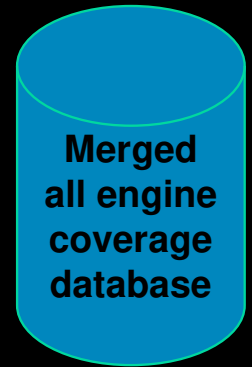
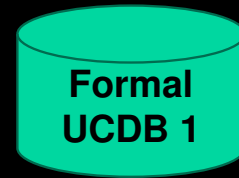
Reachable	Observable	Coverage Result
Yes	Yes	Covered
Yes	No	Uncovered
No	-	Excluded
Inconclusive	-	Uncovered

Questa Verify Property

Enables scalable formal coverage for sub blocks verified with property checking

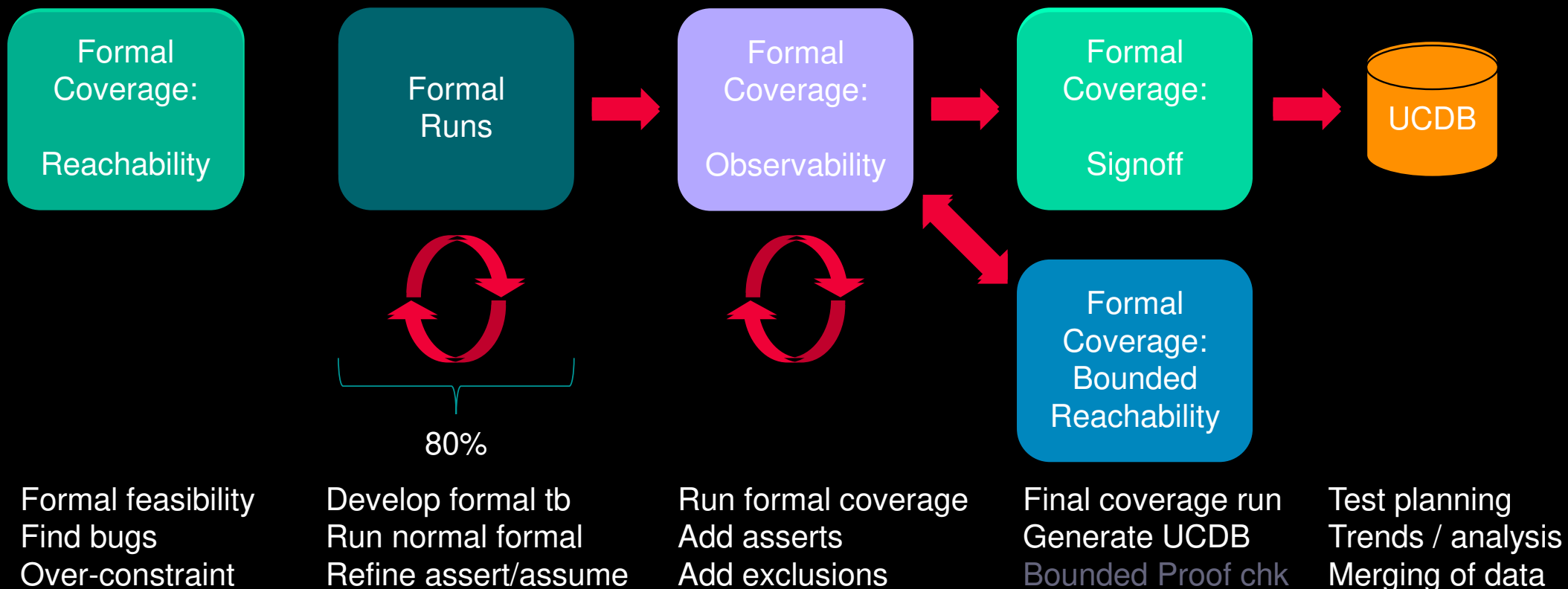


Specification (property) is equivalent to the DUT?



Verify Property formal coverage methodology

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



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;
E_u 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)

Look for Red

Use to goto line for uncovered

What properties can I add to verify this code? Add properties and rerun

Iterate until 0

Coverage Summary - top

Signoff
 Summary For 2 Properties (Recursive)

Coverage	Total	Uncovered	Excluded	Covered (Proofs)	Covered (+Inconclusives)
Branch	66	6	17	7 (14.3%)	43 (87.8%)
Condition	22	2	7	3 (20.0%)	13 (86.7%)
Expression	29	14	1	0 (0.0%)	14 (50.0%)
FSM State	3	0	0	0 (0.0%)	3 (100.0%)
FSM Transition	6	0	3	0 (0.0%)	3 (100.0%)
Statement	75	15	16	7 (11.9%)	44 (74.6%)
Toggle	378	60	37	80 (23.5%)	281 (82.4%)
Covergroup Bin	13	10	0	0 (0.0%)	3 (23.1%)
Total	592	107	81	97 (19.0%)	404 (79.1%)

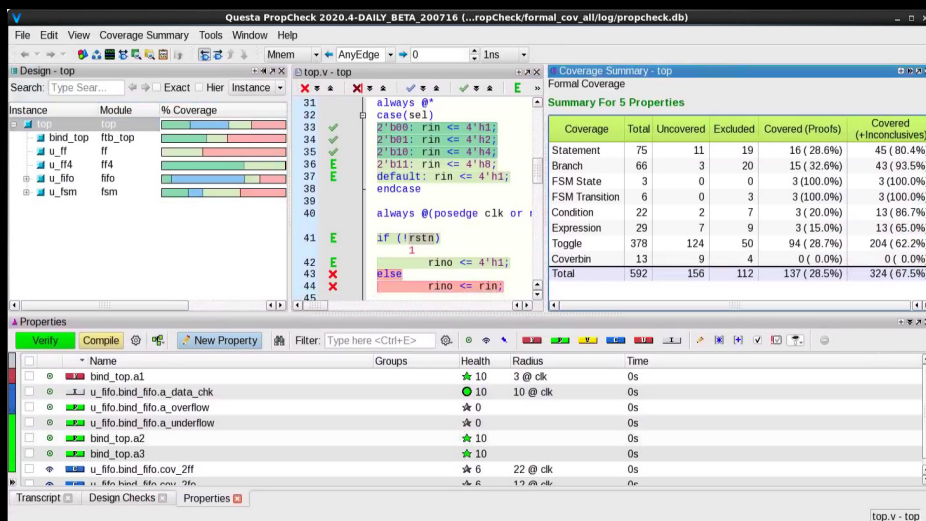
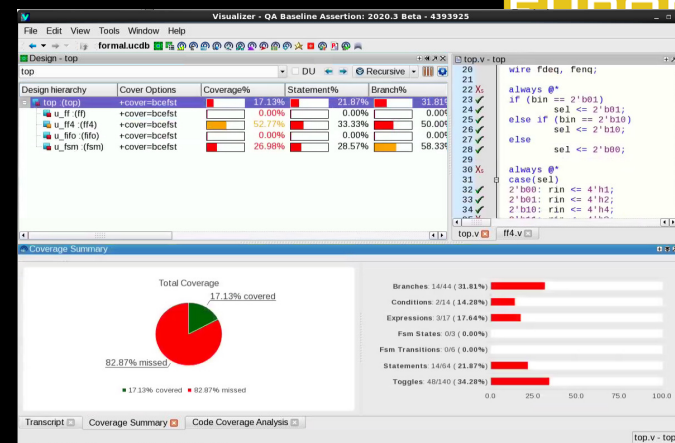
Coverage tracking and analysis Questa Verify Property

Coverage Analyzer and Visualizer supporting views

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

Formal coverage workflow

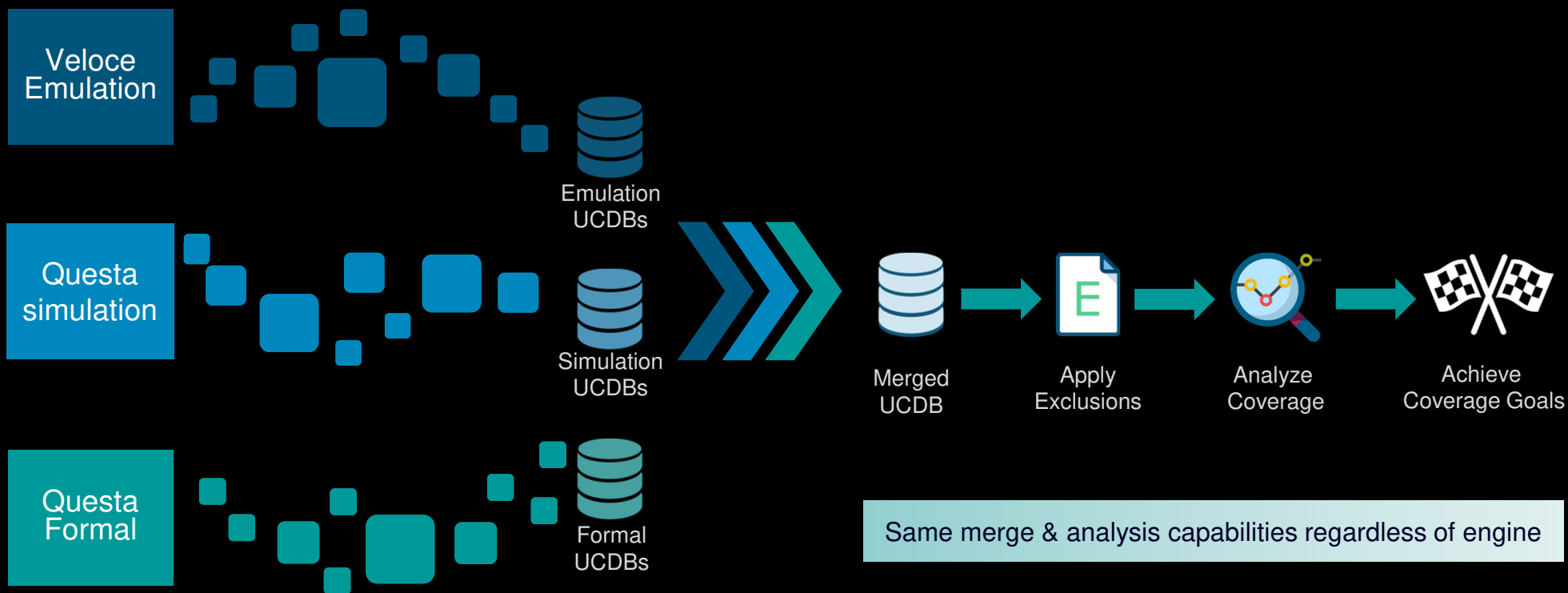
- Debug done in Verify Property
- Tracking/Management same as simulation/emulation using the UCDB



Agenda

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

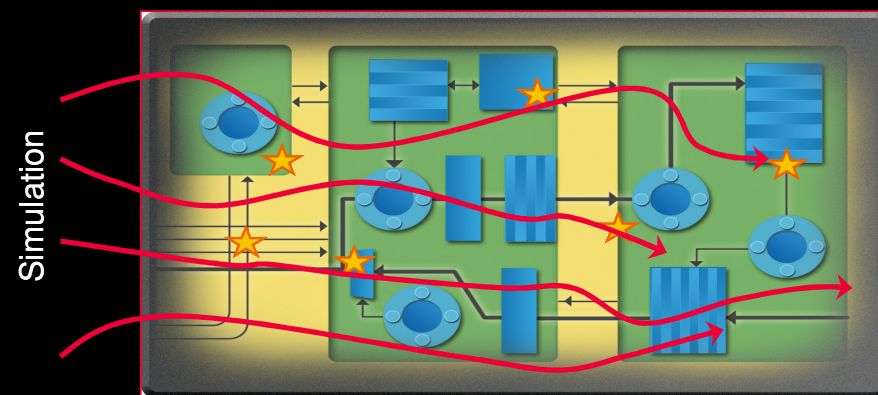
Combining Questa Simulation, Questa Formal and Veloce emulation coverage



Comparing formal coverage to simulation/emulation coverage

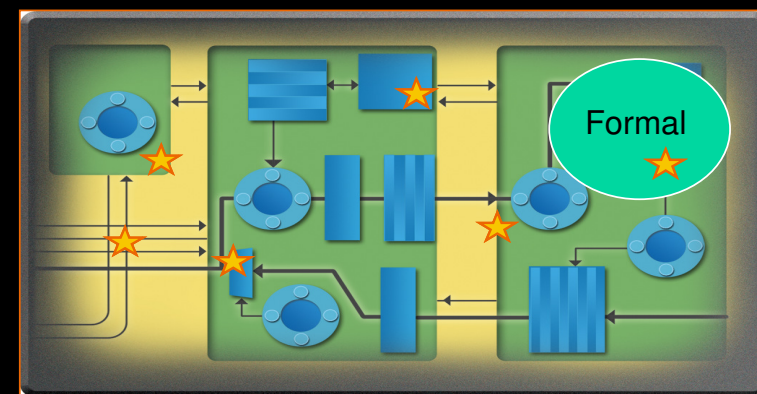
Simulation Code Coverage = Vector Coverage

- 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

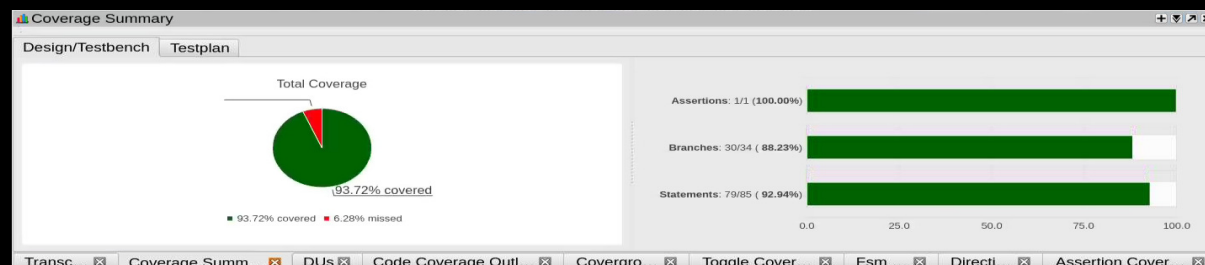
Formal coverage from UCDB (s,b)

- 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

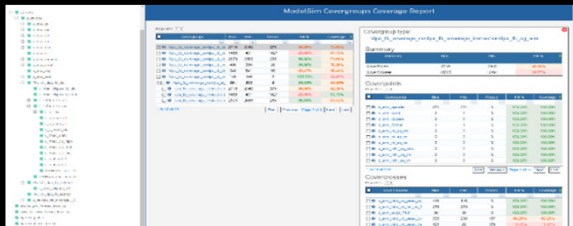
Signoff

- 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



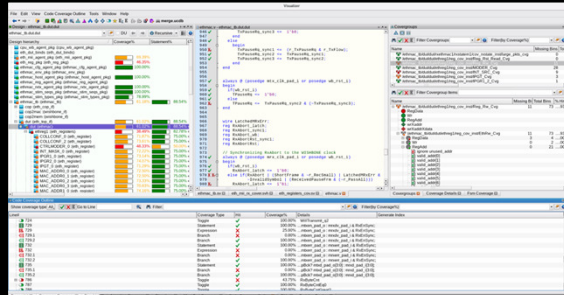
Local Instance Coverage Details (83.74%)

Coverage Type	Blk	Hits	Misses	Coverage
Assertions	105	105	0	100.00%
Covergroups	7	na	na	67.47%
Covergroup Bins	740194	78834	661360	10.62%

```

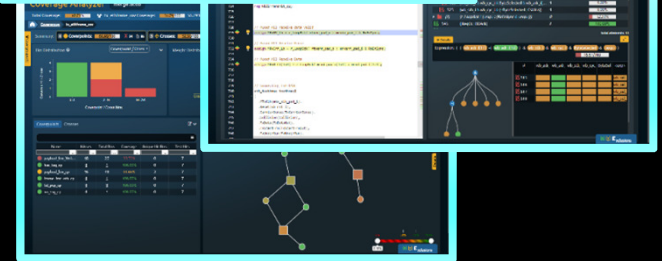
Start Time: 07:28:15
Cover report -by:
Coverage Report Summary Data by instance
Instance: /dpu_tb_coverage_cu
Design Unit: work.dpu_tb_coverage_cu
Enabled Coverage      Active  Hits  Misses % Covered
-----
Cover Group Types      7      na   na    67.47
Coverpoints/crosses    1238   na   na    58
Covergroup Bins        740194 78834 661360 10.65
Assertions              105    105  0     100.00

TOTAL COVERGROUP COVERAGE: 67.47% COVERGROUP TYPES: 7
TOTAL ASSERTION COVERAGE: 100.00% ASSERTIONS: 105
Total coverage By Instance (filtered view): 83.73%
End time: 07:28:15 on Aug 21, 2017, Elapsed time: 0:00:10
Errors: 0, Warnings: 0
    
```



Design Entity	Cover Points	Coverage	Branches	CondBranch	Toggle	FSM Starts	FSM Transitions	Assertions
work.dpu_tb_coverage_cu	1238	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%

Entity	Coverage	Branches	CondBranch	Toggle	FSM Starts	FSM Transitions	Assertions
work.dpu_tb_coverage_cu	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%



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

Source

FSM Coverage

Design

Coverage Summary

Tabs for other Windows

The screenshot displays the Visualizer tool interface with several key components:

- Design Hierarchy (Left):** A tree view showing the project structure, including modules like `hdl_top`, `core_region`, and `riscv_core`. Each module is associated with coverage metrics.
- Coverage Table (Middle-Left):** A detailed table listing coverage metrics for various design units. Columns include Coverage%, Statements%, Branches%, FSM States%, FSM Transitions%, Assertions%, Coverage Bins, and Coverage%. For example, the `hdl_top` unit shows 45.97% coverage, 41.94% statements, and 34.84% branches.
- Source Code (Middle-Right):** A window showing the Verilog source code for `riscv_controller`, with red annotations indicating coverage points.
- FSM Coverage Diagram (Right):** A state transition diagram for the FSM, with states and transitions highlighted in red to show coverage.
- Coverage Summary (Bottom):** A summary section containing:
 - Total Coverage:** A pie chart showing 45.97% covered (green) and 54.03% missed (red).
 - Coverage Types %:** A horizontal bar chart showing the percentage of coverage for different categories:

Category	Count	Percentage
Assertions	1414	100.00%
Branches	2099/6023	34.84%
Coverage groups	83.82%	
Coverage bins	42/52	80.77%
Fsm States	96/345	28.40%
Fsm Transitions	35/669	8.22%
Statements	3389/8079	41.94%
Toggle	2351/95707	24.55%

Visualizer – Linked windows update automatically with interactivity

The image displays a multi-windowed software interface for design coverage analysis. The windows are interconnected, showing how changes in one window affect the others. The windows shown are:

- Design Hierarchy:** A tree view showing the design structure with coverage percentages for various units.
- Code Coverage Outline:** A table showing coverage for specific lines of code.
- Source Code:** A window showing the Verilog source code with coverage bars overlaid on the lines.
- Coverage Details:** A window showing detailed coverage information for a specific state or transition.
- Fsm Coverage:** A state machine diagram showing the coverage for each state and transition.

Red arrows point from the Design Hierarchy and Code Coverage Outline windows to the Source Code window, indicating that the source code is updated based on the selected design unit and code coverage outline. A red box at the bottom of the image contains the text: "All coverage analysis windows are synchronized to your interactions".

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



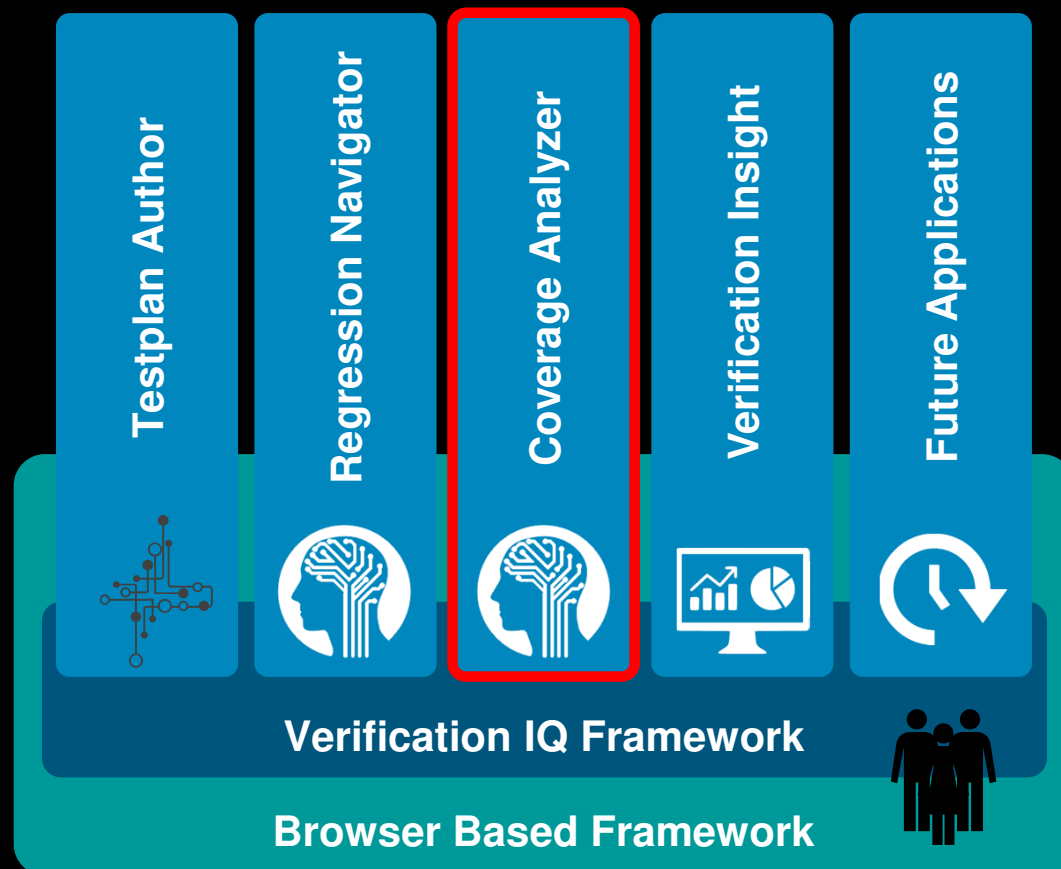
HW features verified from IP to system level to minimizing bug escapes into silicon



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

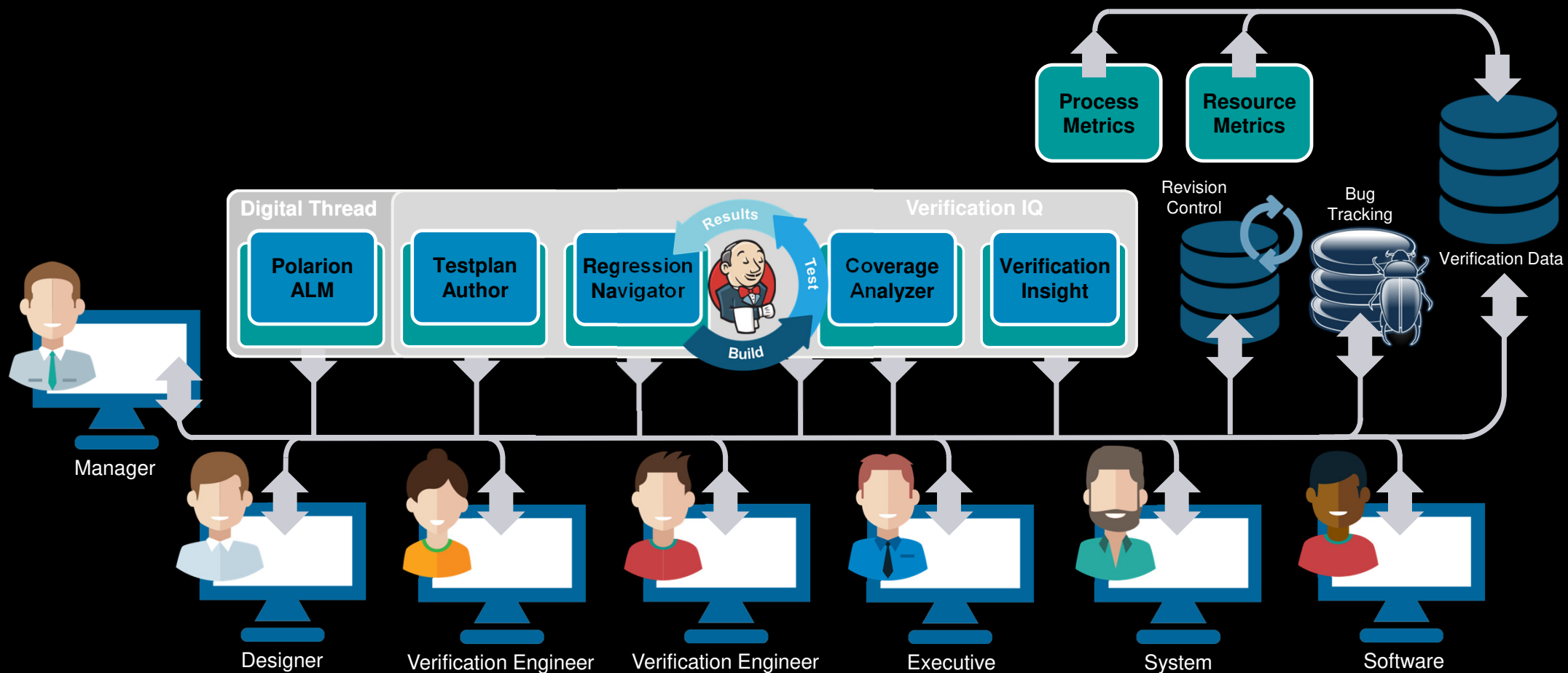
- Code/Functional/Testplan
- Hole, Test & Time Domains
- ML Powered Analytics

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

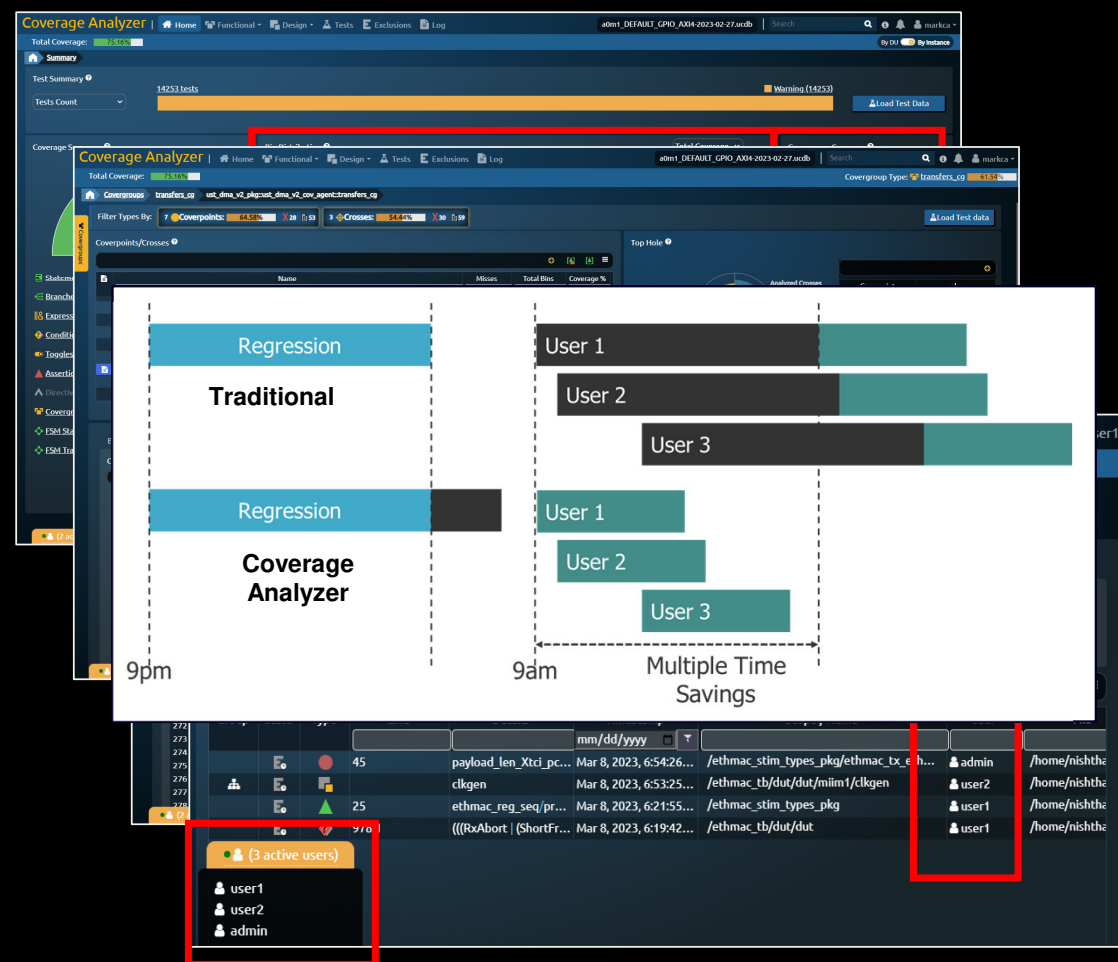
Browser-based Coverage Closure
 Analyze Code, Functional and Testplan Coverage
 Unified support of [Multiple Engines](#)

Power of Collaborative Data Analytics

Improved Coverage Model & Closure Understanding
 Analytical Navigation and Smart Visualization
 Adds User awareness with Questa Exclusion Flows

Application Benefits

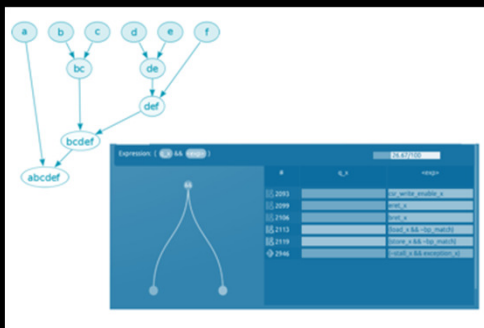
Brings Team Based Analysis to the Questa UCDB
 Unique Analytics for Accelerated Understanding
 ML Accelerated Coverage Closure



Pattern and Hole Analysis

Guiding Closure with Analytical Visualizations Powered by ML

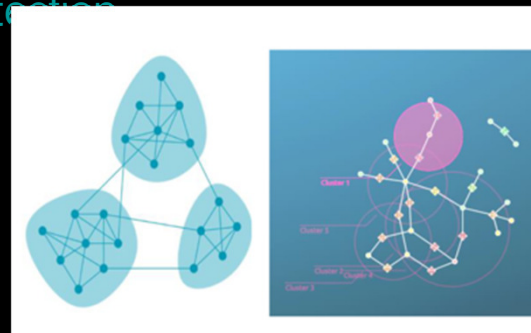
Expression Coverage Agglomerative Clustering



- 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

UCDB Browser

Include Full Path Add

File Name	Owner	Date Added	Users					
merge_all_engines.ucdb	amam	Oct 24, 2023, 9:10:31 am	(0 active users)					
formal_rx_fsm.ucdb	amam	Oct 24, 2023, 9:06:33 am	Not Active					
formal_tx_fsm.ucdb	amam	Oct 24, 2023, 9:06:16 am	Not Active					
emu_traffic_test.ucdb	amam	Oct 24, 2023, 9:05:59 am	Not Active					
sim_register_test.ucdb	amam	Oct 24, 2023, 9:05:34 am	Not Active					

UCDB Diff

Diffs are shown by comparing UCDB #1 to UCDB #2

UCDB #1

UCDB #2

CA Instance Ready
✕

Coverage Analyzer Instance has started
✕

CA Instance Ready
✕

Coverage Analyzer Instance has started
✕

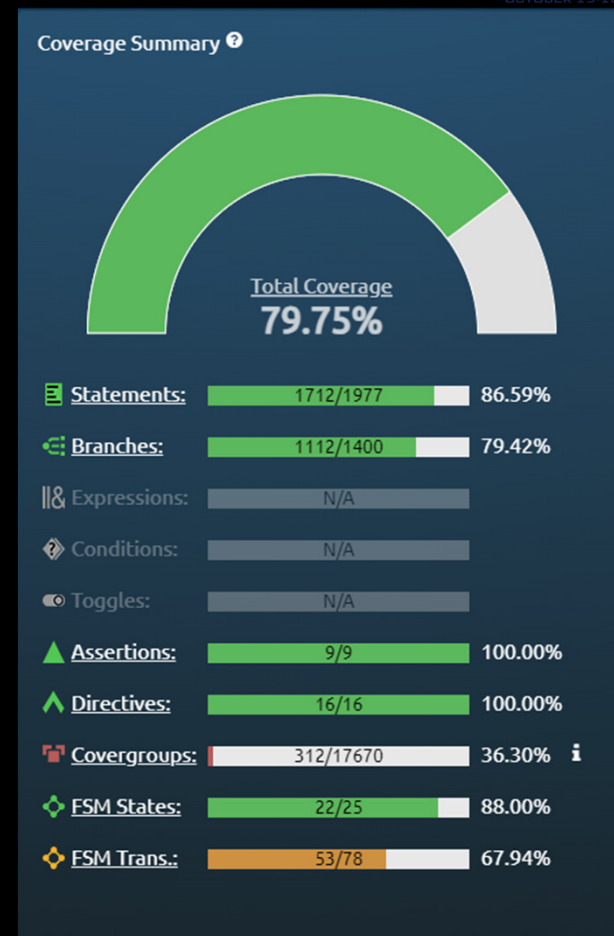
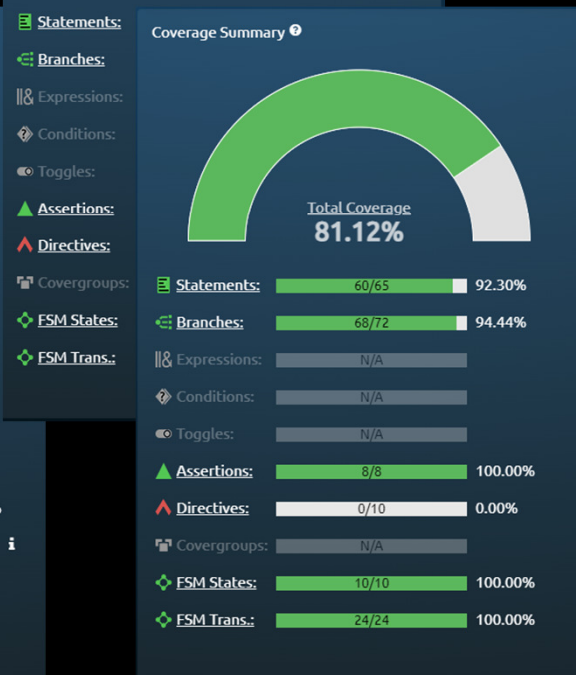
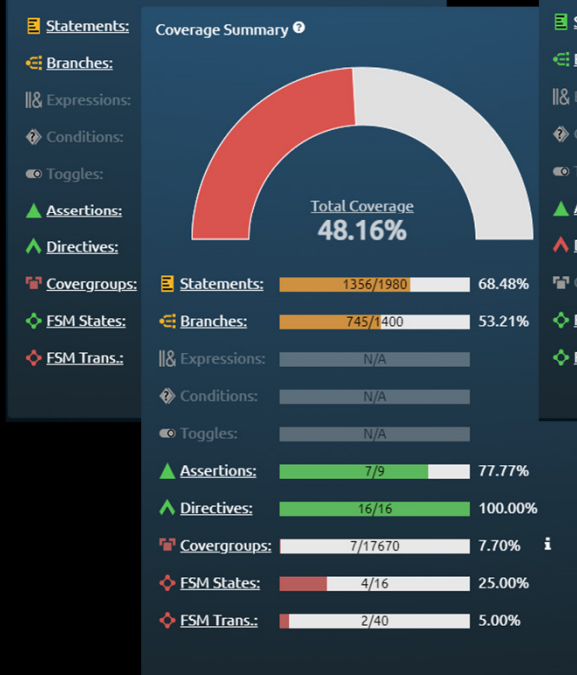
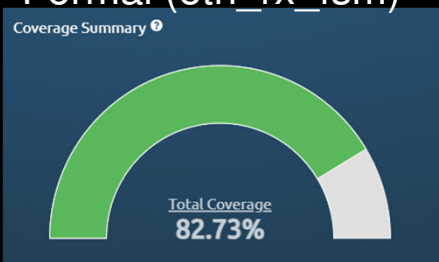
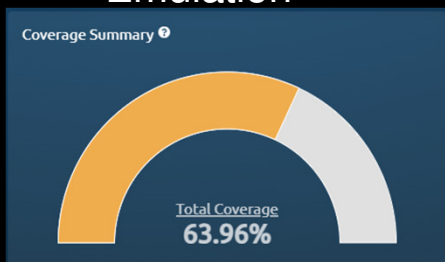
Coverage Analyzer is ready
✕

Coverage summaries loaded for /ww/amam/unifiedCov...ge_all_engines.ucdb
✕

Coverage Results from Unifying Coverage

Emulation

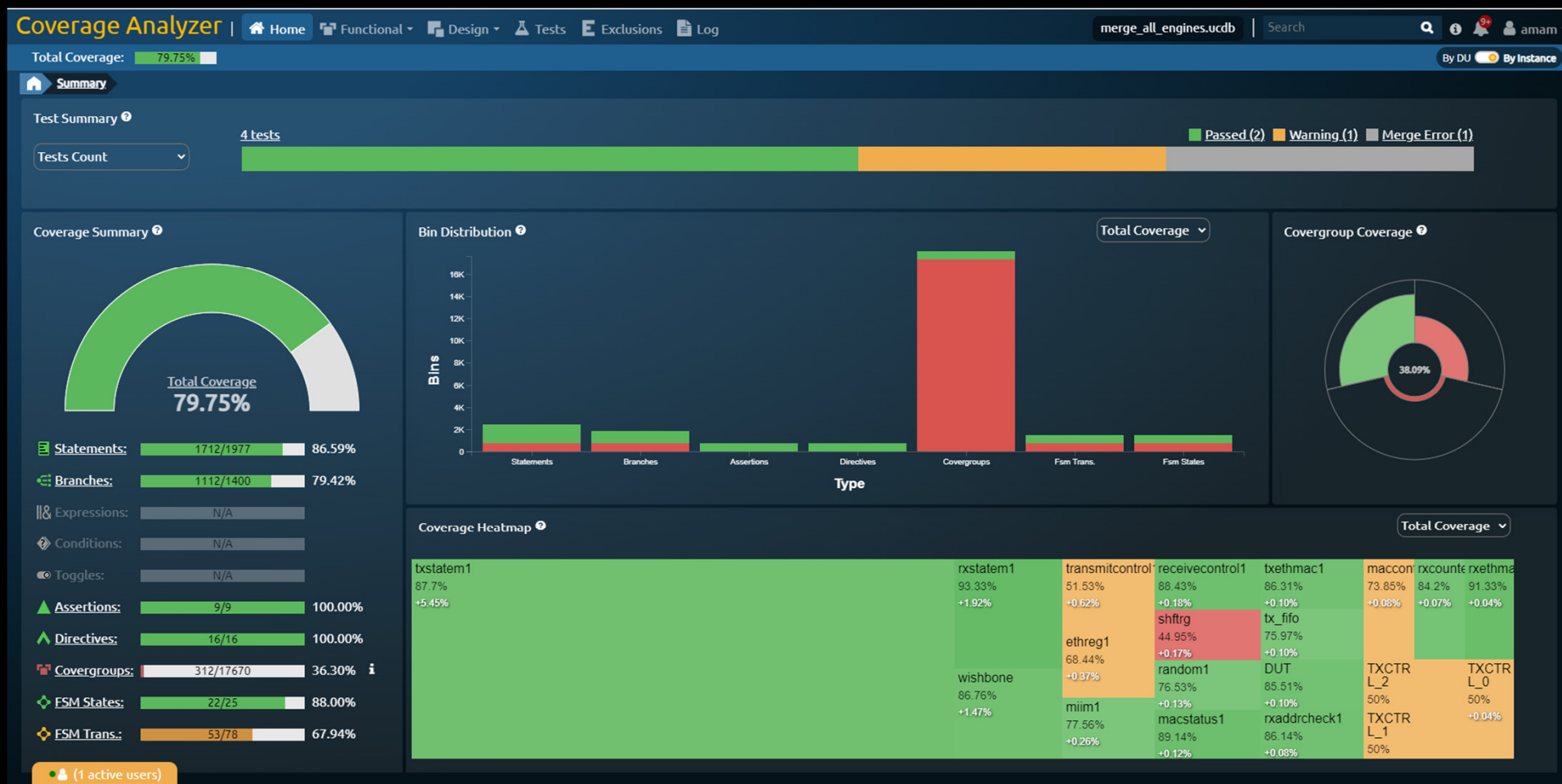
Formal (eth_rx_fsm)



Simulation

Formal (eth_tx_fsm)

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

Design

List Hierarchy Local Recursive

Name	Coverage %	State...	Branch %	Fsm Stat...	Fsm Tran...	Assertio...	Directive...	Covergro...
eth_phy_pkg	0.09%							0.09%
ethmac_env_pkg	24.82%							1.60%
hdl_top								
DUT	90.71%	96.42%	85.00%					
ethreg1	86.18%	92.13%	80.23%					
maccontrol1	85.50%	96.00%	75.00%					
recevecontrol1	88.43%	90.10%	86.76%					
transmitcontrol1	51.53%	56.52%	46.55%					
macstatus1	89.14%	89.83%	88.46%					
mim1	78.08%	85.05%	71.11%					
rxethmac1	94.03%	96.07%	92.00%					
crcrx	100.00%	100.00%	100.00%					
rxaddrcheck1	86.14%	87.09%	85.18%					
rxcounters1	84.20%	90.62%	77.77%					
rxstatem1	92.50%	100.00%	100.00%	100.00%	70.00%			
txethmac1	93.10%	93.24%	92.95%					
random1	76.53%	94.73%	58.38%					
txcounters1	91.15%	95.65%	86.66%					
txcrc	100.00%	100.00%	100.00%					
txstatem1	85.99%	100.00%	96.05%	81.25%	66.66%			

Design hdl_top DUT wishbone

Filter Types by: Statements 589/662 Branches 400/472 Expressions N/A Conditions N/A Toggles N/A FSM States N/A FSM Transitions N/A Assertions N/A Directives N/A

eth_wishbone.v (589/662) (400/472) (0/0) (0/0) (0/0) (0/0) (0/0) (0/0) (0/0)

```

744 ✓ WbEn <= 1'b1;
745 ✓ RxEn <= 1'b0;
746 ✓ TxEn <= 1'b0;
747 ✓ ram_addr <= 8'h0;
748 ✓ ram_di <= 32'h0;
749 ✓ BDRead <= 1'b0;
750 ✓ BDWrite <= 0;
751   end
752 ✓ else
753   begin
754     // Switching between three stages depends on enable signals
755     /* verilator lint_off CASEINCOMPLETE */ // JB
756 X   case ({WbEn_q, RxEn_q, TxEn_q, RxEn_needed, TxEn_needed}) // synopsys paralle
757 ✓     5'b100_10, 5'b100_11 :
758   begin

```

Outline

Line#	Hit	Coverage	Details
688	✓		assign m_wb_bte_o = 2'b00; // Linear burst
690	✓		assign m_wb_stb_o = m_wb_cyc_o;
692	✓		always @ (posedge WB_CLK_I)
694	✓		WB_ACK_O <= (BDWrite) & WbEn & WbEn_q BDRead ...
697	✓		assign WB_DAT_O = ram_do;
719	✓		assign ram_ce = 1'b1;
720	✓		assign ram_we = (BDWrite & {4{(WbEn & WbEn_q)}}
722	✓		assign ram_oe = BDRead & WbEn & WbEn_q TxEn & Tx...
727	✓		always @ (posedge WB_CLK_I or posedge Reset)
729.1	✓		All False

Functional Coverage

Covergroups, Assertions, Directives (Covers)

Covergroups ethmac_coverage_cg

Filter Types By: 4 **Coverpoints:** 26.54% X 3014 3112 3 **Crosses:** 2.06% X 4566 4662

Coverpoints/Crosses: Top Hole

Holes Bins Explore

Assertions & Directives

Filter Types by: **Assertions** 9/9 **Directives** 16/16

Filter Types by: **Assertions** 9/9 **Directives** 16/16

List Hierarchy Local Recursive

Name	Assertions %	Directives %	Language	Type	Scope	Hit #
hdl_top	100.00%	100.00%				
DUT	100.00%	100.00%				
rxethmac1	100.00%	100.00%				
rxstatem1	100.00%	100.00%				
bind_eth_rxstat...	100.00%	100.00%				
cov_Data0			SVA	Concurrent	/hdl_top/DUT/rxethmac1/rx...	19919
cov_Data1			SVA	Concurrent	/hdl_top/DUT/rxethmac1/rx...	19870
			SVA	Concurrent	/hdl_top/DUT/rxethmac1/rx...	3
			SVA	Concurrent	/hdl_top/DUT/rxethmac1/rx...	51
			SVA	Concurrent	/hdl_top/DUT/rxethmac1/rx...	51
			SVA	Concurrent	/hdl_top/DUT/rxethmac1/rx...	2

```

33 // Covers
34 cov_Idle:   cover property (@(posedge MRxClk) $fell(StateIdle) );
35 cov_IPG:   cover property (@(posedge MRxClk) $fell(StateDrop) );
36 cov_Preamble: cover property (@(posedge MRxClk) $fell(StatePreamble) );
37 cov_Data0: cover property (@(posedge MRxClk) $fell(StateData[0]) );
38 cov_Data1: cover property (@(posedge MRxClk) $fell(StateData[1]) );
39 cov_PAD:   cover property (@(posedge MRxClk) $fell(StateSFD) );
40
41 endmodule
42
  
```

Directives Summary

Hit Ratio: 16/16

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

The screenshot displays the Siemens Coverage Closure GUI. On the left, a table lists excluded code lines. On the right, the Verilog source code for 'eth_registers.v' is shown with a context menu open over line 956, offering options to 'Exclude', 'Exclude With Comment', or 'Include'.

State	Type	Line	Details	Scope / Name	File
E			Linerange 163 168	/hdl_top/DUT/rxethmac1/rxstatem1	/medsj/v3alpha3/pmotafr/MED/cov...
E		267		/hdl_top/DUT/txethmac1/bxstatem1	/medsj/v3alpha3/pmotafr/MED/cov...
E			work.uvmf_base_pkg		
E		956		/hdl_top/DUT/ethreg1	/medsj/v3alpha3/pmotafr/MED/cov... This line is not necessary for coverage. amam
E			Linerange 162 168	/hdl_top/DUT/rxethmac1/rxstatem1	/medsj/v3alpha3/pmotafr/MED/cov...
E		267		/hdl_top/DUT/txethmac1/bxstatem1	/medsj/v3alpha3/pmotafr/MED/cov...

```

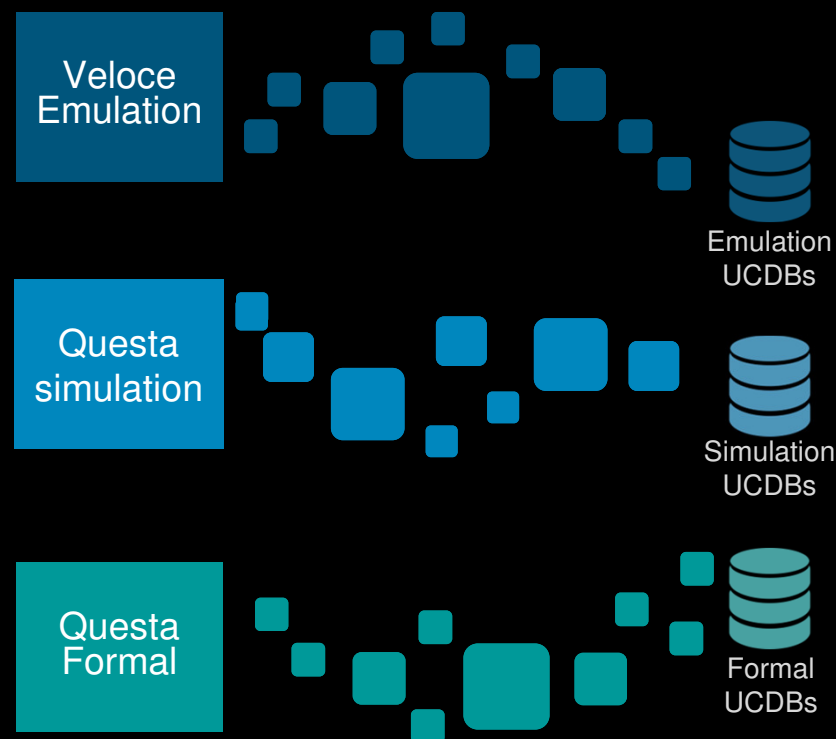
eth_registers.v
946 ✓ assign r_txPauseIV[15:0] = TXCTRLOut[15:0];
947 ✓ assign r_txPauseRq      = TXCTRLOut[16];
948
949
950 // Synchronizing TxC Interrupt
951 ✓ always @ (posedge TxClk or posedge Reset)
952   begin
953   ✓   if(Reset)
954   ✓     SetTxCIrq_txclk <= 1'b0;
955   ✓   else
956   ✗     if(TxCtrlEndFrm & StartTxDone & r_TxFlow)
957   ✗       SetTxCIrq_txclk <= 1'b1;
958
959   ✗
960   ✗
961   end
  
```


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



Questions



Thank You