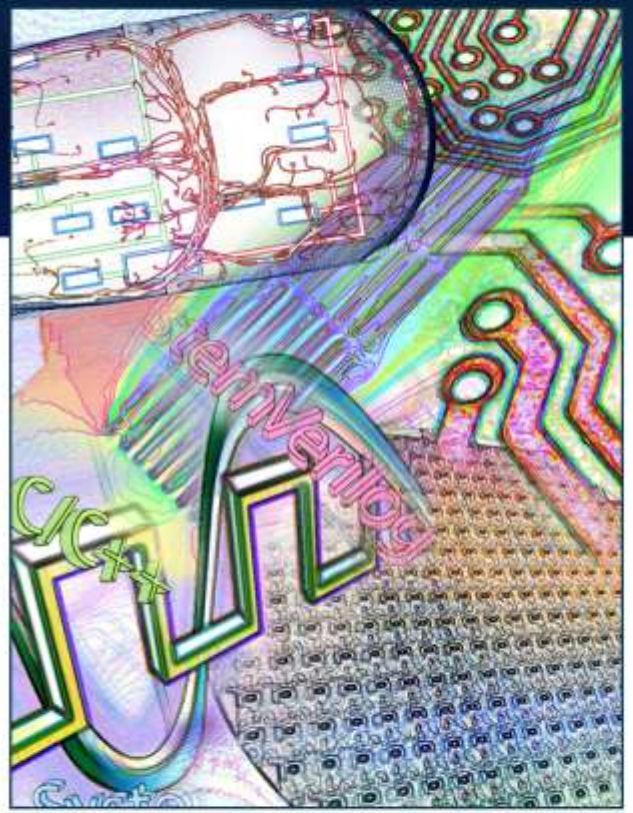
Evolution of Digital Verification



Walter Gude

Applications Engineering Consultant

Walter_gude@mentor.com

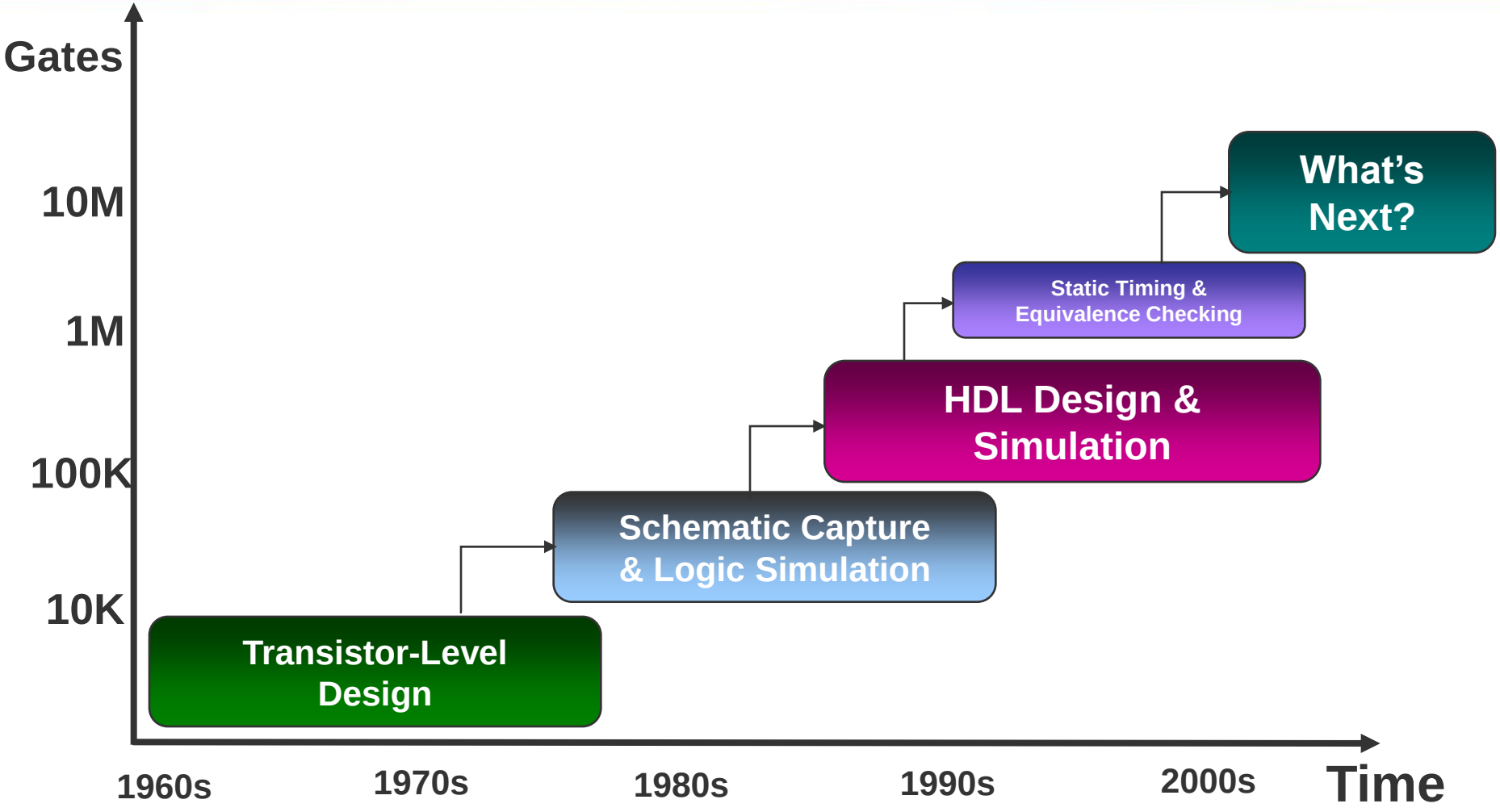
Presented to the IEEE Long Island Section
Signal Processing Society & Photonics Society
on Tuesday October 8th, 2013

Mentor Graphics

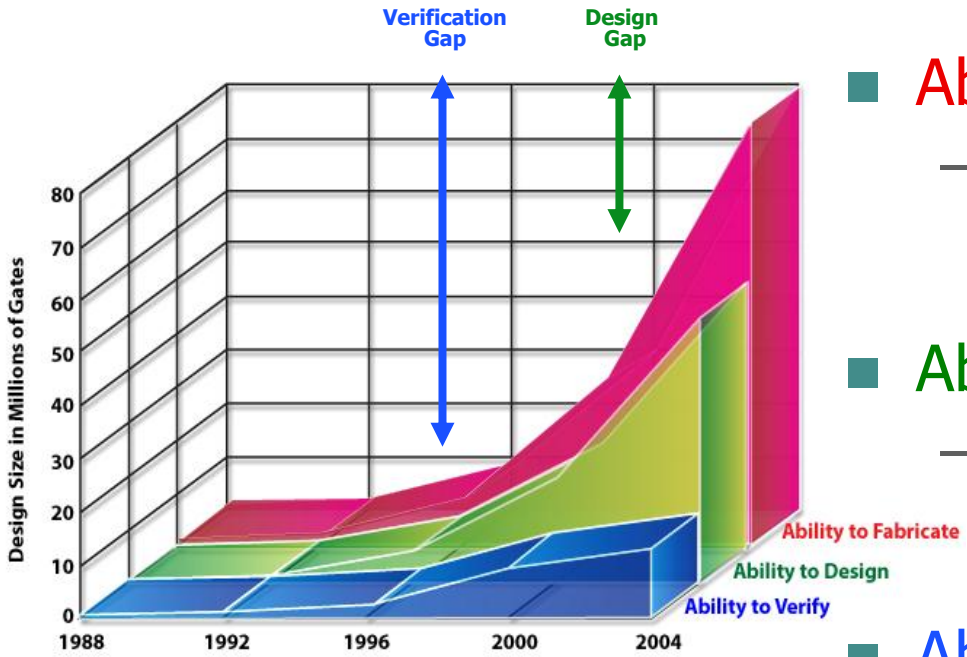
Agenda

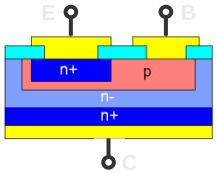
- Verification Overview
  - Assertion and Functional Coverage
  - Constrained Random
  - Requirements Tracing
  - Algorithmic TB InFact
  - Questa Formal
  - Questa Codelink
  - Questa VIP

The Evolution of Digital Design & Verification

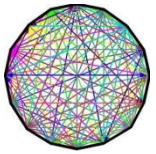


Creating HUGE Verification Challenges

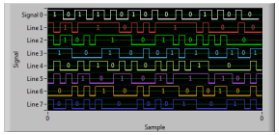




- Ability to Fabricate
  - Dominated by feature size

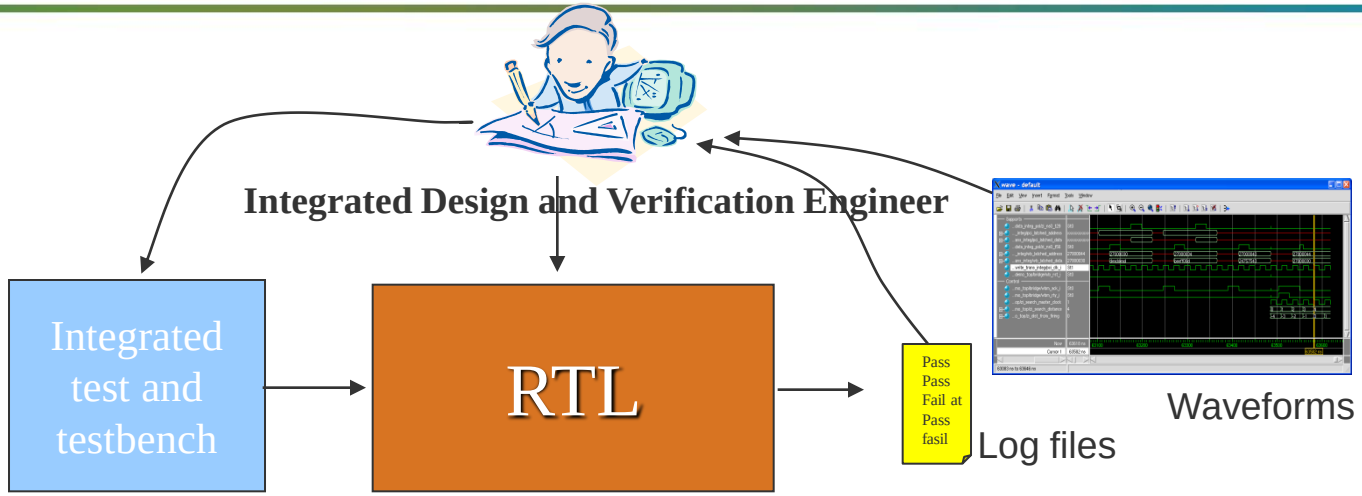


- Ability to Design
  - Dominated by interconnect



- Ability to Verify
  - Dominated by behavior over time and complexity of concurrency

Traditional Directed Test Flow….

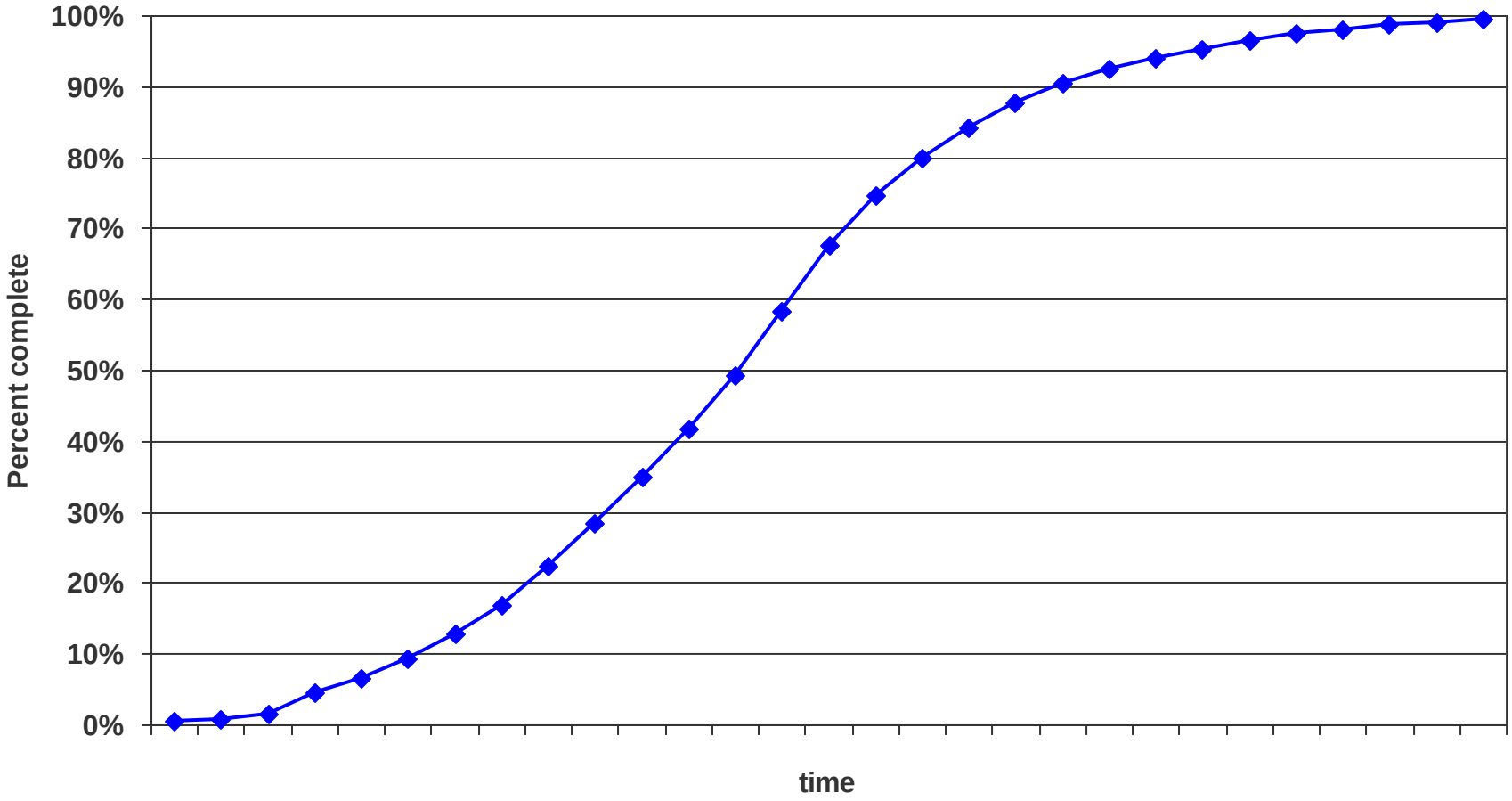


- **Mainly directed tests**
  - **HDL and some C/C++**
- **Ad hoc test planning**
- **Code Coverage tools for advanced users**
- **Manually inspected output files and waveforms**

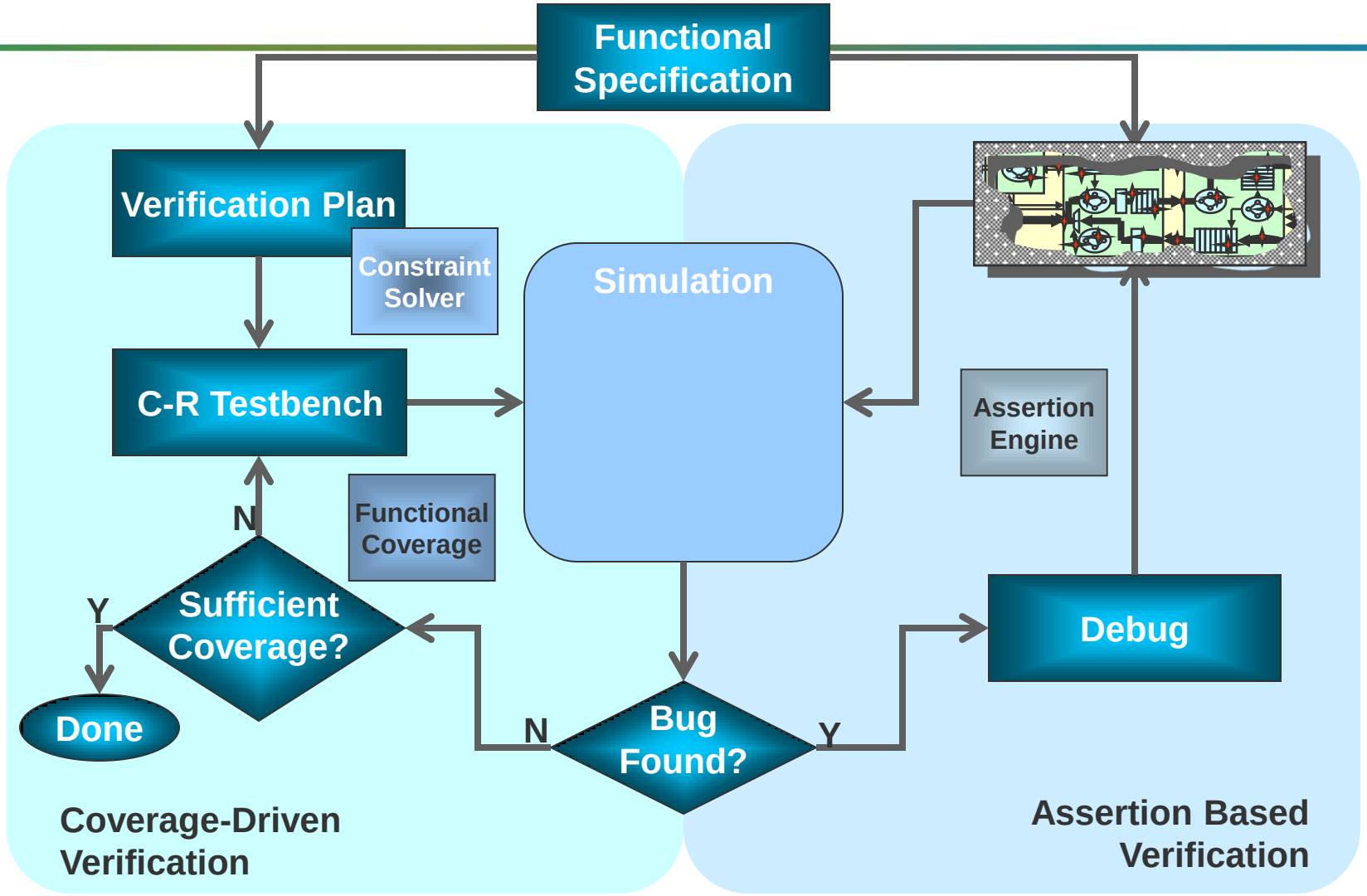
- **Problem**: Creation of directed tests does NOT scale with increased complexity
  - Number of tests directly linked to number of engineers writing tests
  - Amount of testbench code becomes difficult to manage
- NEED better way to create stimulus!

- Problem: Code Coverage does not measure how well specifications are covered.

When Is Verification Complete?

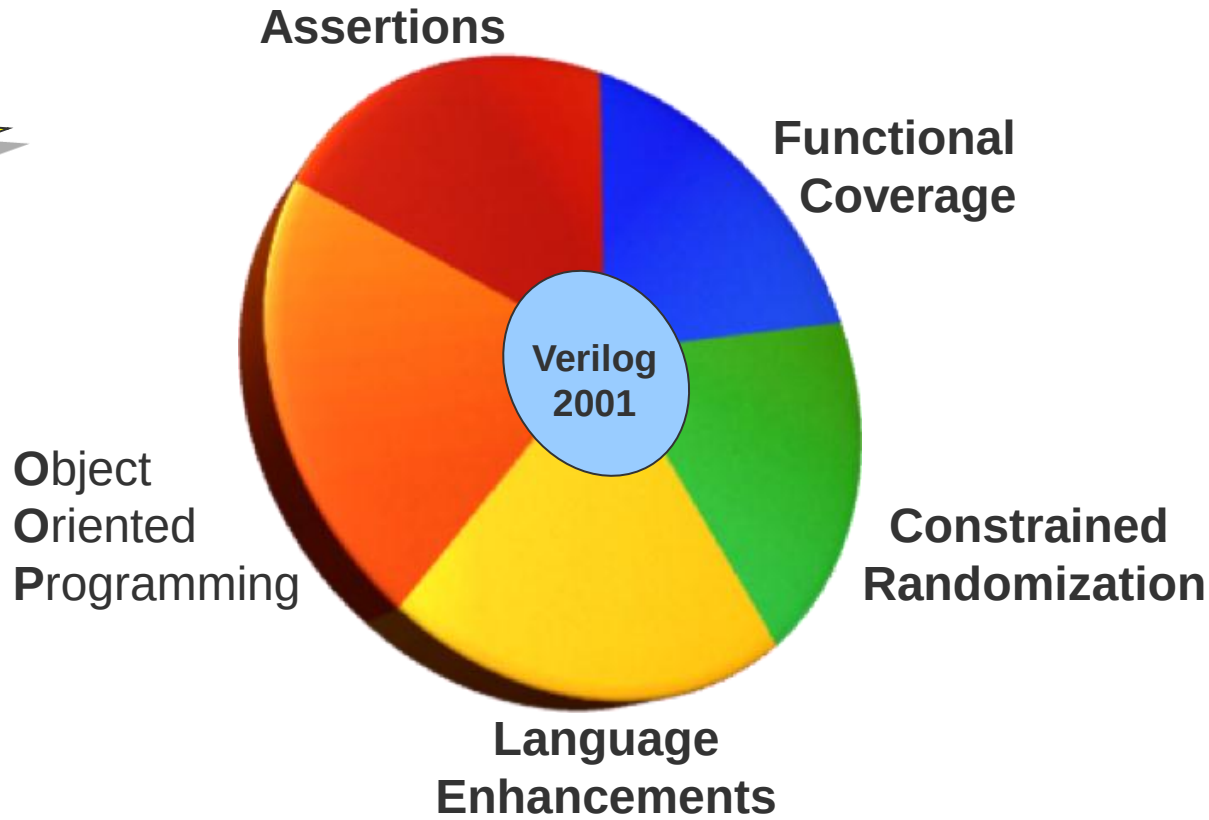


Advanced Verification

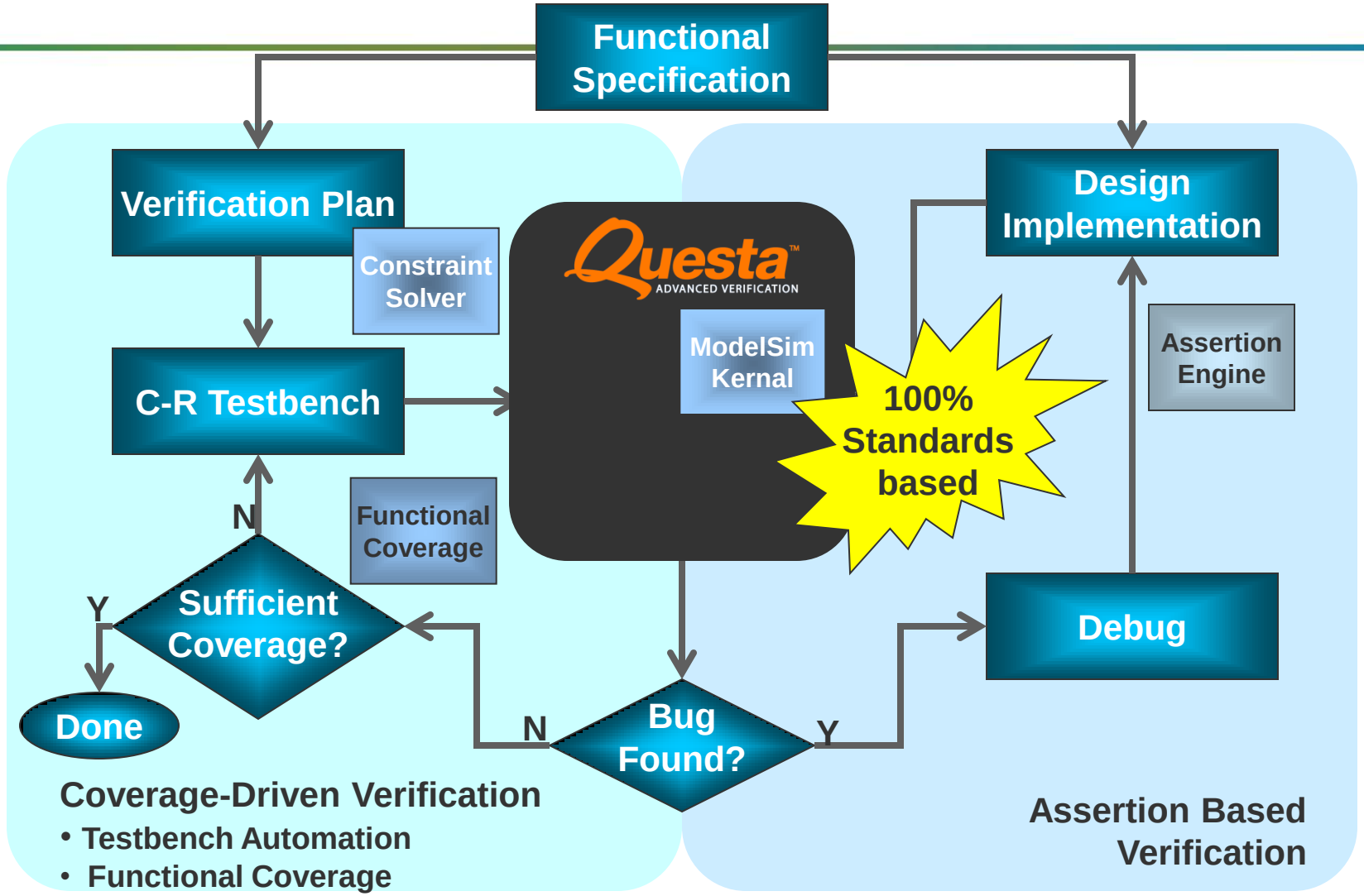


SystemVerilog

- Verilog 2001 base…
- Massive enhancements

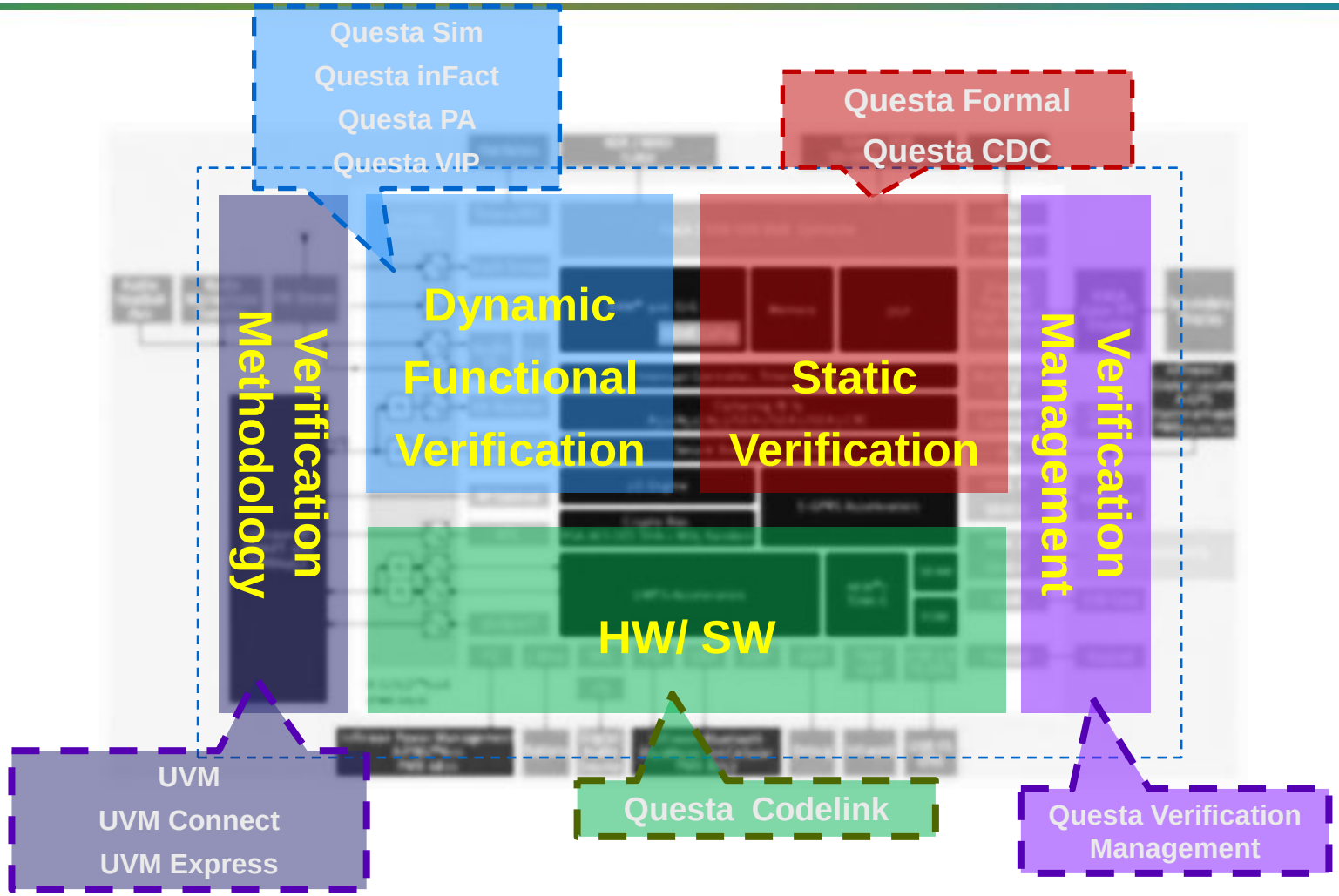


The Verification Process



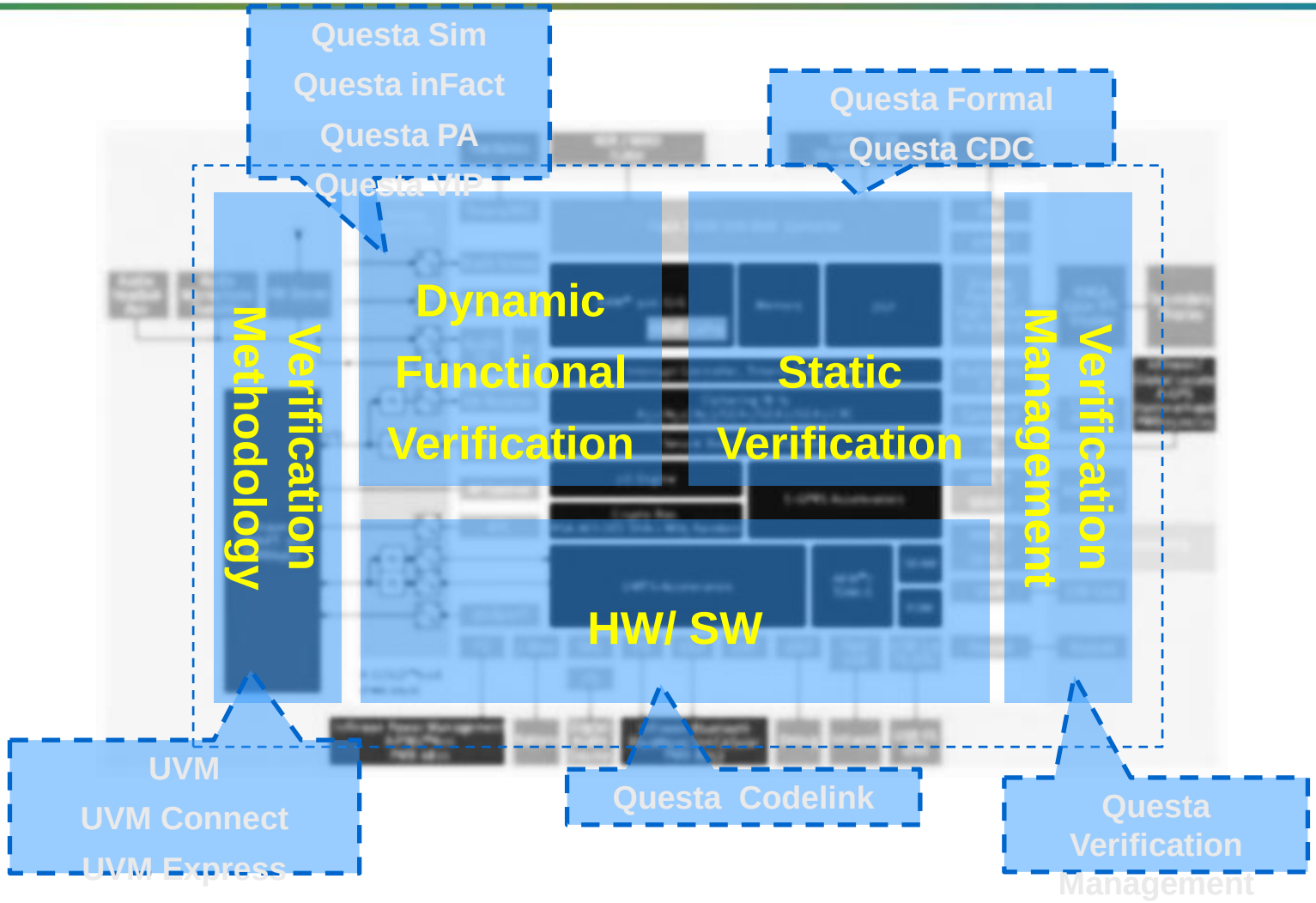
Questa Verification Platform

Best In Class Engines



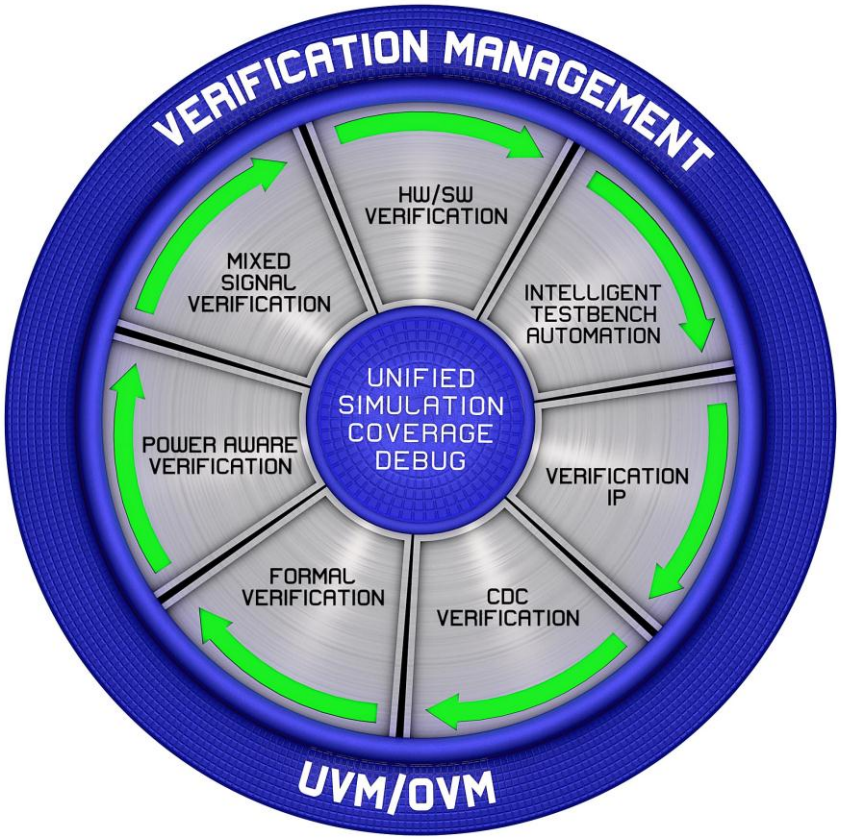
Questa Verification Platform

Best In Class Engines - Unified Front End Analysis & Compile



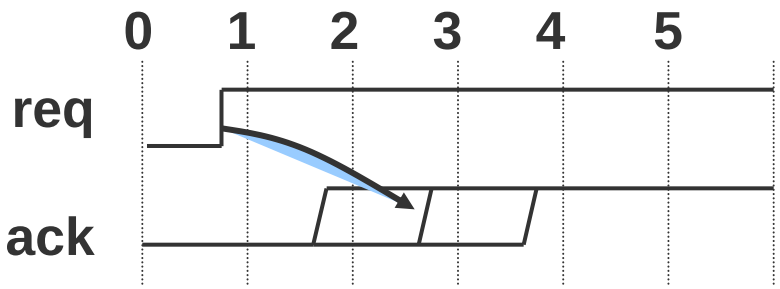
Questa Verification Platform

- Comprehensive integrated SOC verification platform
  - Best in class engines
  - Integrated
  - Comprehensive debug analysis
- Industry Leading SOC Verification Solutions
  - Coverage Closure Solution
  - Low Power Verification Solution
  - Software Driven Verification Solution
- Standards Leadership
  - Driving the evolution of IEEE standards
  - Major donations to Accellera UVM
  - Accellera UCIS from Mentor UCDB



What is an Assertion?

A concise description of [un]desired behavior



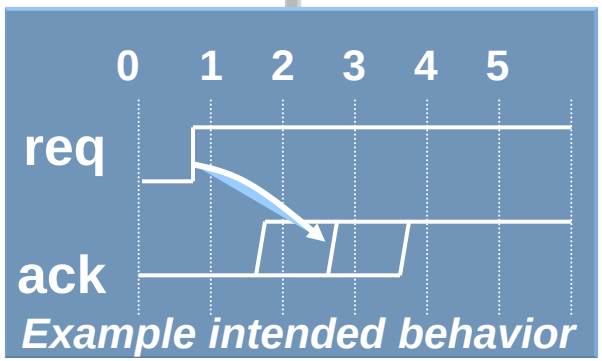
Example intended behavior

"After the request signal is asserted, the acknowledge signal must come 1 to 3 cycles later"

SV Assertions

SV Assertion

```
property req_ack(req,ack);
 @(posedge clk) $rose(req) |-> ##[1:3] $rose(ack);
endproperty
as_req_ack: assert property (req_ack(req1,ack1));
```



VHDL

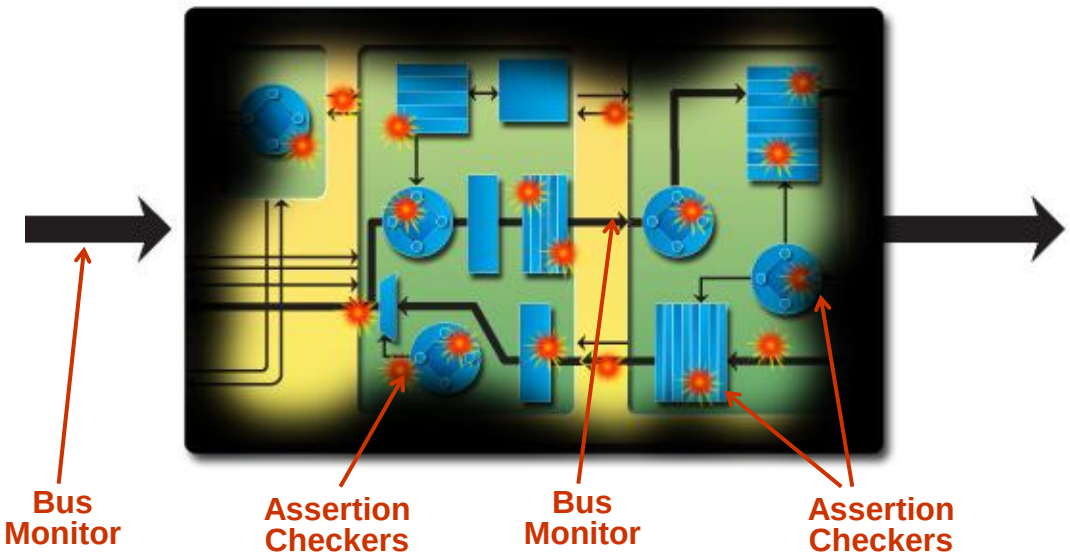
```
sample_inputs : process (clk)
begin
  if rising_edge(clk) then
    STROBE_REQ <= REQ;
    STROBE_ACK <= ACK;
  end if;
end process;
protocol: process
  variable CYCLE_CNT : Natural;
begin
  loop
    wait until rising_edge(CLK);
    exit when (STROBE_REQ = '0') and (REQ = '1');
  end loop;
  CYCLE_CNT := 0;
  loop
    wait until rising_edge(CLK);
    CYCLE_CNT := CYCLE_CNT + 1;
    exit when ((STROBE_ACK = '0') and (ACK = '1')) or (CYCLE_CNT = 3);
  end loop;
  if ((STROBE_ACK = '0') and (ACK = '1')) then
    report "Assertion success" severity Note;
  else
    report "Assertion failure" severity Error;
  end if;
end process protocol;
```

HDL Assertion

Assertions Need to be Everywhere

Assertions Enable Higher Quality Designs

- Assertions provide observability for higher complexity designs
- Assertions describe (un)desired behavior
- Assertions dramatically shorten debug and repair time
- Assertions stay on during block, chip and system-level tests
  - Finds bugs you weren't looking for



Payoff Is High: Assertions Find Bugs

Assertion Monitors	34%
Cache Coherency Checkers	9%
Register File Trace Compares	8%
Memory State Compare	7%
End-of-Run State Compare	6%
PC Trace Compare	4%
Self-Checking Test	11%
Simulation Output Inspection	7%
Simulation Hang	6%
Other	8%

Kantrowitz and Noack *[DAC 1996]*

Assertion Monitors	25%
Register Miscompares	22%
Simulation "No Progress"	15%
PC Miscompare	14%
Memory State Miscompare	8%
Manual Inspection	6%
Self Checking Test	5%
Cache Coherency Check	3%
SAVES Check	2%

Taylor et al. *[DAC 1998]*

34% of all bugs found were identified by assertions on DEC Alpha 21164 project

[Kantrowitz and Noack DAC 1996]

17% of all bugs found were identified by assertions on Cyrix M3(p1) Project

[Kronik '98]

25% of all bugs found were identified by assertions on DEC Alpha 21264 project

[Taylor et al. DAC 1998]

50% of all bugs were identified by assertions on Cyrix M3(p2) Project

[Kronik '98]

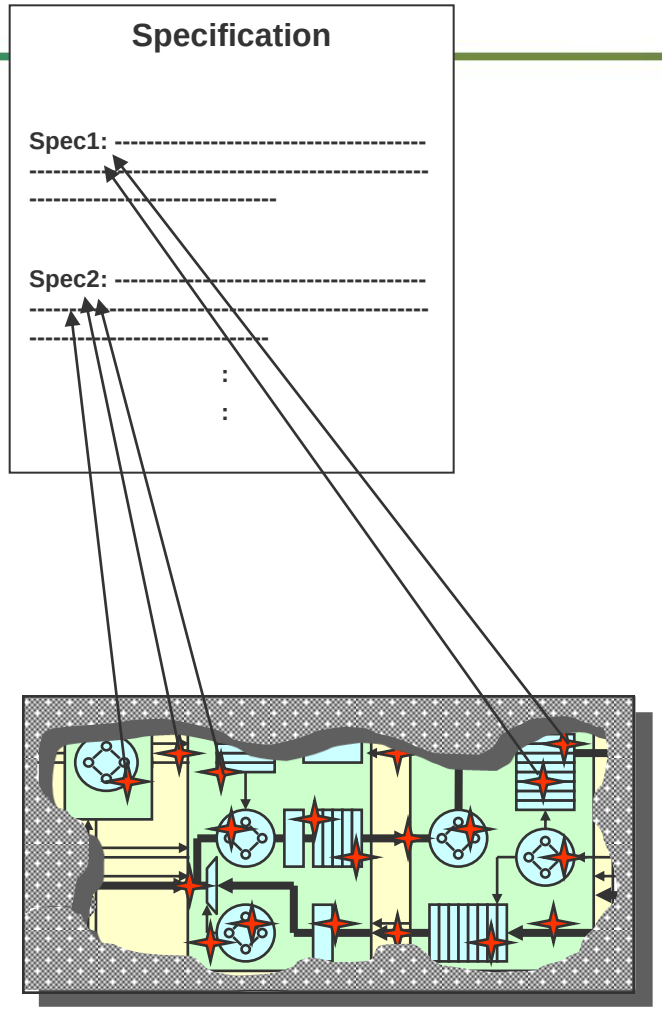
85% of all bugs were found using over 4000 assertions on HP Project

[Foster and Coelno HDLCon 2000]

10,000 assertions in Cisco project

[Sean Smith 2002]

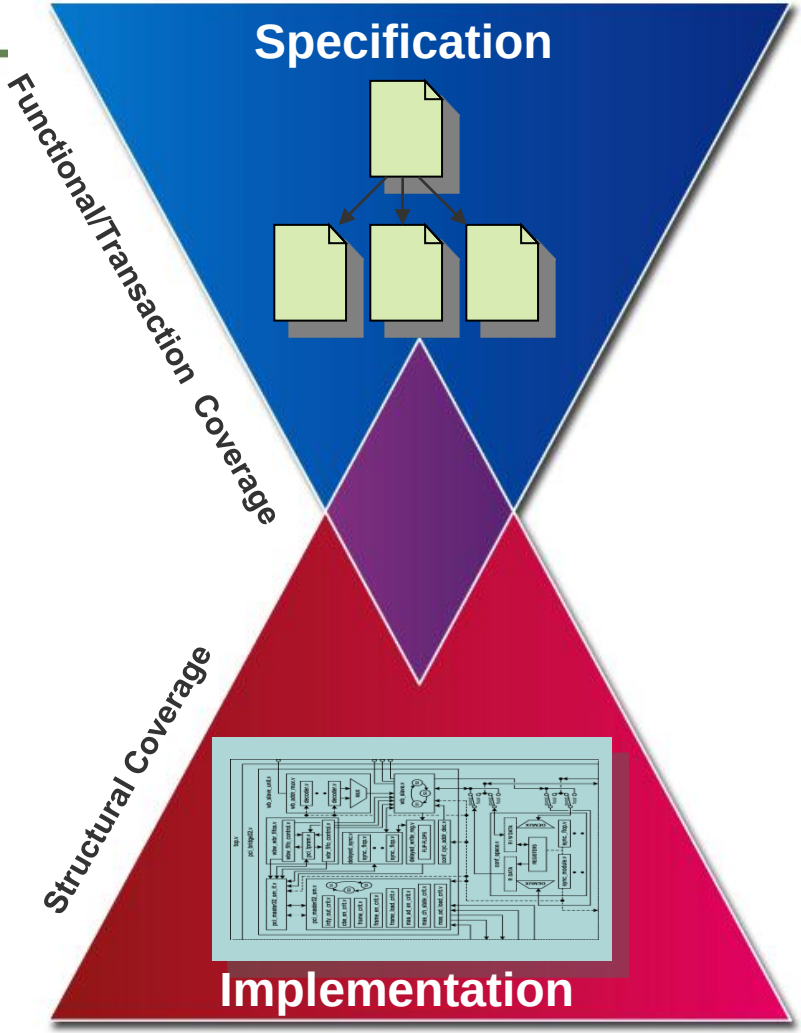
Formal Specification



- Weaknesses of natural language specifications
  - Ambiguity
    - Including the ease of misinterpreting Completeness
  - Cannot be executed to verify
    - Completeness
    - Accuracy
- SystemVerilog/PSL are a specification languages
  - Unambiguous
  - Precise semantics
    - No emotional (connotation) baggage
  - Can be used to create executable specification
    - Coverage of properties (functionality, tests)
    - Accuracy of specification

Total Coverage Model

- Functional (specification-based)
  - Checks that all functions of the design are tested
  - Created by verification team
- Structural (implementation-based)
  - Checks that all corner-cases of the design are tested
  - Created by designers



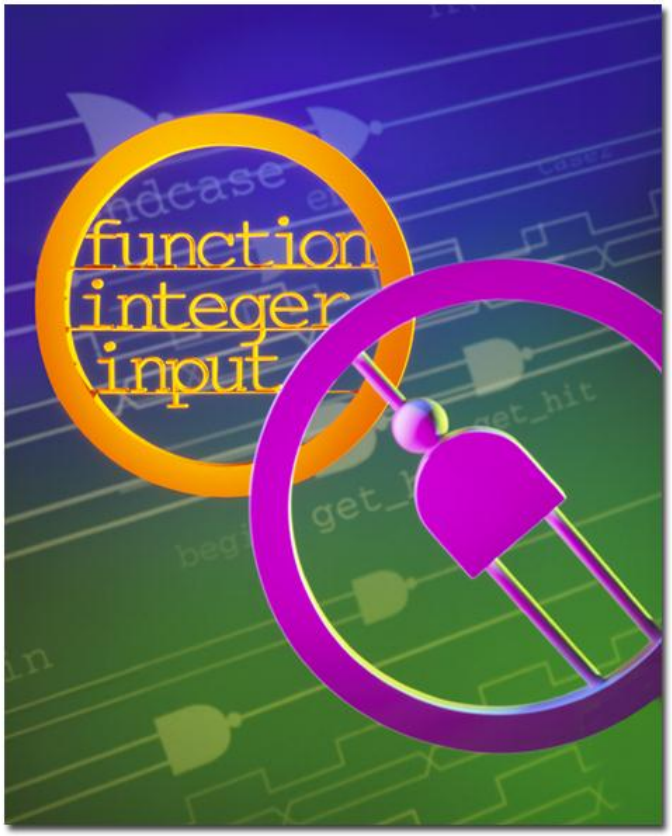
SV Coverage Models

```
// States for a DRAM controller
enum {IDLE, MEM_ACC, SWITCH, RAS_CAS, OP_ACK, REF1, REF2} fsm_state;
```

```
covergroup dram_ctrl_fsm_states @(posedge clk);
  // An impli
  c1: cover
endgroup
```

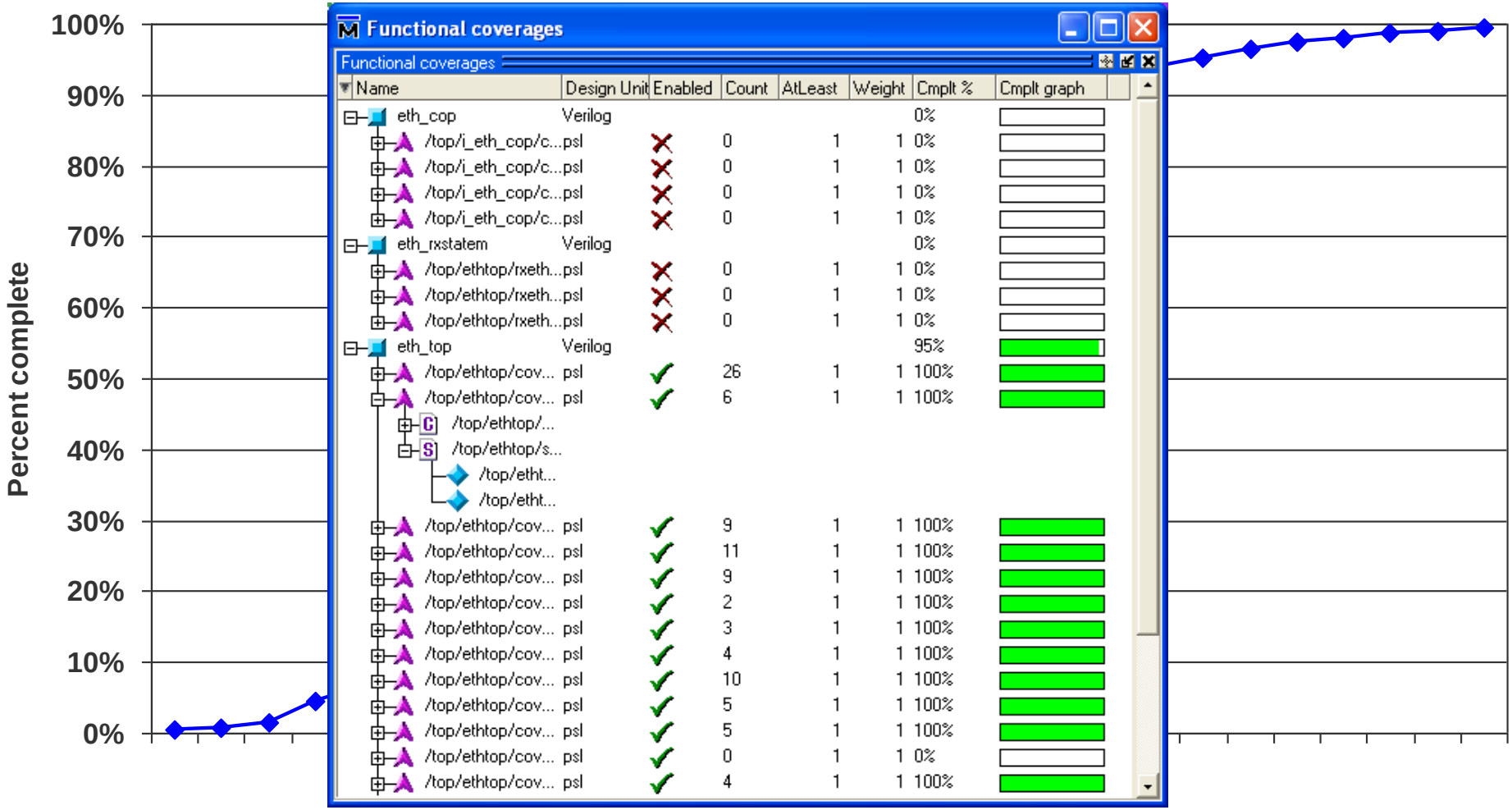
```
covergroup dram_ctrl_fsm_transitions @(posedge clk);
  c1: coverpoint fsm_state {
    bins idle_bin = (IDLE => {REF1, MEM_ACC}) iff (!rst);
    bins ref1_bin = (REF1 => REF2 => IDLE) iff (!rst);
    // RAS_CAS can last one or two cycles depending on if
    // memory access is a read or a write.
    bins mem_acc_bin = (MEM_ACC => SWITCH =>
                        RAS_CAS[*1:2] => OP_ACK => IDLE)
                        iff (!rst);
    bins rst2idle_bin = ({IDLE:REF2} => IDLE) iff (rst);
    bins erroneous = default;  // Catch undefined transitions
  }
endgroup
```

When is verification complete?

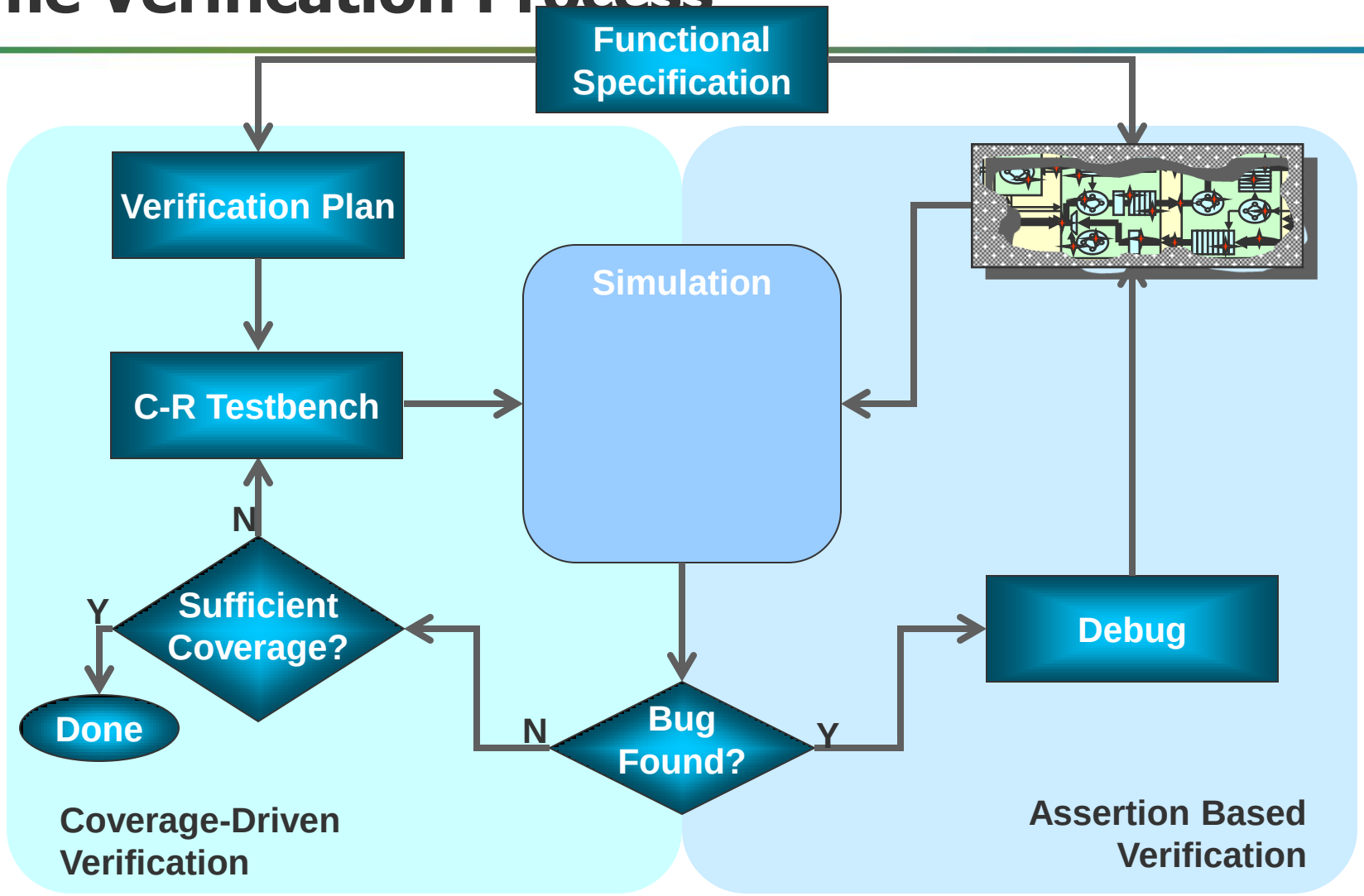


- Verification is effectively metric-less
  - Few designers know if their strategy is adequate or efficient
  - Sign-off criteria are ad hoc and vary by company
  - Code coverage is not a functional verification metric
- If it isn't verified, it's broken

When Is Verification Complete?



The Verification Process



Directed Tests

Specification

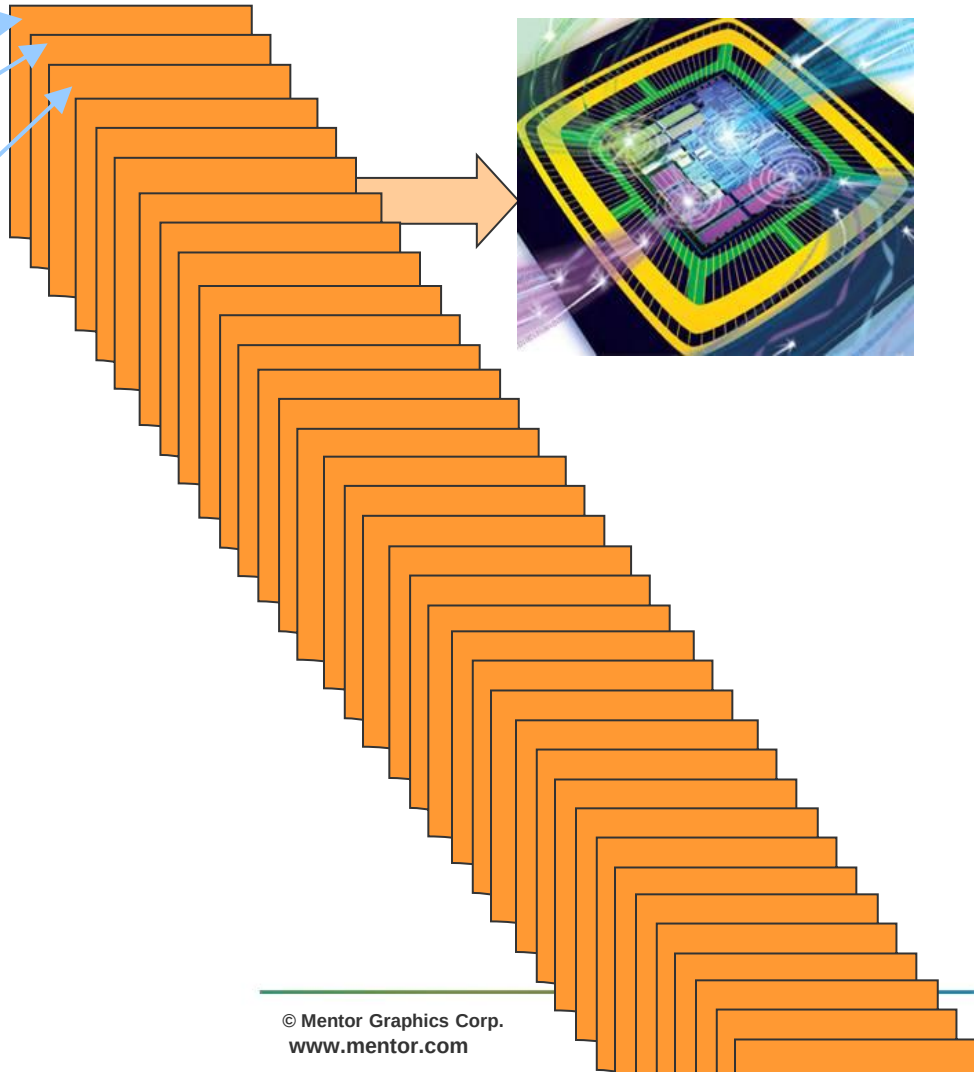
Spec1: ----------

Spec2: ----------

Spec3: ----------

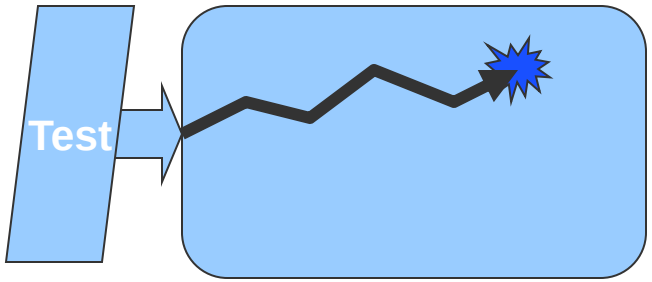
:
:

Spec10,000: ----------

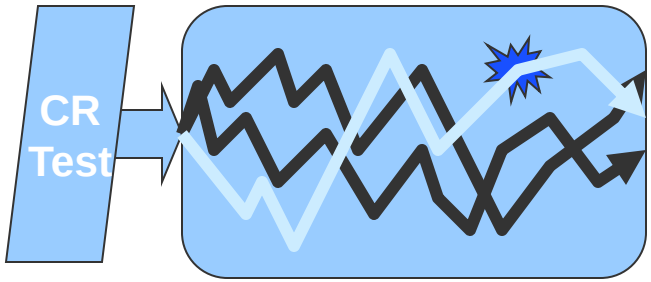


Functional Coverage and Contrained Random Testing

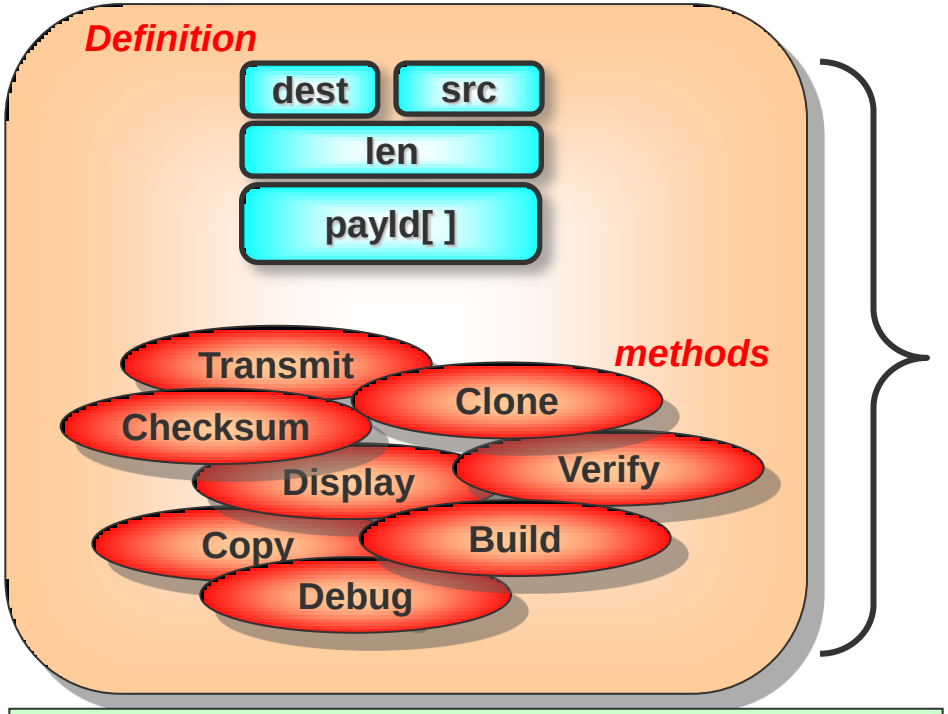
- In a directed test, the coverage points are coded in the test itself
  - Test writer must code *each* specific scenario to specify intent explicitly
  - Must be able to predict interesting cases in order to code them
  - Doesn't scale well



- Randomness inherent in C-R allows unpredicted scenarios to be exercised
  - Exposes corner cases the designer may not have considered
  - Testbench is an objective description of intent against which to check the design
  - Kind of like an independent verification team
- Functional Coverage tells which features were exercised



SV-OOP vs. Verilog




```
class ether_packet extends packet;

// Ethernet Packet Fields
  bit[47:0] dest, src;
  bit[15:0] len;
  bit [7:0] payld [ ];

// onboard methods
  function new(int i);
    payld = new[i];  len = i;
  endfunction : new

  function void display;
    $displayh("\t      src: ", src);
    . . .
    foreach (payld[i])
     $display("payld[i] = %d",i,payld[i]);
  endfunction : display

  task transmit_frame();
    . . .

  function clone();
    . . .

endclass : ether_packet
```

In Verilog, a module is the only container that can hold both the definition of the packet fields and all the methods needed to manipulate those fields.

In SV, classes are infinitely better suited, because:

- Objects are dynamic, not static
- Create or destroy objects at will
- Classes are inheritable, polymorphic etc.

Tools and Technology Aren't Enough

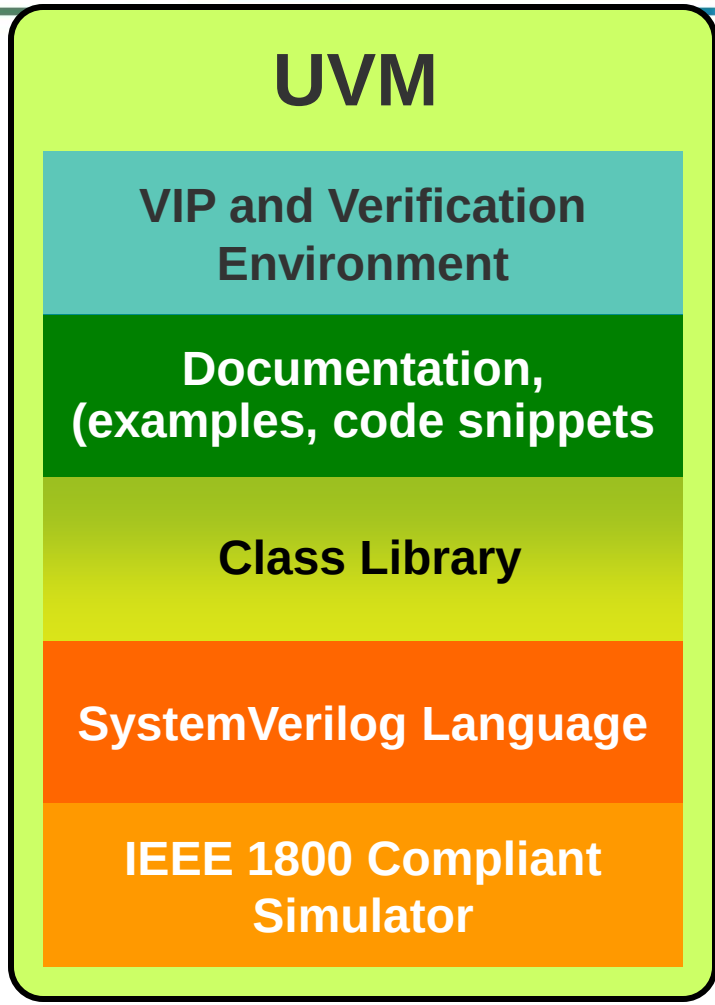
- EDA sells tools and "materials"

- To build a bookcase, you need to apply the tools and technologies in useful ways
- What we need are Advanced Methodologies

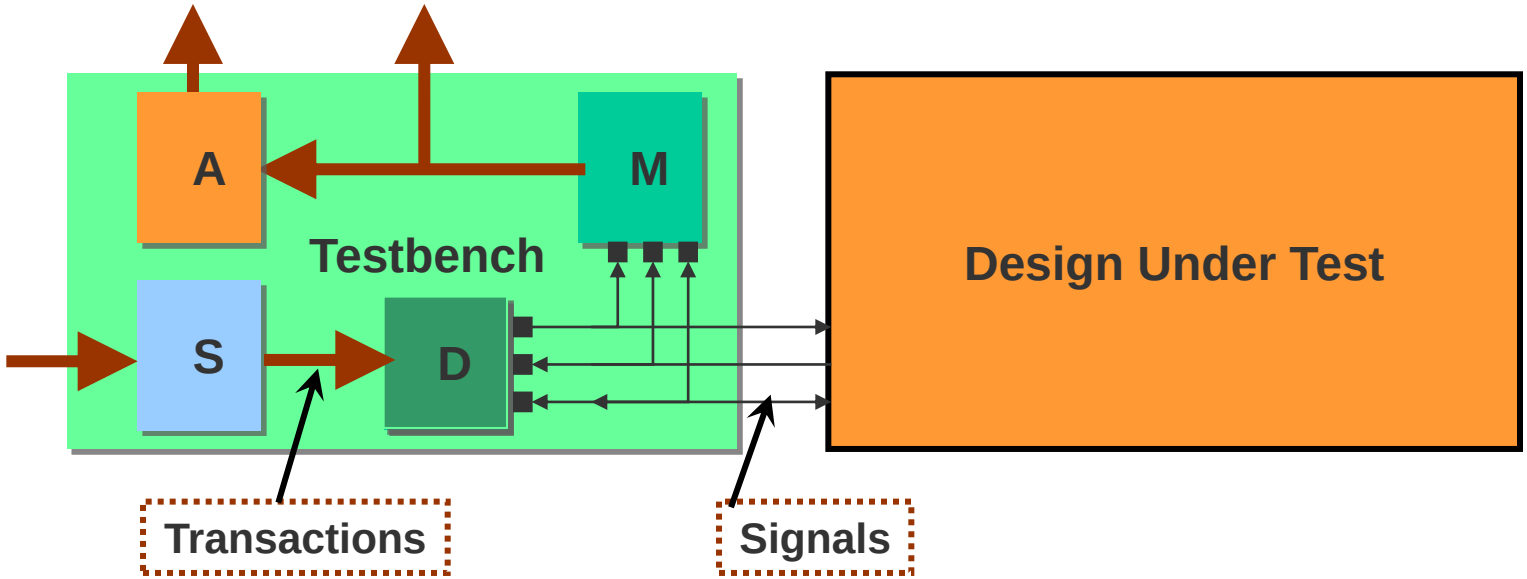
Unified Verification Methodology

- UVM - Open-source framework for reusable verification based on SystemVerilog
- SystemVerilog Class Library
  - Set of core building blocks for effective, reusable verification environments
  - Library of common services

Interoperability and reuse



UVM-based Testbench Architecture



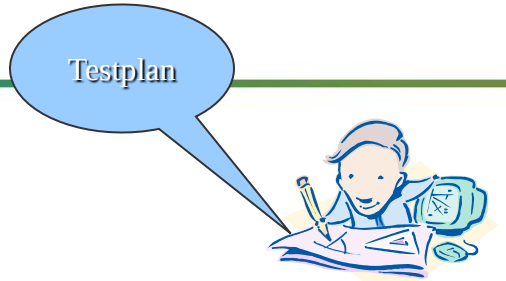
D — Drivers - translate transaction into signals

M — Monitors - translate signals into transactions

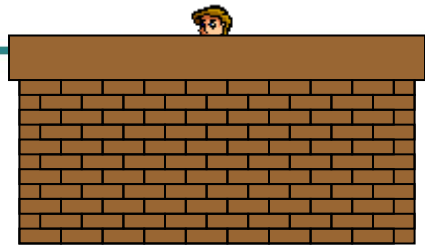
S — Sequencers - generates sequences of transactions

A — Analyzers - analyzes sequences of transactions

Coverage Data Overload



Verification Engineer



Project Manager

VIP

VIP reduces testbench development time
Increases verification efficiency

T1
T2
...
T(n)

Testbench

System

VIP Coverage

Functional

Assertion

Coverage Files

Code/FSM

Run Data

Formal Verification

Functional coverage enables intelligent verification
Increases quality

TLM compo ts enables faster verifica n performance
Enables re-use

Constrained Random doubles productivity of verification engineers
Greatly reduces the need for directed tests

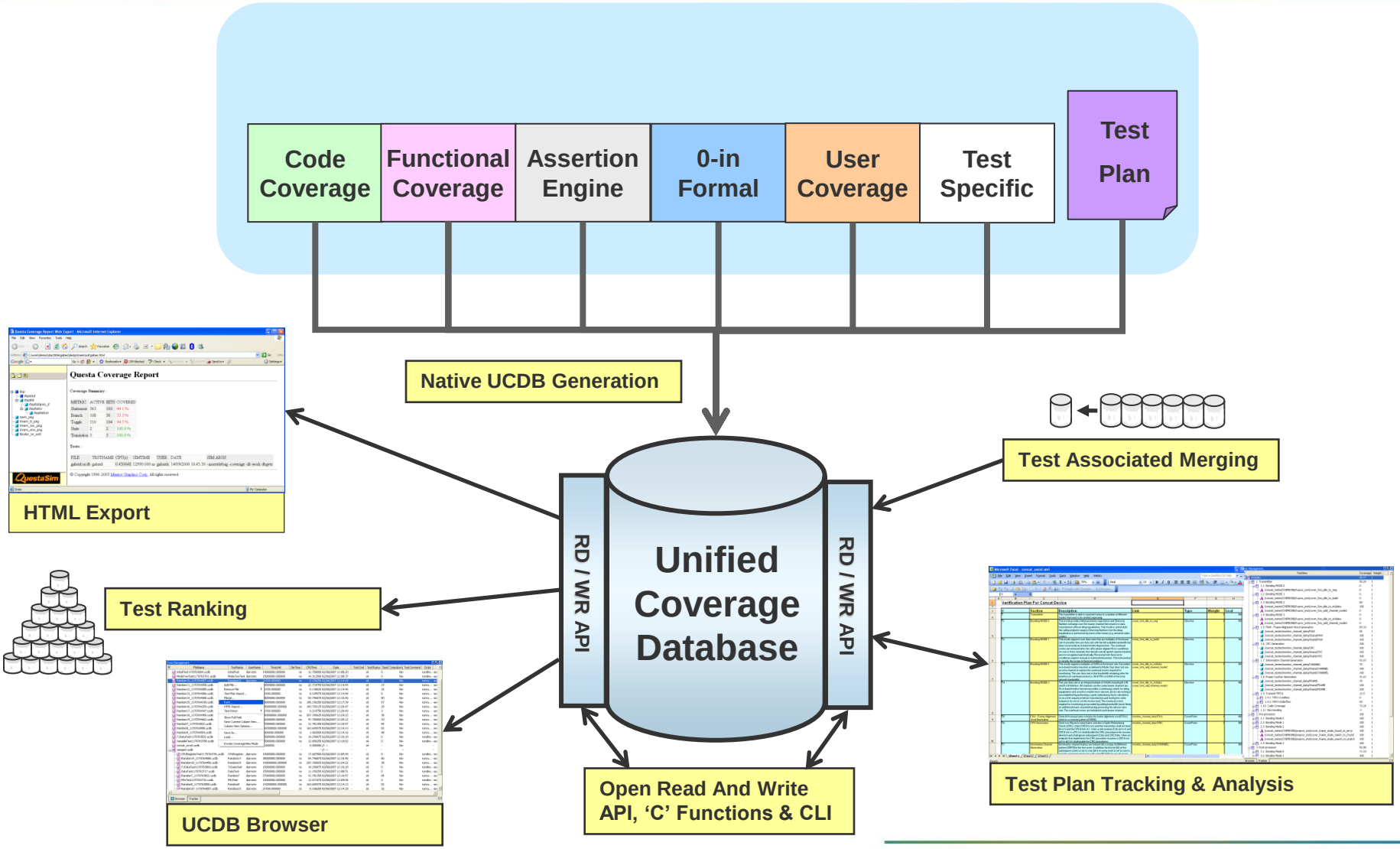
Assertion Library

Assertions improve debug time by 50%
Find hard bugs
Increases quality

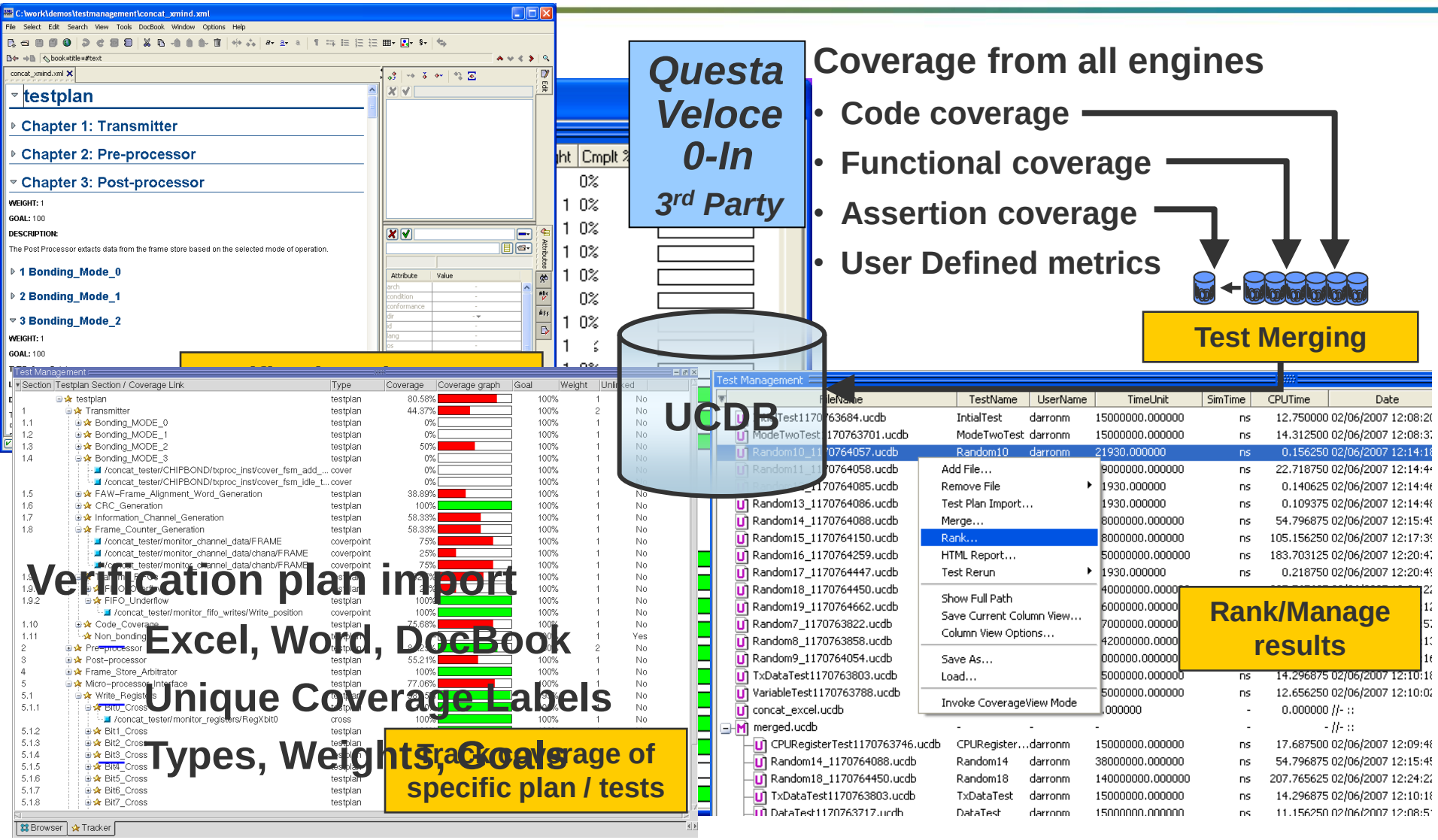
Giga/Tera Bytes of information

Thousands of tests "data explosion" lead to lack of process visibility

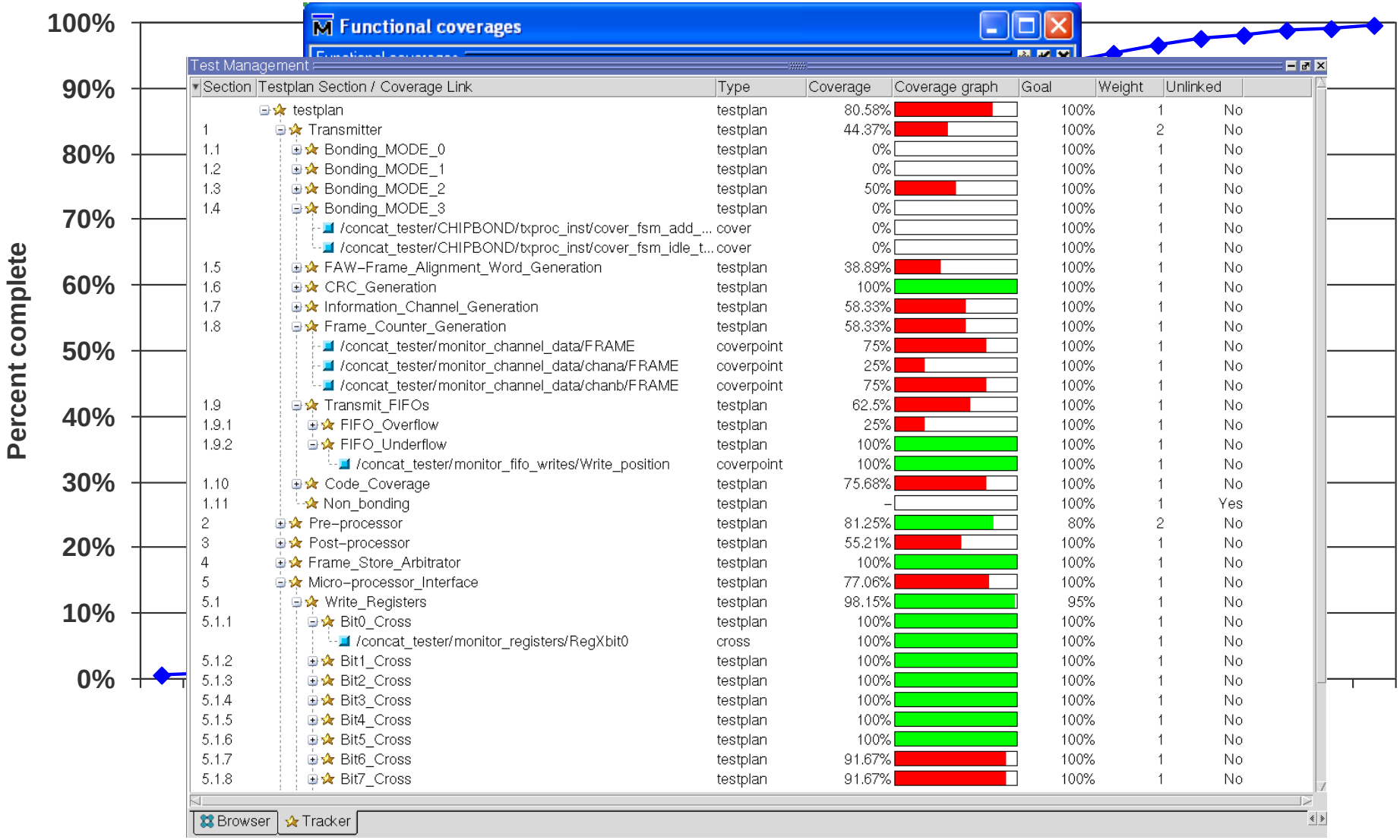
Mentor's Unique Advantage



Managing Verification Through Metrics



When Is Verification Complete?



Section	Testplan Section / Coverage Link	Type	Coverage	Coverage graph	Goal	Weight	Unlinked
	testplan	testplan	80.58%		100%	1	No
1	Transmitter	testplan	44.37%		100%	2	No
1.1	Bonding_MODE_0	testplan	0%		100%	1	No
1.2	Bonding_MODE_1	testplan	0%		100%	1	No
1.3	Bonding_MODE_2	testplan	50%		100%	1	No
1.4	Bonding_MODE_3	testplan	0%		100%	1	No
	/concat_tester/CHIPBOND/txproc_inst/cover_fsm_add_...	cover	0%		100%	1	No
	/concat_tester/CHIPBOND/txproc_inst/cover_fsm_idle_t...	cover	0%		100%	1	No
1.5	FAW-Frame_Alignment_Word_Generation	testplan	38.89%		100%	1	No
1.6	CRC_Generation	testplan	100%		100%	1	No
1.7	Information_Channel_Generation	testplan	58.33%		100%	1	No
1.8	Frame_Counter_Generation	testplan	58.33%		100%	1	No
	/concat_tester/monitor_channel_data/FRAME	coverpoint	75%		100%	1	No
	/concat_tester/monitor_channel_data/chana/FRAME	coverpoint	25%		100%	1	No
	/concat_tester/monitor_channel_data/chanb/FRAME	coverpoint	75%		100%	1	No
1.9	Transmit_FIFOs	testplan	62.5%		100%	1	No
1.9.1	FIFO_Overflow	testplan	25%		100%	1	No
1.9.2	FIFO_Underflow	testplan	100%		100%	1	No
	/concat_tester/monitor_fifo_writes/Write_position	coverpoint	100%		100%	1	No
1.10	Code_Coverage	testplan	75.68%		100%	1	No
1.11	Non_bonding	testplan	–		100%	1	Yes
2	Pre-processor	testplan	81.25%		80%	2	No
3	Post-processor	testplan	55.21%		100%	1	No
4	Frame_Store_Arbitrator	testplan	100%		100%	1	No
5	Micro-processor_Interface	testplan	77.06%		100%	1	No
5.1	Write_Registers	testplan	98.15%		95%	1	No
5.1.1	Bit0_Cross	testplan	100%		100%	1	No
	/concat_tester/monitor_registers/RegXbit0	cross	100%		100%	1	No
5.1.2	Bit1_Cross	testplan	100%		100%	1	No
5.1.3	Bit2_Cross	testplan	100%		100%	1	No
5.1.4	Bit3_Cross	testplan	100%		100%	1	No
5.1.5	Bit4_Cross	testplan	100%		100%	1	No
5.1.6	Bit5_Cross	testplan	100%		100%	1	No
5.1.7	Bit6_Cross	testplan	91.67%		100%	1	No
5.1.8	Bit7_Cross	testplan	91.67%		100%	1	No

Agenda

- Verification Overview
  - Assertion and Functional Coverage
  - Constrained Random
  - Requirements Tracing
  - Algorithmic TB InFact
  - Questa Formal
  - Questa Codelink
  - Questa VIP

The ReqTracer Solution

INFRASTRUCTURE

Requirements Management

INTEGRATION

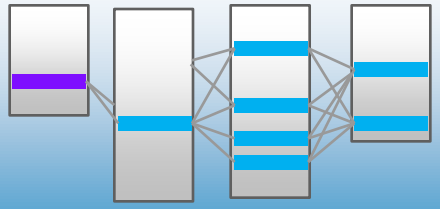
Mentor Graphics

HDL Designer™

Questa™ ADVANCED VERIFICATION

ReqTRACER™

TRACEABILITY



Dynamic Tracing

DOCUMENT

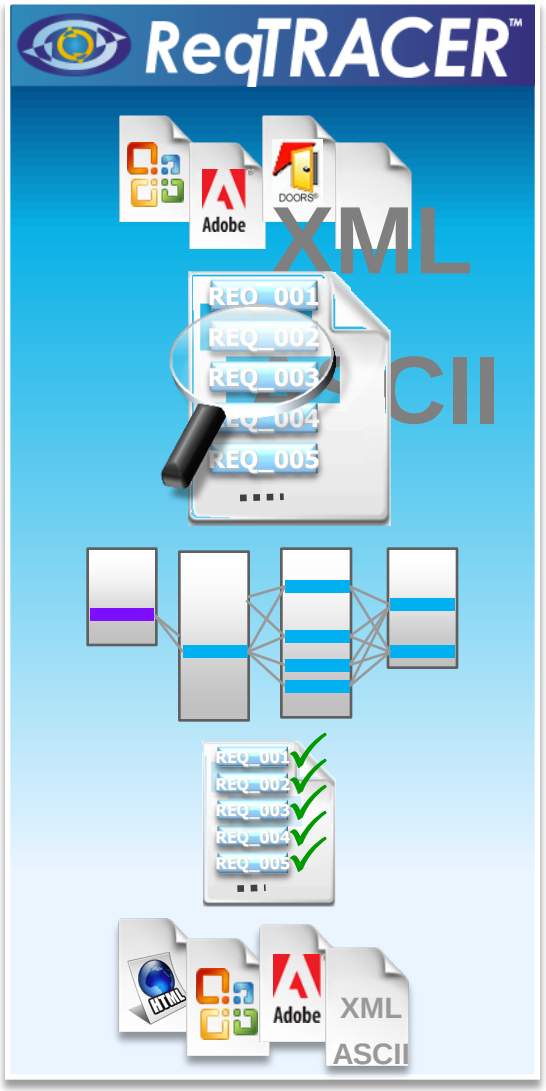
Automated Documentation

Mentor Graphics

ReqTracer in your Process

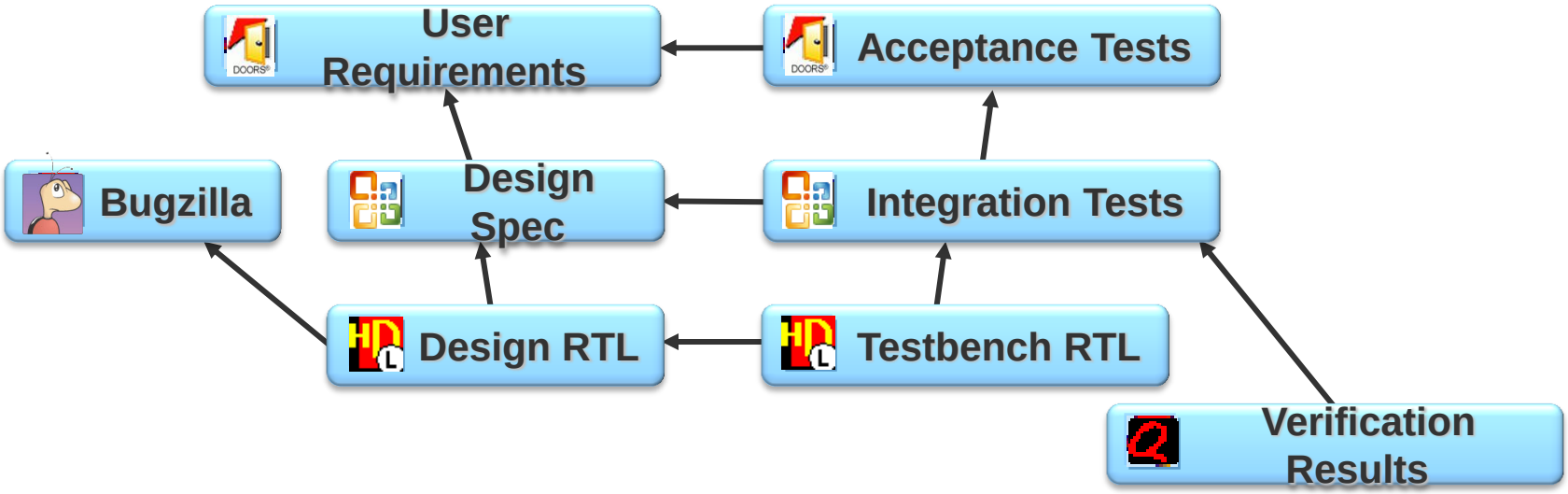
Functionality & Verification Results Meet Requirements

- Capture
  - Requirements from many interfaces & formats
  - Add documents to project
- Tag
  - Link relationships & dependencies
  - Dynamic linking
- Trace
  - Requirements ↔ design ↔ implementation ↔ verification
- Monitor/Analyze
  - Impact analysis on ECOs
  - Completeness of requirements
  - Extraneous code
- Validate
  - Status of requirements against test plan
- Report
  - Automated report generation
  - Simplifies design reviews/audits
  - Wide variety of formats

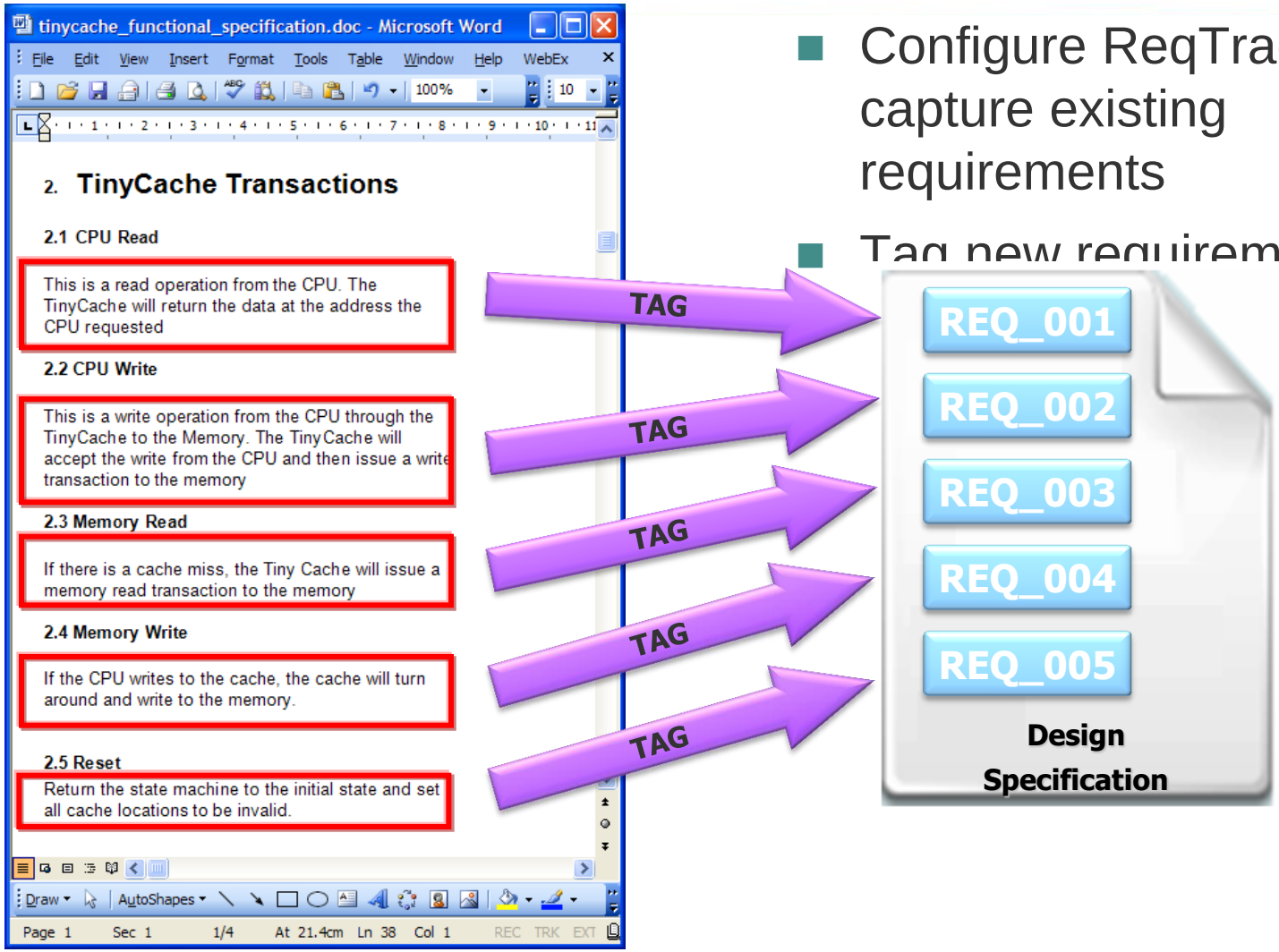


Capture

- Projects
  - Add documents to project
  - Associate documents → define coverage relationships
  - Design "covers" requirements
  - Verification results "verify" test plan

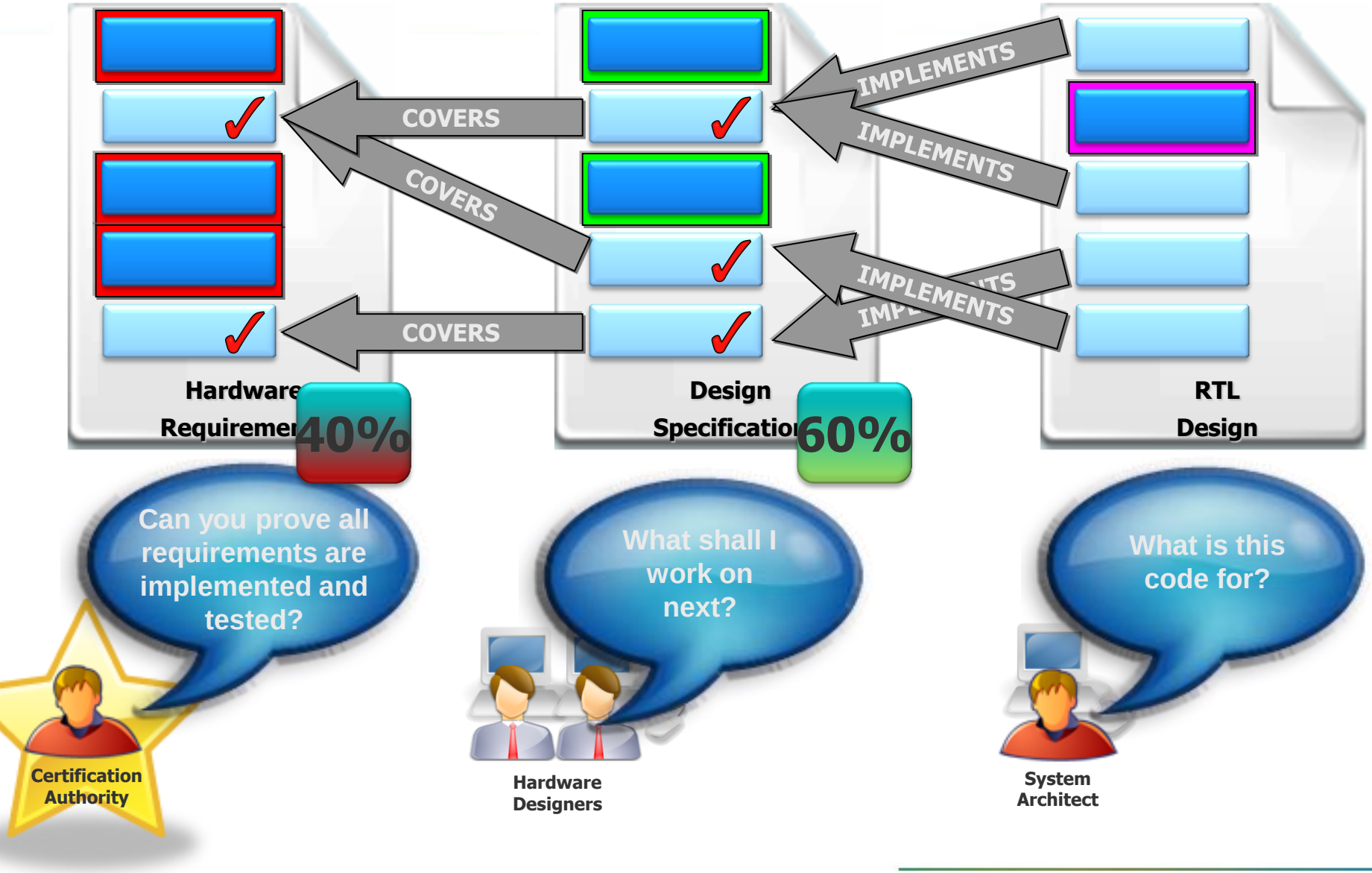


Tag

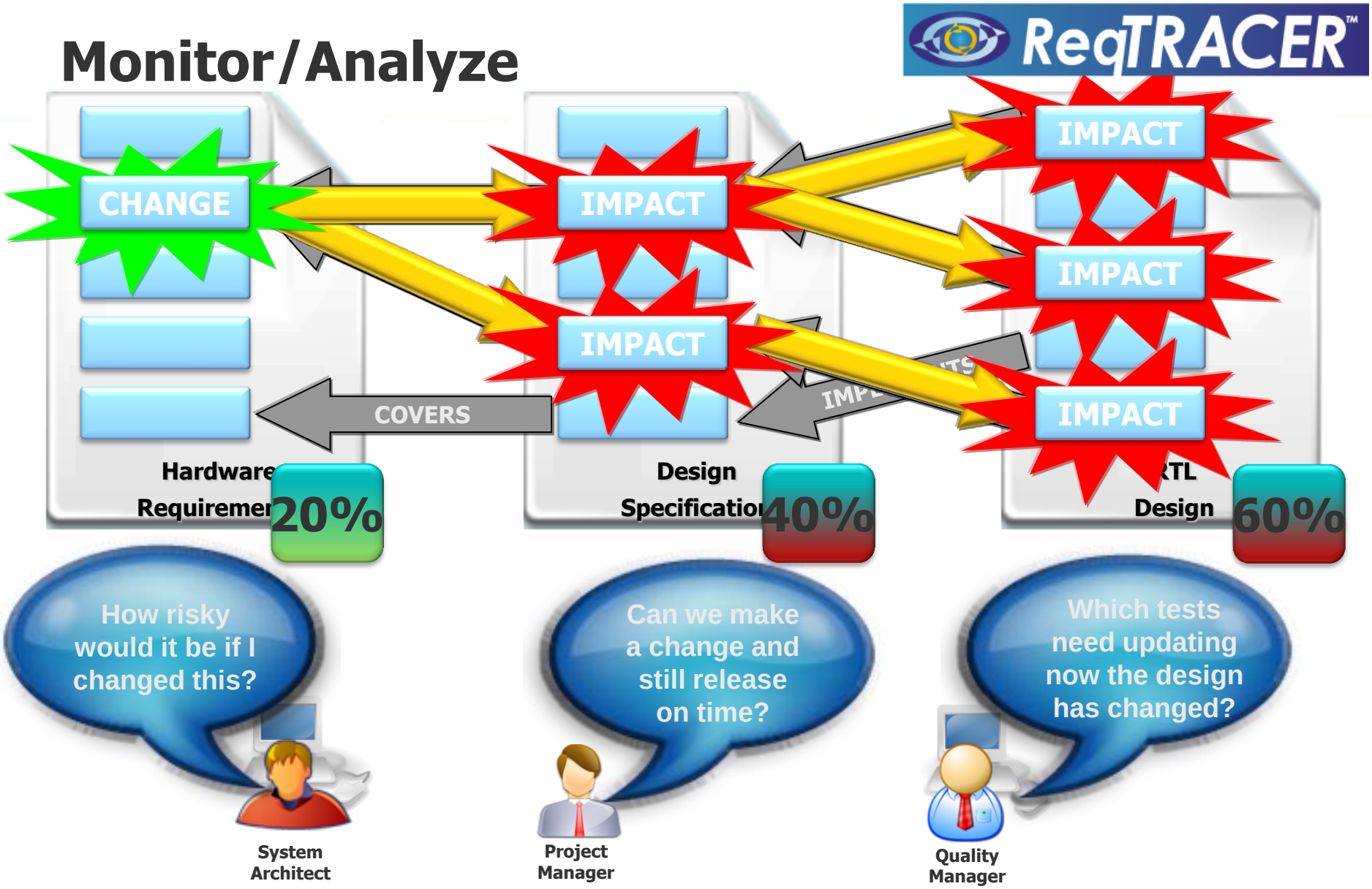


- Configure ReqTracer to capture existing requirements
- Tag new requirements

Trace



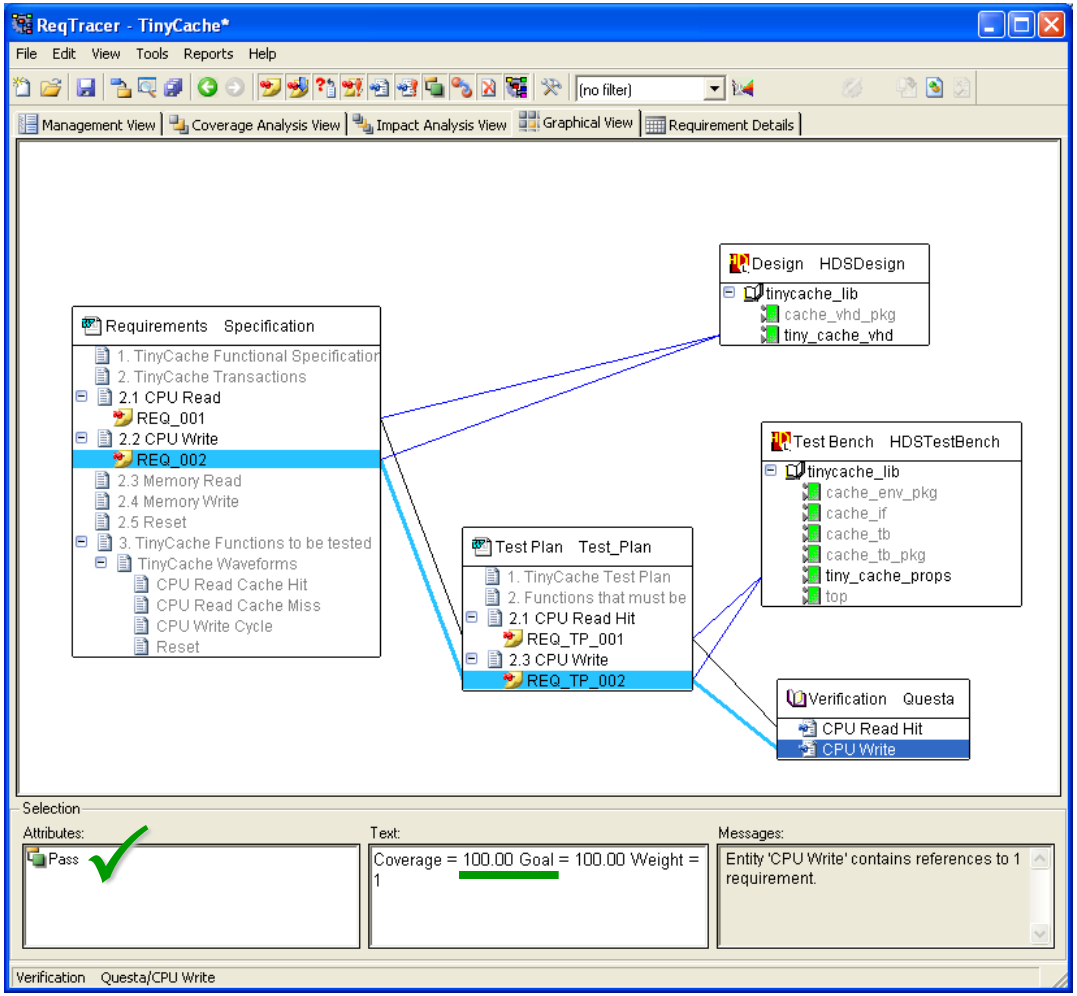
Monitor/Analyze



Validate

- **Ensure links**
  - **Requirements ↔ RTL source**
  - **Requirements ↔ test plan**
  - **Test plan ↔ testbench**
  - **Test plan ↔ verification results**

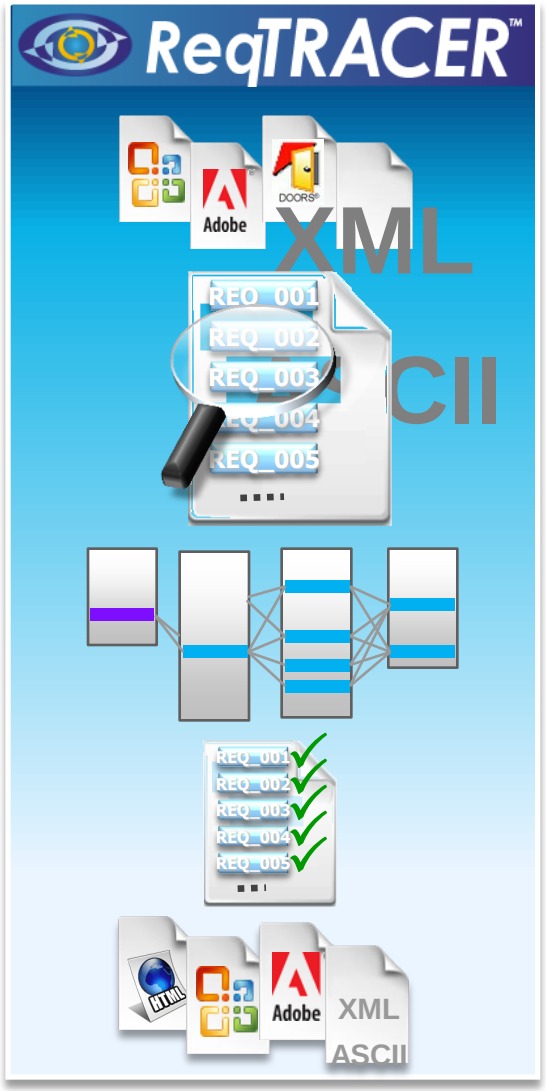
- View "Coverage" of the tests
  - **Determine Pass or Fail**
    - **Does verification test result satisfies the test plan?**



Summary

Requirements tracing is best as an active process throughout the project development

- ReqTracer
  - Aggregates requirements from multiple sources
  - Traces requirements through the HW design process
  - Provides direct ECO impact analysis
  - Automates report generation
  - Interfaces with HDL Designer
    - HDL code, version management & documentation
  - Interfaces with Questa
    - Validation of test plan
  - Satisfies DO-254 needs for certification



Agenda

- Verification Overview
  - Assertion and Functional Coverage
  - Constrained Random
  - Requirements Tracing
  - Algorithmic TB (InFact)
  - Questa Formal
  - Questa Codelink
  - Questa VIP
- Rule Checking
- Precision

Three Generations of Functional Verification

Testbench Evolution

Directed Tests

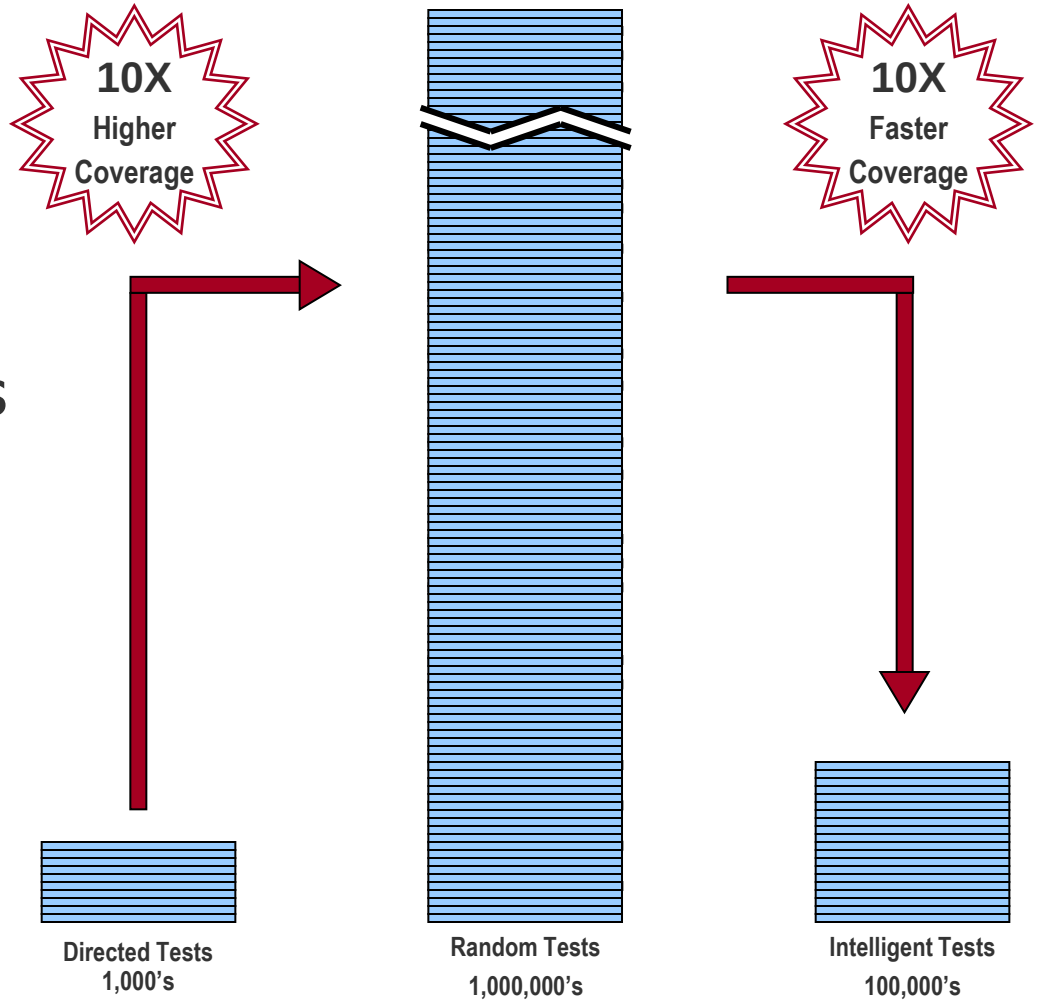
- ↑ Quality - Engineer Directed
- ↓ Quantity - Engineer Limited
- ❑ Good - But Not Scalable

Constrained Random Tests

- ↑ Quantity - Time & CPU Limited
- ↓ Quality - Redundant
- ❑ Better - But Not Sufficient

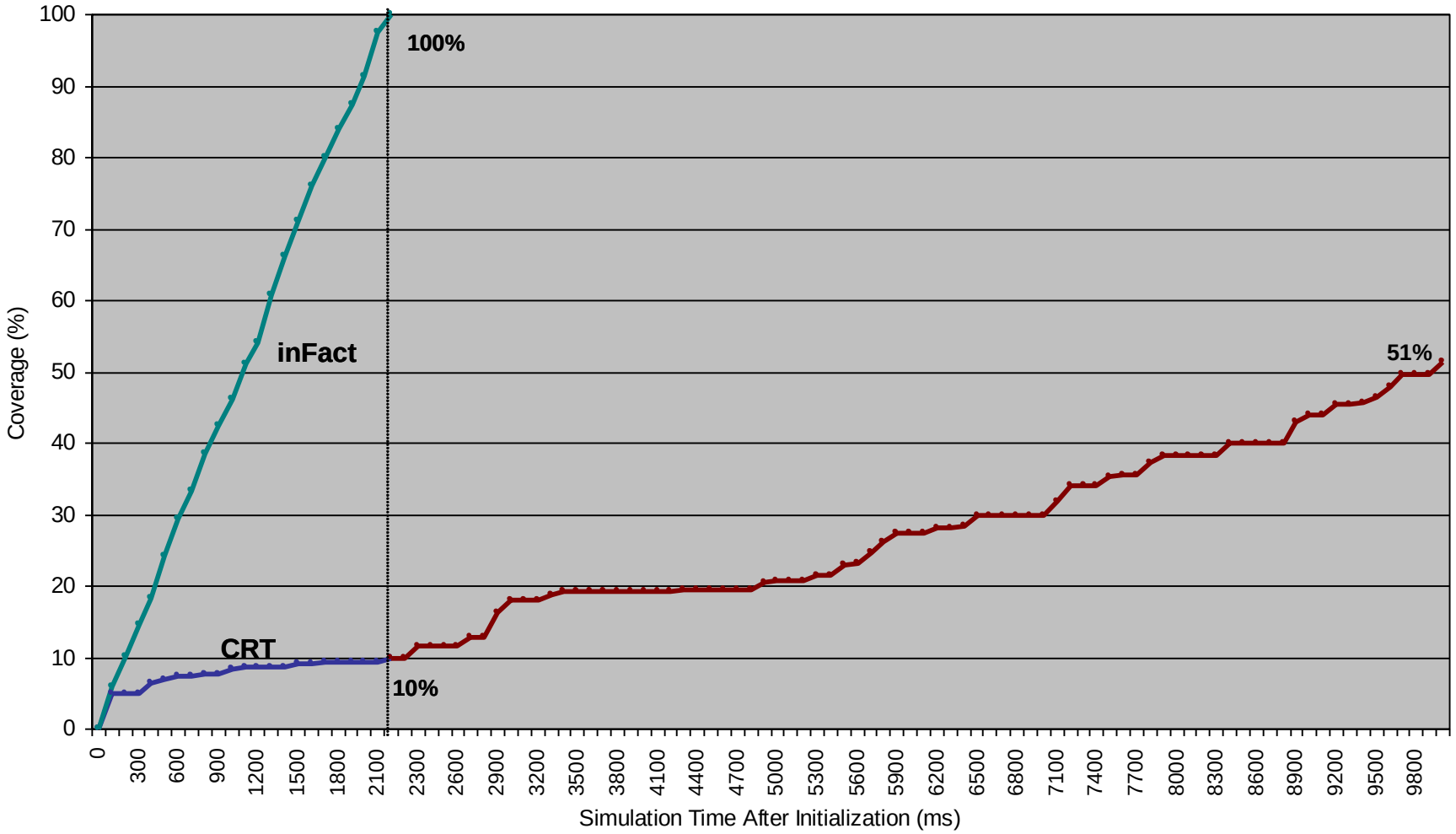
Intelligent Tests

- ↑ Quantity - High
- ↑ Quality - High
- ❑ Best - Do More With Less



Intelligent Testbench Automation

Using Questa inFact to Verify an AMBA AHB Bus Fabric



Applying iTBA

Incremental Change Results in >>10X Improvement

Re-Use Existing Testbench

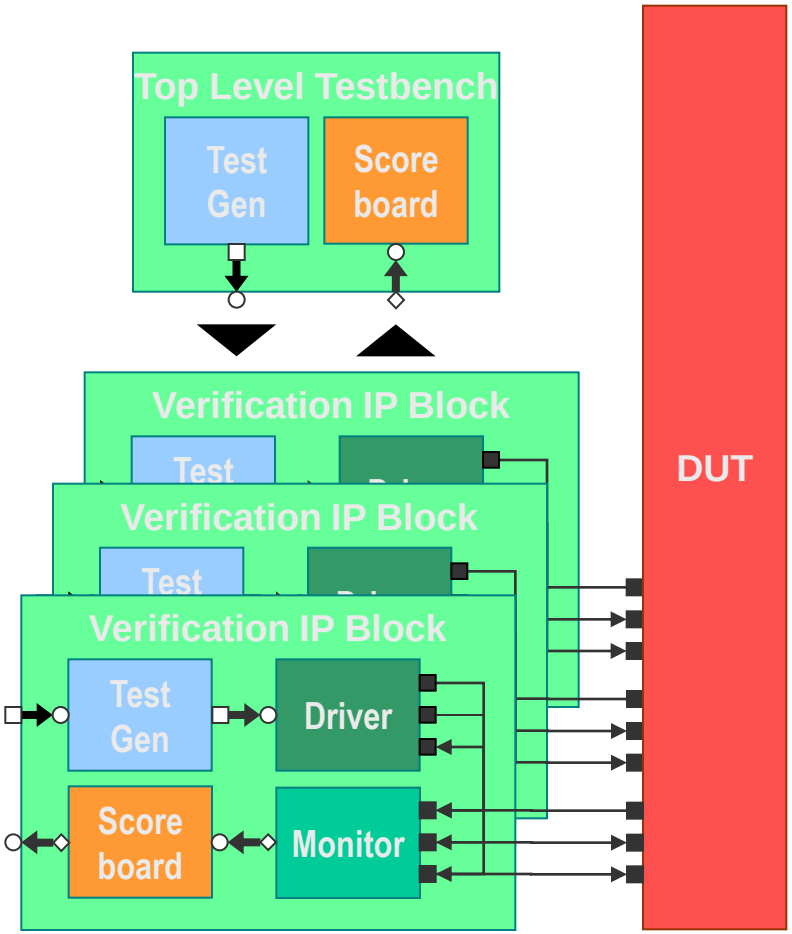
- No Change to Verification Language
- No Change to Verification Methodology
- No Change to Verification IP Blocks

Incremental Changes

- Top Level Testbench Only
- Describe Test Plan Functionality in a Graph
- Replace Existing Sequence Generator

inFact Simulation Results

- Graph Controls VIP Blocks
- Removes, Reduces, or Keeps Redundancy
- Targets Coverage, Not Just Measures It



Simplicity of iTBA

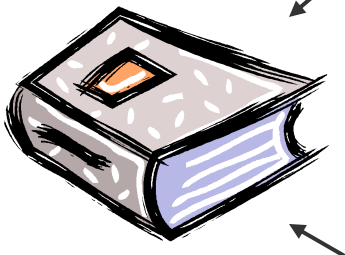
Top Layer is Described in a Standard Graph

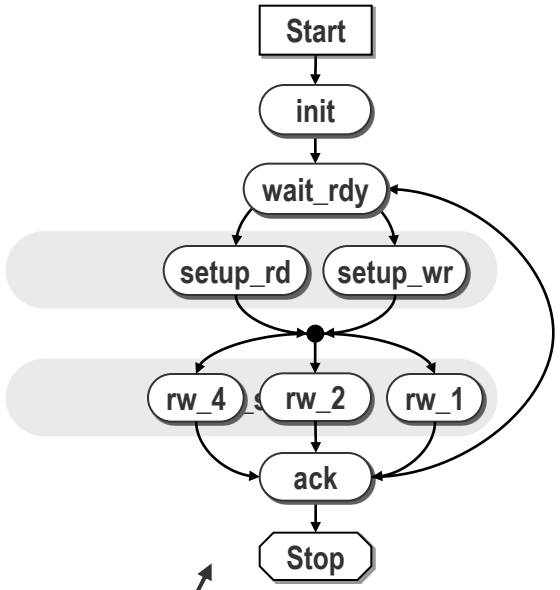
Simplicity of Graphs

- These are standard - nothing new here
- Used since 1960's for compiler & software test
- Easy to write, read, and control - visual
- Contain highly compressed information

Origin of Graphs

- Pre-date Verilog, VHDL, etc.
- John Backus - IBM Fellow in 1963
- Also invented Fortran and Algol






```
Start = init   repeat ( wait_rdy  Rw_opts  Rw_size  ack ) ;

Rw_opts = setup_rd | setup_wr ;

Rw_size =       rw_1 | rw_2 | rw_4  ;
```

Power of iTBA

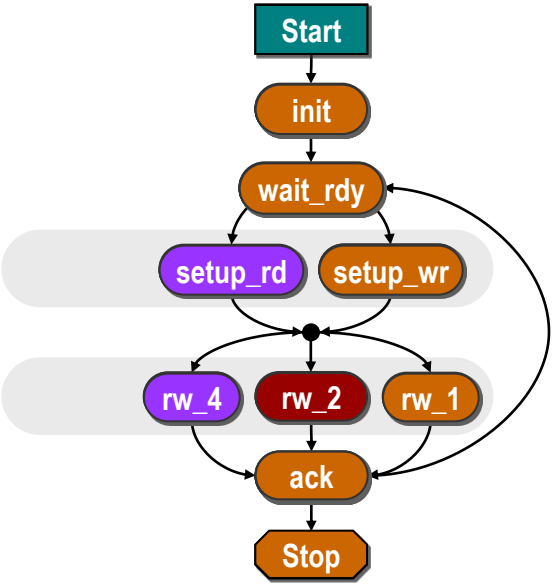
Questa inFact Adapts and Learns During Simulation

Power of Algorithms

- Original graph based test applications were limited to deterministic systems
- Used by many computer companies to test software & compilers
- But VLSI hardware systems are non-deterministic
- *How to automatically generate deterministic stimulus for non-deterministic systems?*

Uniqueness of Questa inFact

- Extend graph technology to VLSI design verification
- Apply breakthroughs in automata based math
- Close coverage >>10X faster



Creating Graphs

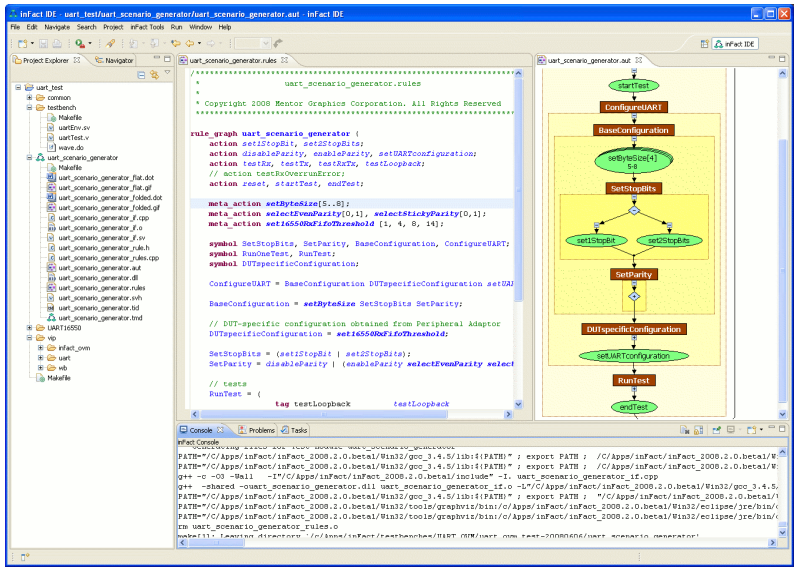
Easy to Use

Interactive Development Environment

- Graphical User Interface
- Automatically Generates Graphs from Text

Integrate With Questa Simulation

- Compile Testbenches
- Link Graphs to Waveforms
- Unmatched Debugging

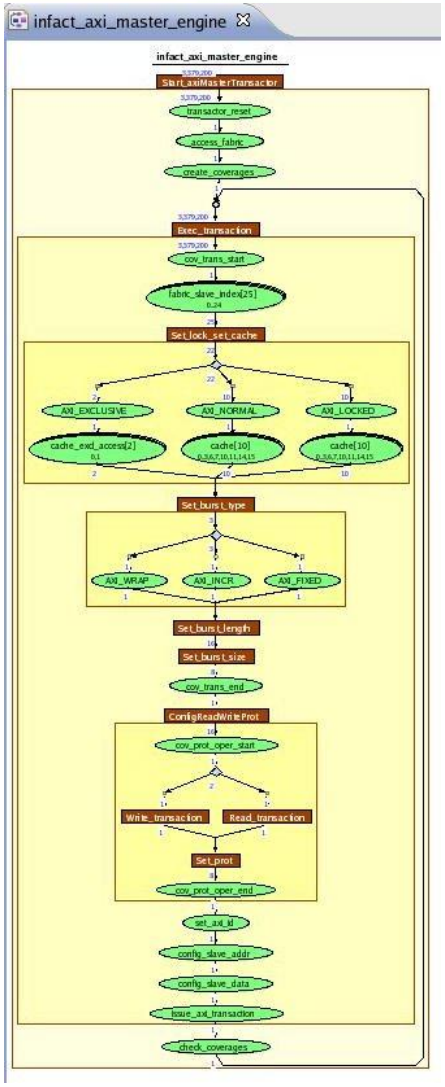


inFact AXI Master

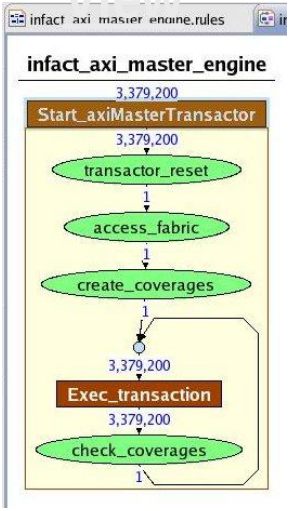
Rule Code, Expanded Graph, and Collapsed Graph

*infact_axi_master_engine.rules

```
/*****************************************************************
 * infact_axi_master_engine.rules
 *****************************************************************

rule_graph infact_axi_master_engine {

 // System Actions
  action transactor_reset, access_fabric;
  action issue_axi_transaction;

  // Coverage Labels
  action create_coverages, check_coverages;
  action cov_prot_oper_start, cov_prot_oper_end;
  action cov_trans_start, cov_trans_end;

  // AXI ID, read/write, addr/data, burst attrib, lock
  action set_axi_id;
  action dir_read;
  action dir_write;
  action config_slave_data;
  action config_slave_addr;
  action AXI_WRAP, AXI_INCR, AXI_FIXED;
  meta_action burst_size[0..7];
  meta_tag m_burst_size%d burst_size[0..7];
  meta_action burst_length[1..16];
  action AXI_NORMAL, AXI_EXCLUSIVE, AXI_LOCKED;

  // AXI cache support
  meta_action cache[    0x0, //Non_cacheable_non_bufferable
                        0x1, //Bufferable_only;
                        0x2, //Cacheable_no_allocate;
                        0x3. //Cacheable bufferable no alloca
  // .... additional definitions ....
  // High level graph definitions
  Read_transaction = dir_read;
  Write_transaction = dir_write;
  ConfigReadWriteProt = cov_prot_oper_start
            (tag write_prob Write_transaction |
             tag read_prob Read_transaction)
            Set_prot cov_prot_oper_end;
  Exec_transaction =
        cov_trans_start
        fabric_slave_index
        Set_lock_set_cache
        Set_burst_type
        Set_burst_length
        Set_burst_size
        cov_trans_end
        ConfigReadWriteProt
        set_axi_id
        config_slave_addr
        config_slave_data
        issue_axi_transaction
        ;

  Start_axiMasterTransactor =
        transactor_reset
        access_fabric
        create_coverages
        repeat{ Exec_transaction check_coverages };

  infact_axi_master_engine = Start_axiMasterTransactor;
}
```



Collapsed View



Intelligent Testbench Automation

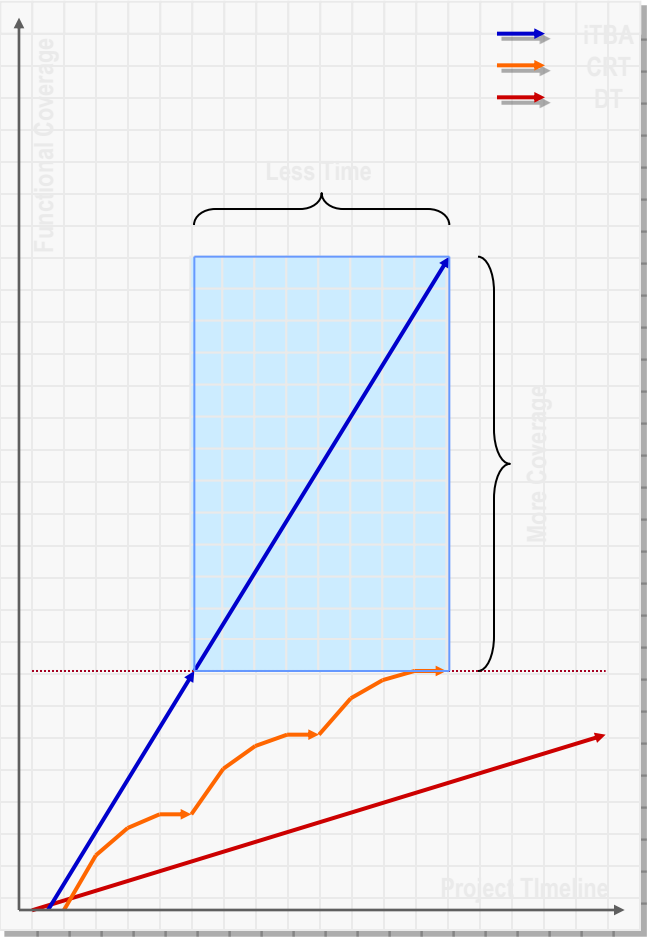
Achieve Target Coverage >>10X Faster

Achieve Target Coverage in Less Time

- Verify all functionality called out in test plan
- Reduce or remove unwanted redundancy
- Confirm test plan coverage with UCDB

Leave Time to Expand Coverage

- Add tests to cover more functionality
- Target test generation to desired areas
- Hunt for bugs so they don't escape into mfg

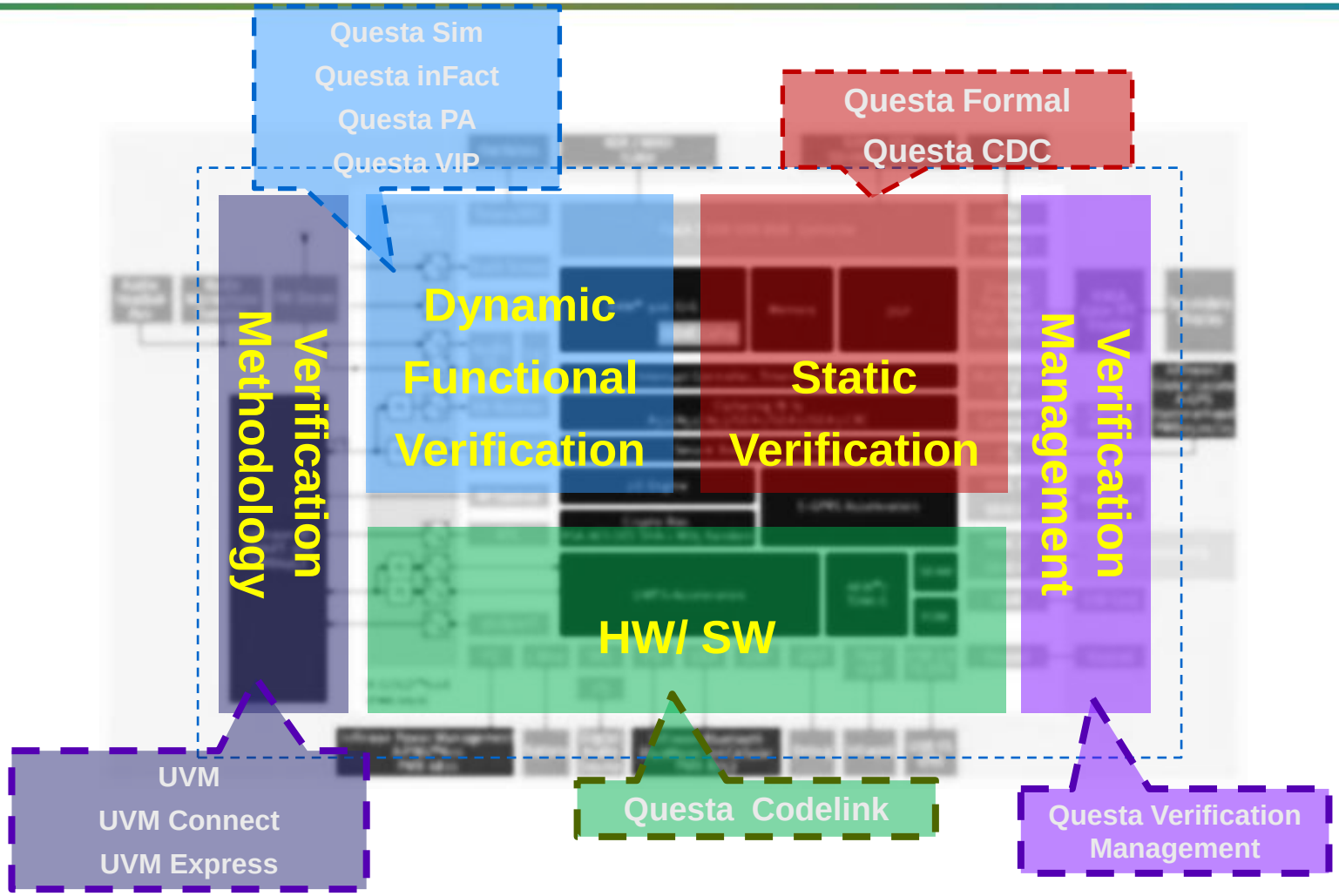


Agenda

- Verification Overview
  - Assertion and Functional Coverage
  - Constrained Random
  - Requirements Tracing
  - Algorithmic TB (InFact)
  - Questa Formal
  - Questa Codelink
  - Questa VIP

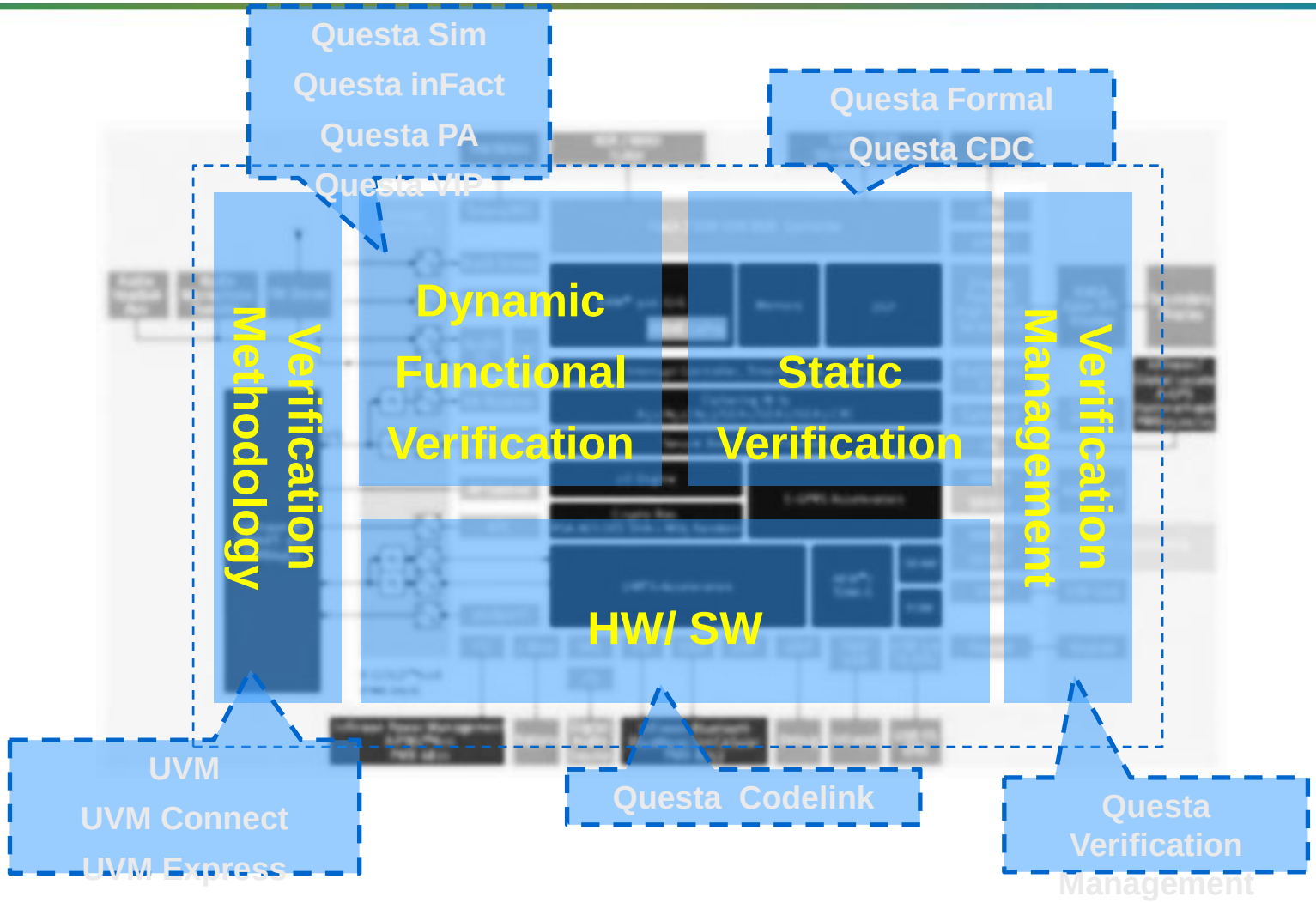
Questa Verification Platform

Best In Class Engines



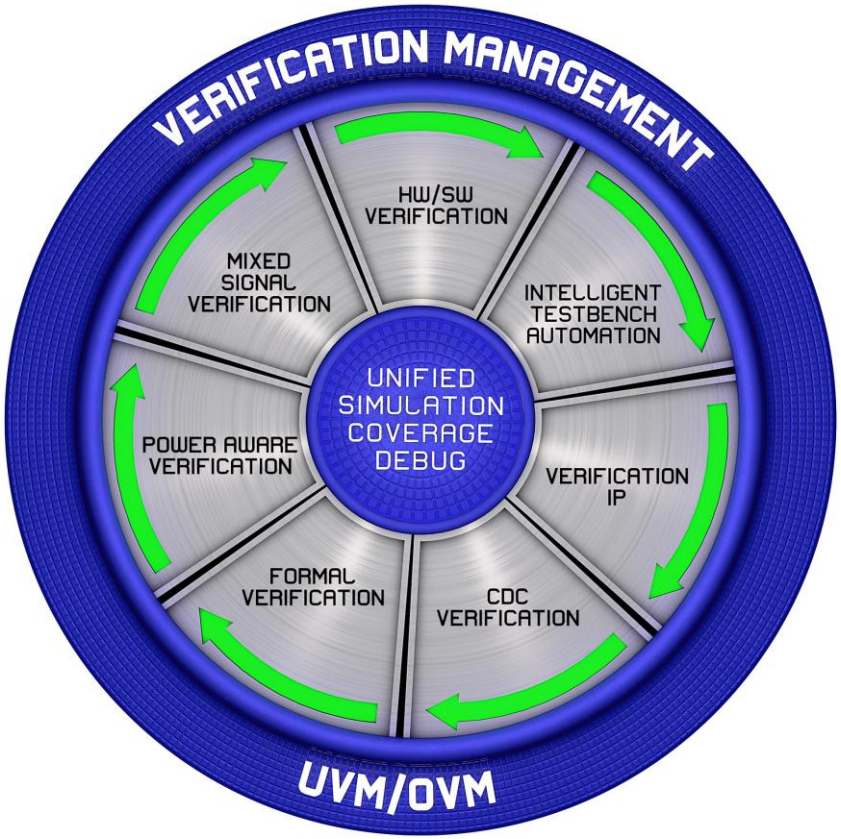
Questa Verification Platform

Best In Class Engines - Unified Front End Analysis & Compile



Questa Verification Platform

- Comprehensive integrated SOC verification platform
  - Best in class engines
  - Integrated
  - Comprehensive debug analysis
- Industry Leading SOC Verification Solutions
  - Coverage Closure Solution
  - Low Power Verification Solution
  - Software Driven Verification Solution
- Standards Leadership
  - Driving the evolution of IEEE standards
  - Major donations to Accellera UVM
  - Accellera UCIS from Mentor UCDB



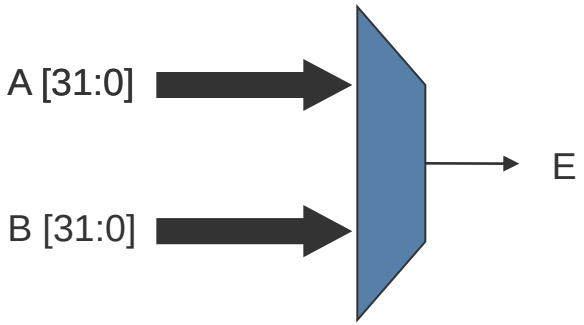
Simulation Alone Is Insufficient

MYTH:

"If we only had a faster simulator, if we only had a better coverage model, things would have been better."

- **Simulation algorithms, no matter how good, have inherent limitations**
- **0-In formal verification directly addresses these limitations**

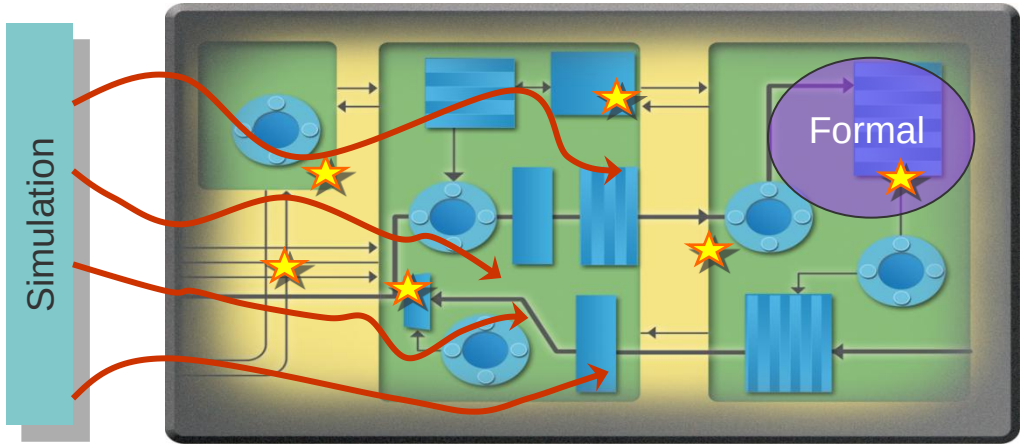
assert always (A==B) <-> E ;



2^{64} vectors * 1 vector every µs = 584,941 years

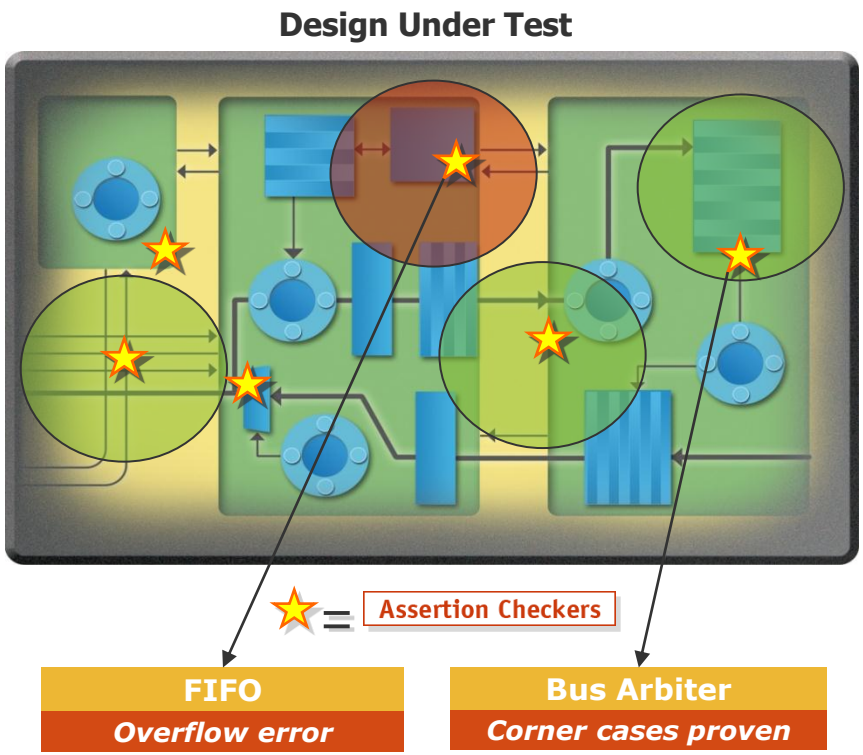
Simulation vs. Formal

- Simulation
  - Let's see what happens if I apply this stimulus?
  - Probabilistic whether or not stimulus exposes interesting behavior
- Formal
  - What stimulus do I have to apply to make this happen?
  - Use mathematics to solve for the stimulus

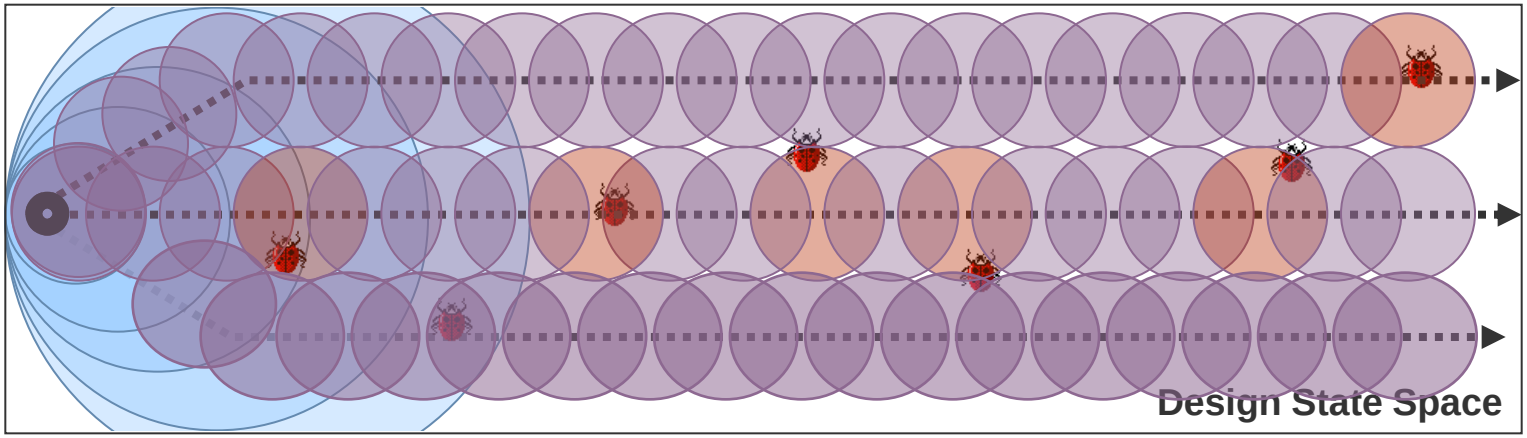


Static Formal Verification

- Static Formal
  - Takes a single start-state
  - Performs exhaustive verification of an assertion(s)
  - Provides proofs
    - 0in_prove
  - Provides counterexamples
    - 0in_confirm
- Can be used early in the design cycles
  - Does not require a testbench
  - Ensures critical functionality is correct

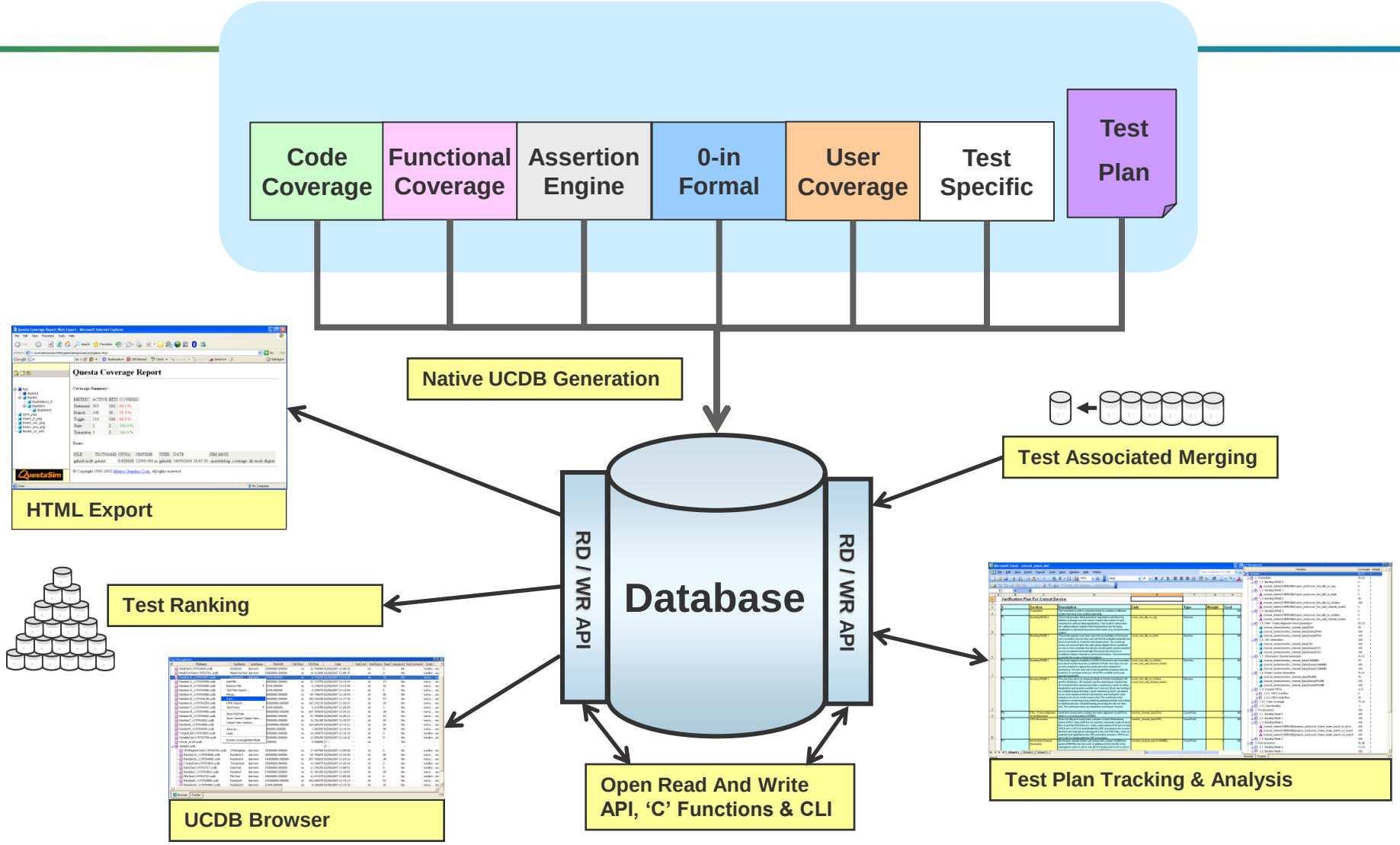


Dynamic Formal Verification



- Overcomes inherent limitation of formal techniques
- Combines formal verification and simulation to cover more of the design
- Unified coverage ties techniques together

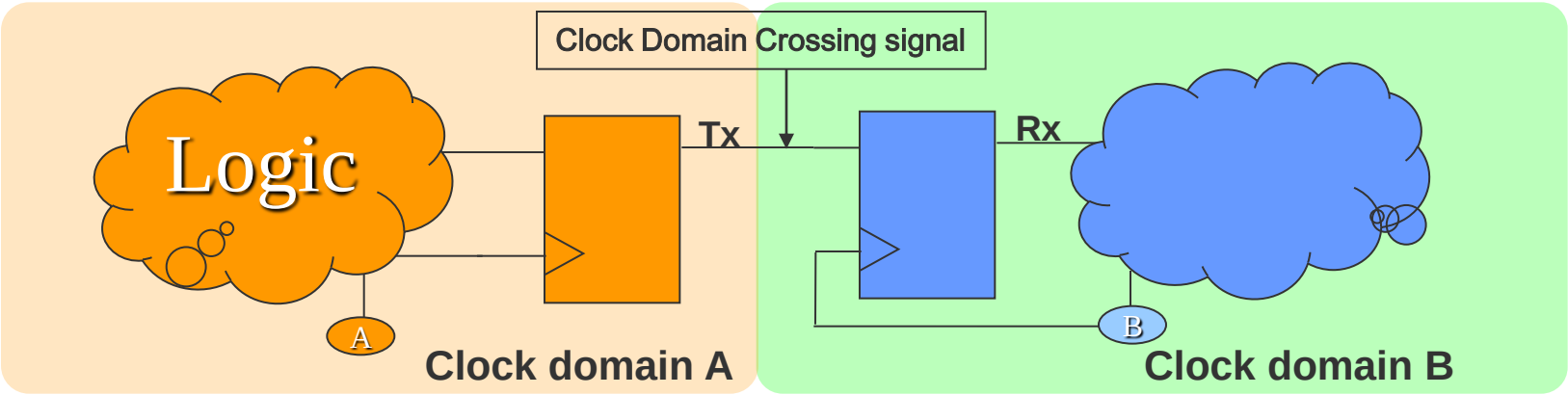
Unified Coverage Data Base

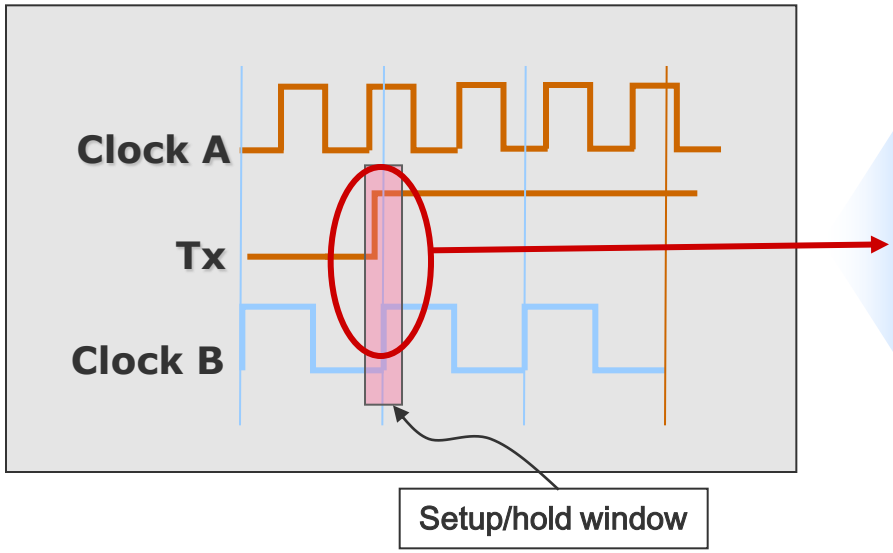


Agenda

- Verification Overview
  - Assertion and Functional Coverage
  - Constrained Random
  - Requirements Tracing
  - Algorithmic TB (InFact)
  - Questa Formal
    - Questa CDC
  - Questa Codelink
  - Questa VIP

Domains Require Special Attention

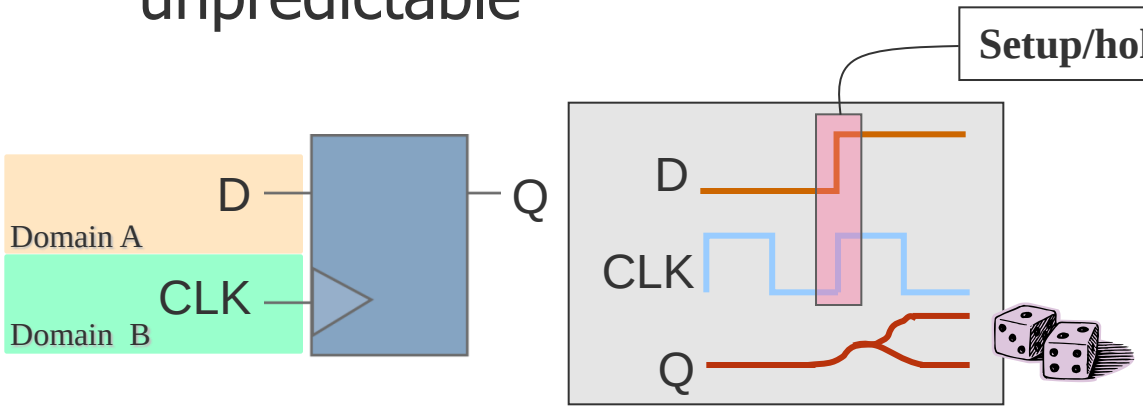




Signals that cross asynchronous clock domains (CDC signals) WILL violate setup and hold conditions

Clock Domain Crossing (CDC) Signals Cause Metastability

- When setup/hold conditions are violated, the output of a storage element becomes unpredictable



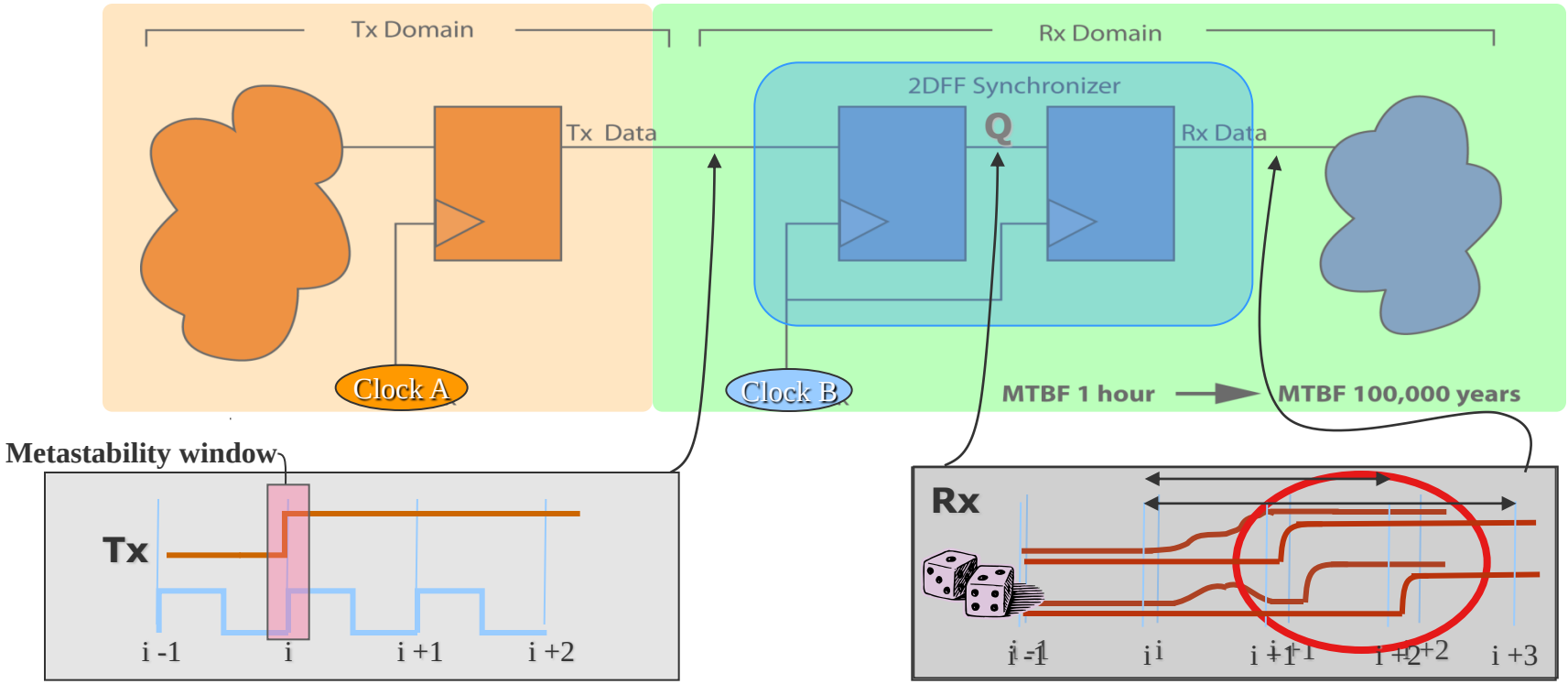
$$MTBF = \frac{1}{f_{clk} \times f_{in} \times t_d}$$

f_{clk} = Clock Frequency
f_{in} = Input Signal Frequency
t_d = Duration of critical time window

- This effect is called metastability

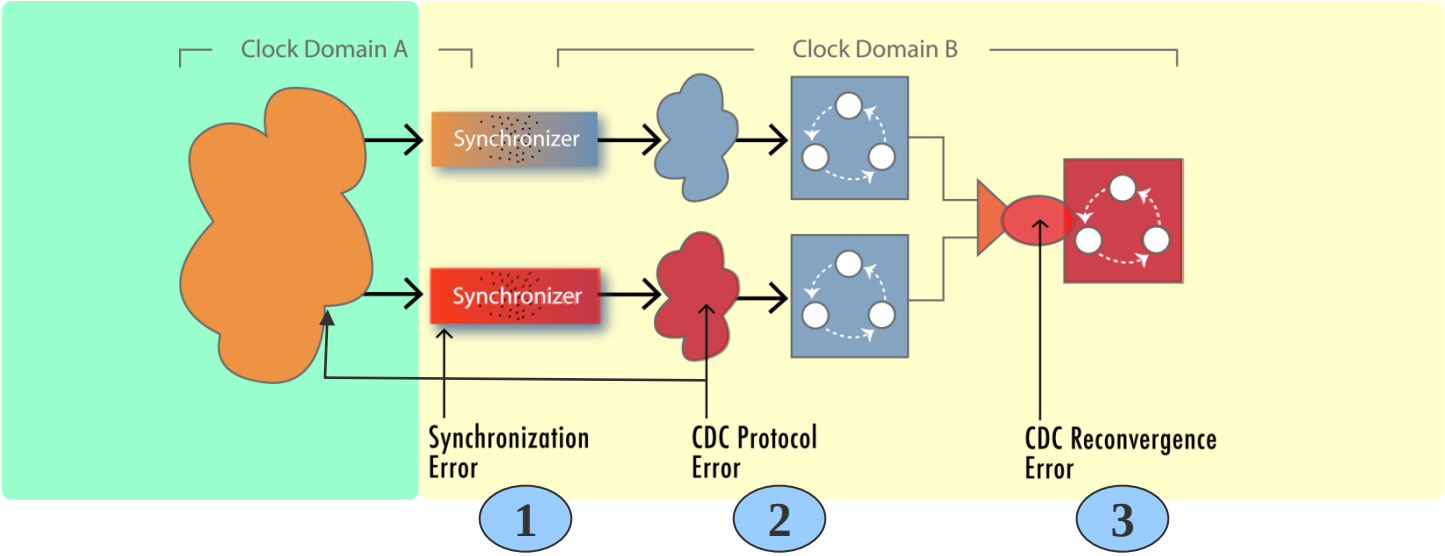
Metastability is UNAVOIDABLE in designs with multiple asynchronous clocks

Designers Use Synchronizers to Isolate Metastability



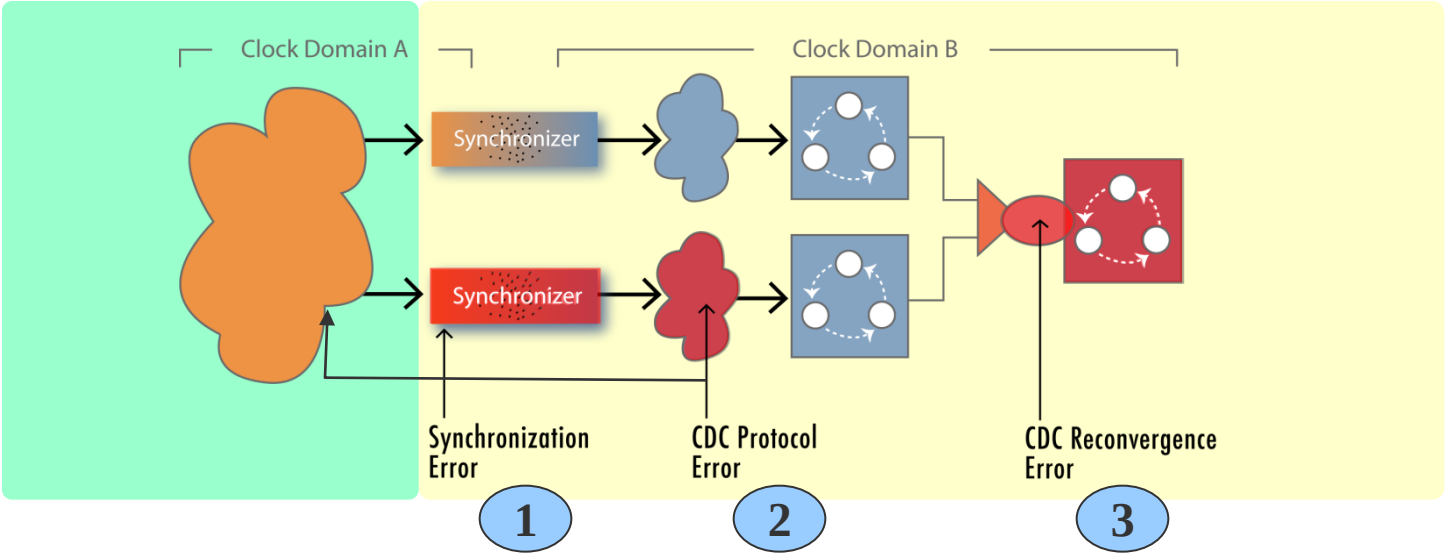
When metastability occurs, the delay through a synchronizer becomes unpredictable

Complete Anatomy of CDC Bugs



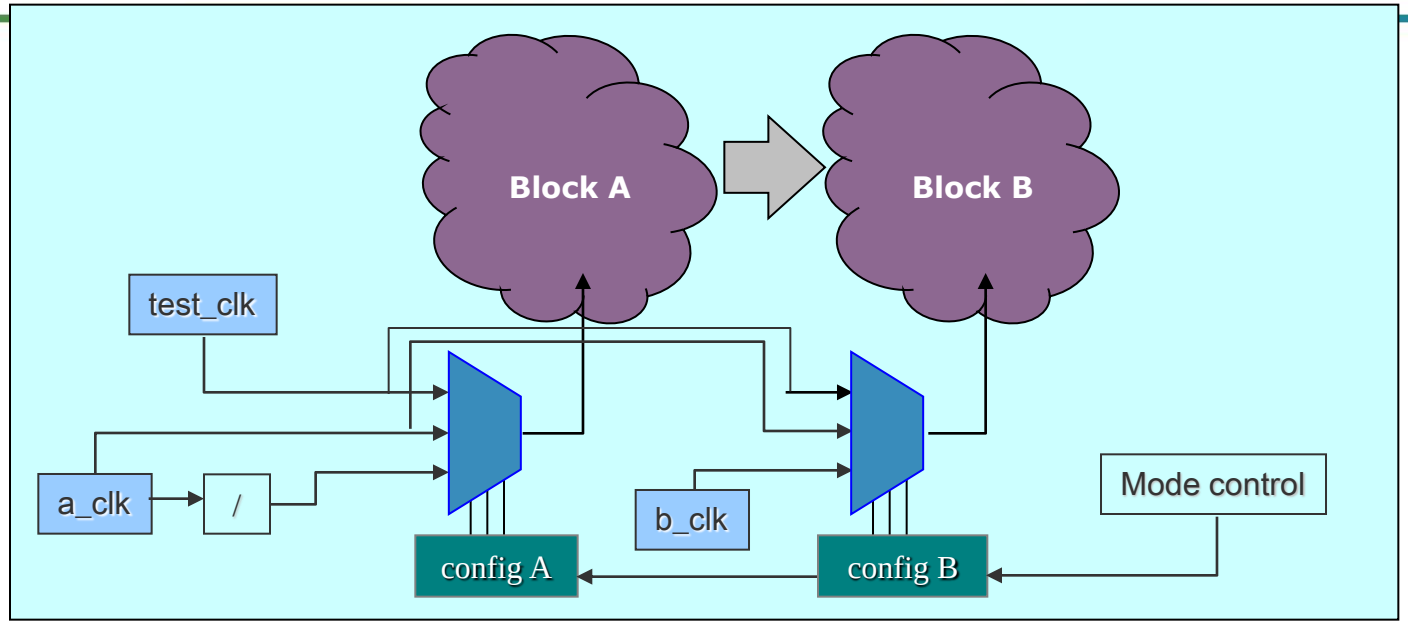
1. Missing or incorrect synchronizer
2. Incorrectly implemented CDC protocol
3. Design does not account for nondeterministic delay through synchronizers (a.k.a. reconvergence error)

The Benchmark CDC Verification Solution



1. Provides complete structural CDC analysis
2. Provides complete protocol verification to ensure correct transfer of data across synchronizers
3. Supports metastability injection in simulation to enable detection of reconvergence errors

Step 1. Structural CDC Analysis

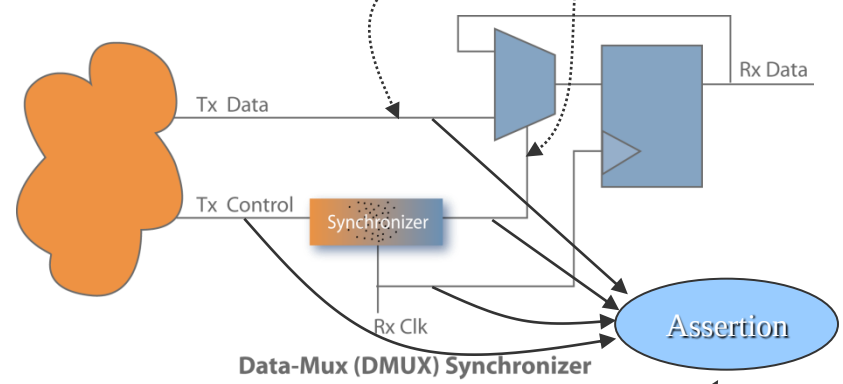


- Identify all primary asynchronous clocks
- Identify the clock distribution/control strategy
  - Derived clocks, clock dividers
  - Clock gating, on/off schemes

Provides Automatic Protocol Checks

- CDC protocols ensure that data is predictably transferred between two clock domains
- The 0-In® CDC verification solution automatically generates assertions to capture these protocols

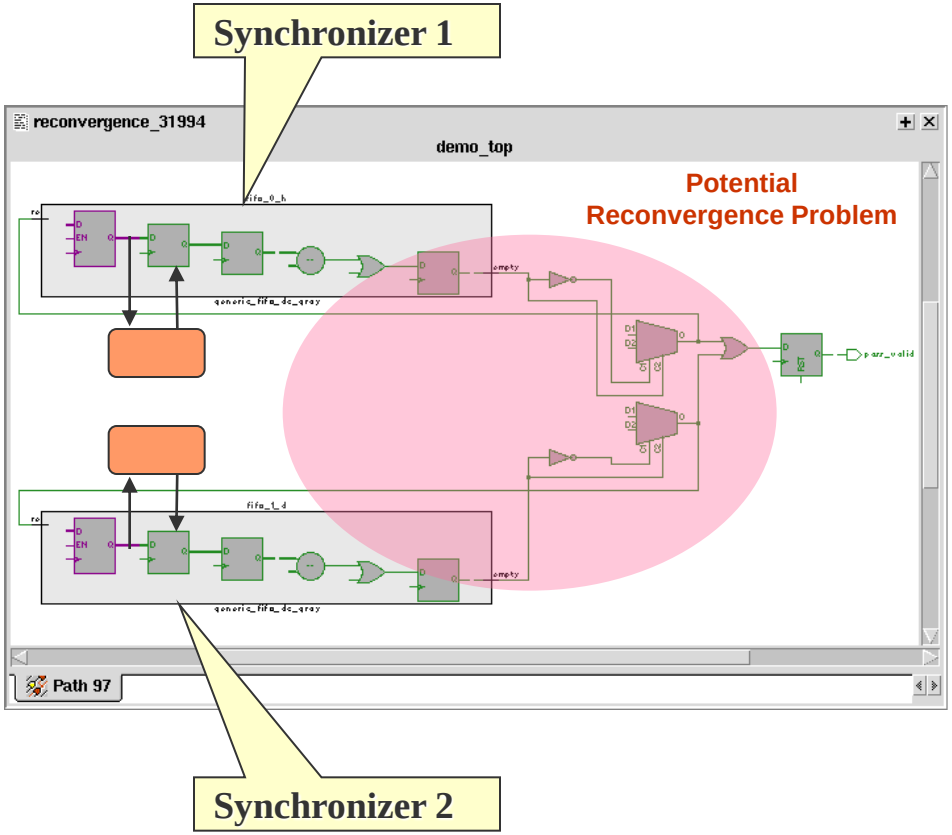
Protocol: When a transmitter's data select signal crosses a clock domain and drives the select input of a data multiplexer in the receiver, it must be held stable long enough for the signal to be sampled reliably by the receiver and the data must remain stable while the data select signal asserts.



Only when CDC protocols are represented as assertions can they be used both in simulation and in formal verification

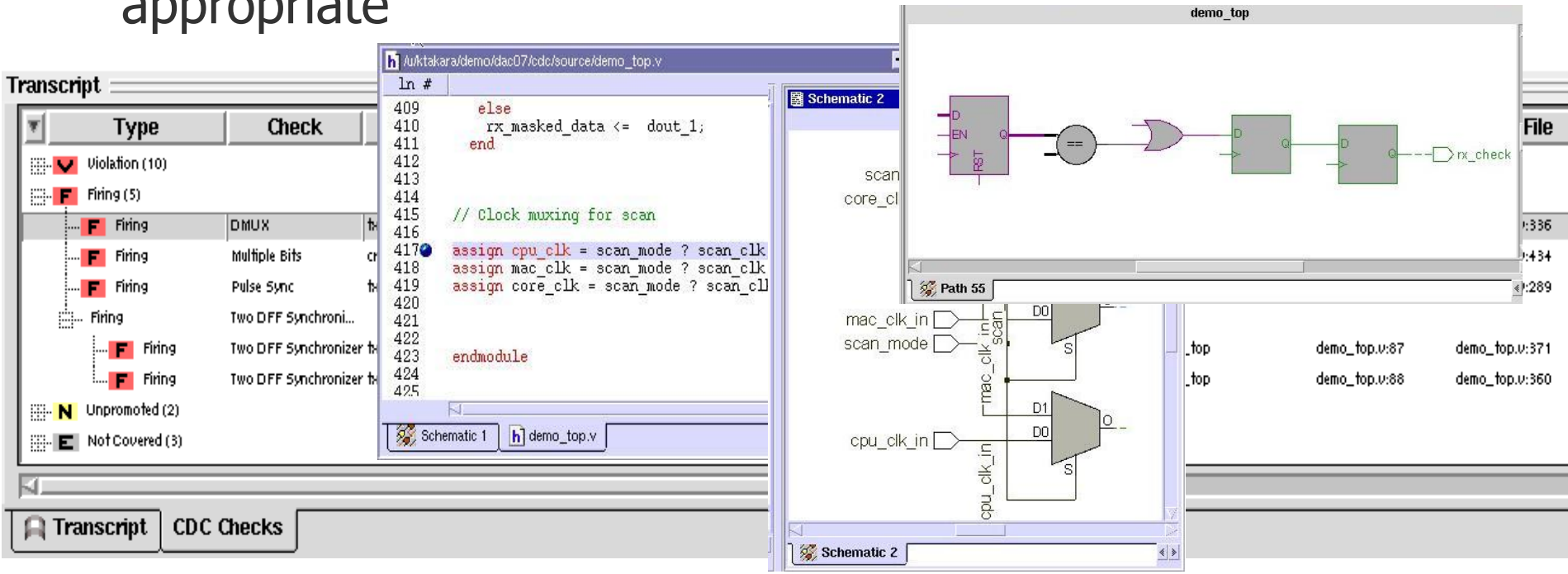
Performs Static Reconvergence Analysis

- Automatically identifies potential reconvergence problems in logic
- Generates metastability injection assertions to be used in simulation
- Supports both combinational and sequential reconvergence

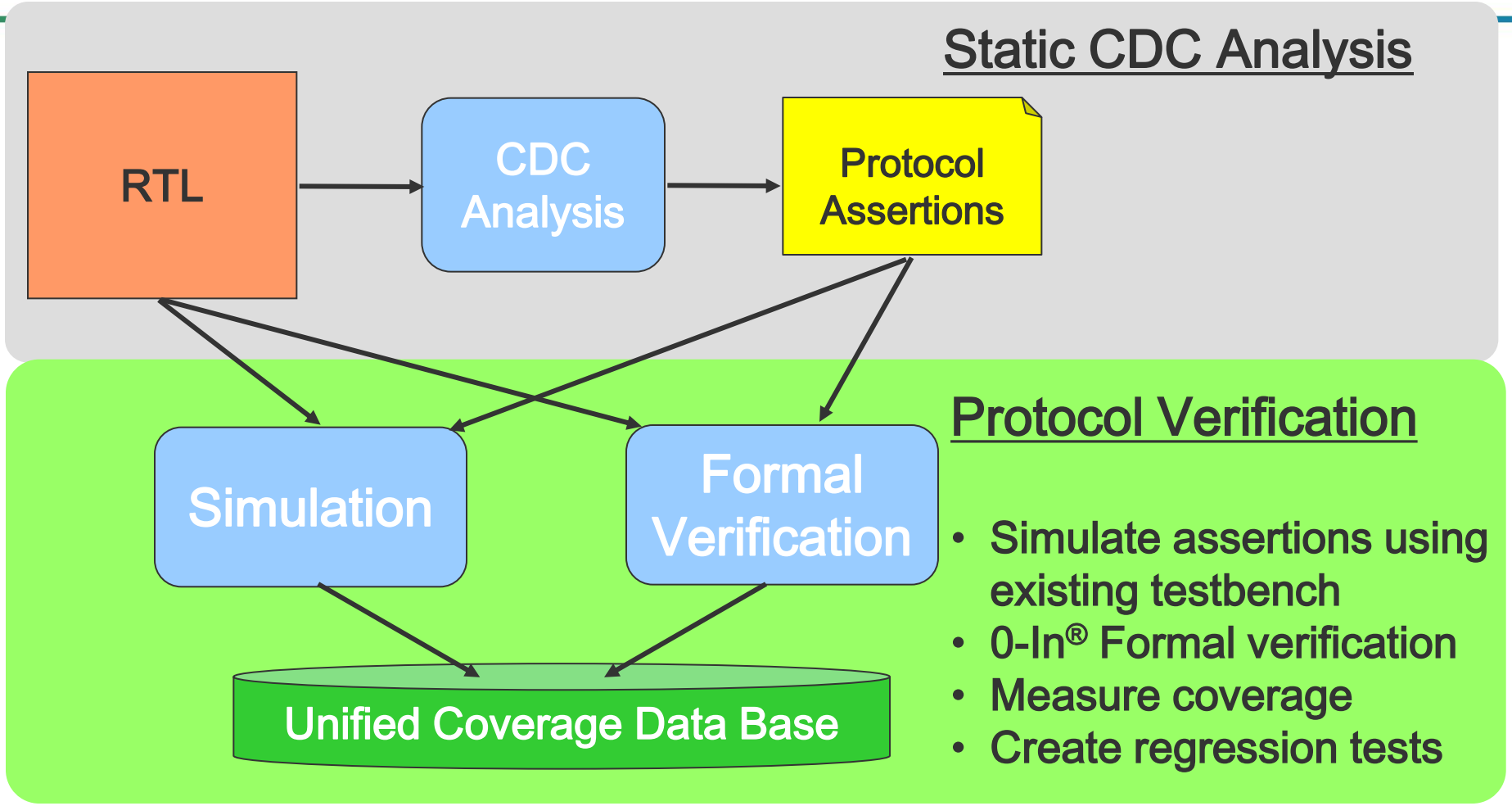


Easy to Use Reporting and Debugging

- Provides a results overview window and details for all CDC issues
- Uses generated schematics (with user control) where appropriate

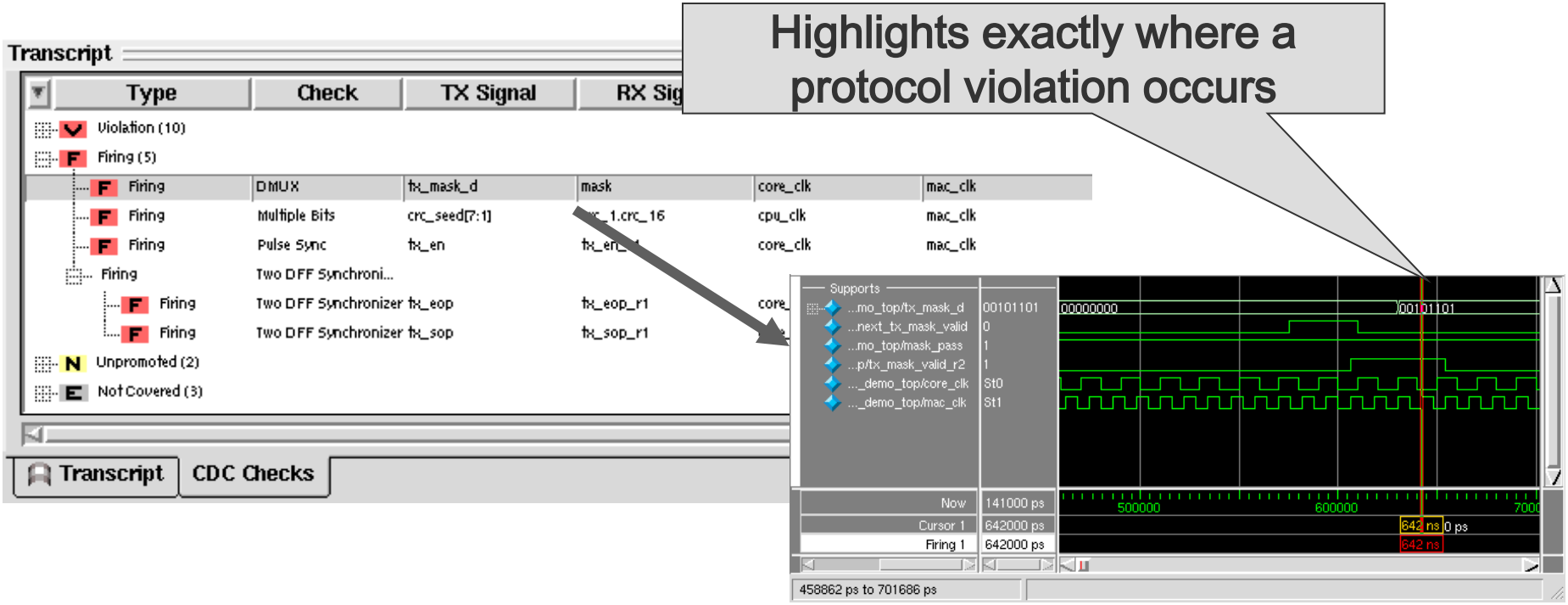


Step 2. CDC Protocol Verification



Verifying Protocol Assertions in Simulation

- All CDC protocol violations are shown in CDC analysis window
- Debug is performed in simulation environment



Coverage Metrics for Protocol Assertions

Using assertions to capture CDC protocols enables coverage metrics to be collected

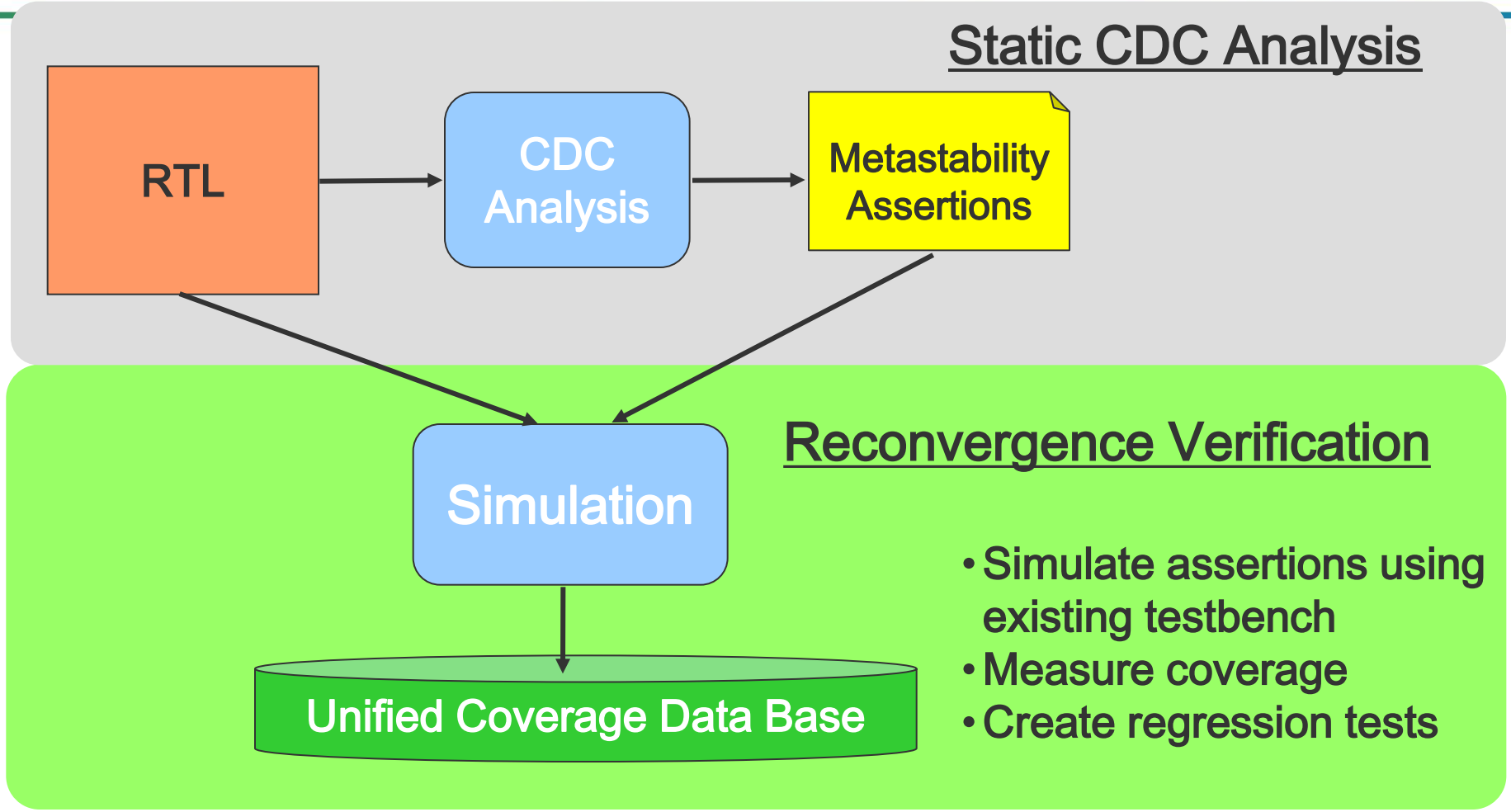
Instance	Graph	Coverage
Checkers (16)		
cdc_sync two_dff_5840 (2)		50.0
cdc_sync two_dff_81824 (2)		50.0
cdc_sync two_dff_11314 (2)		50.0
cdc_sync two_dff_54238 (2)		50.0
cdc_sync two_dff_52495 (2)		100.0
cdc_sync pulse_sync_13276 (2)		50.0
cdc_hamming_distance_one bus_two_dff_80275 (1)		100.0
cdc_hamming_distance_one bus_two_dff_94611 (1)		100.0
cdc_hamming_distance_one bus_two_dff_53361 (1)		100.0
cdc_hamming_distance_one bus_two_dff_62001 (1)		100.0
cdc_sample no_sync_47305 (5)		60.0
cdc_sample no_sync_2628 (5)		40.0
Evaluated		100.0
All One		0.0
All Zero		100.0
All Zero to One		0.0
All One to Zero		0.0
cdc_sample no_sync_31547 (5)		40.0
cdc_sample multi_bits_76265 (5)		100.0
cdc_dsel dmux_30303 (2)		50.0
cdc_dsel dmux_12402 (2)		100.0

Coverage metrics:

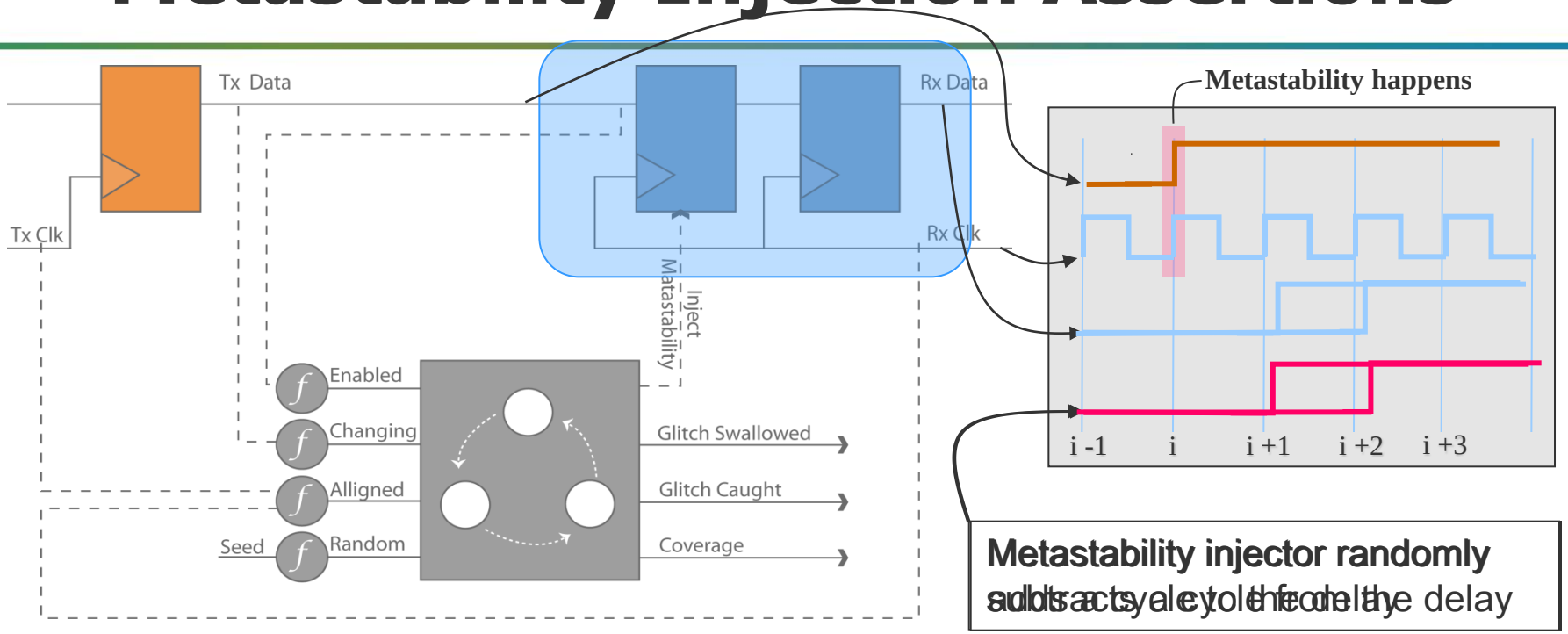
- Ensure that all CDC paths are exercised
- Ensure that all protocol corner cases are stressed
- Enable regression test development for full coverage

Unified Coverage Data Base

Step 3. Reconvergence Verification



Metastability Injection Assertions



Metastability injection assertions randomly modify the delays through synchronizers (+1 or -1 cycle) when metastability conditions are present in silicon

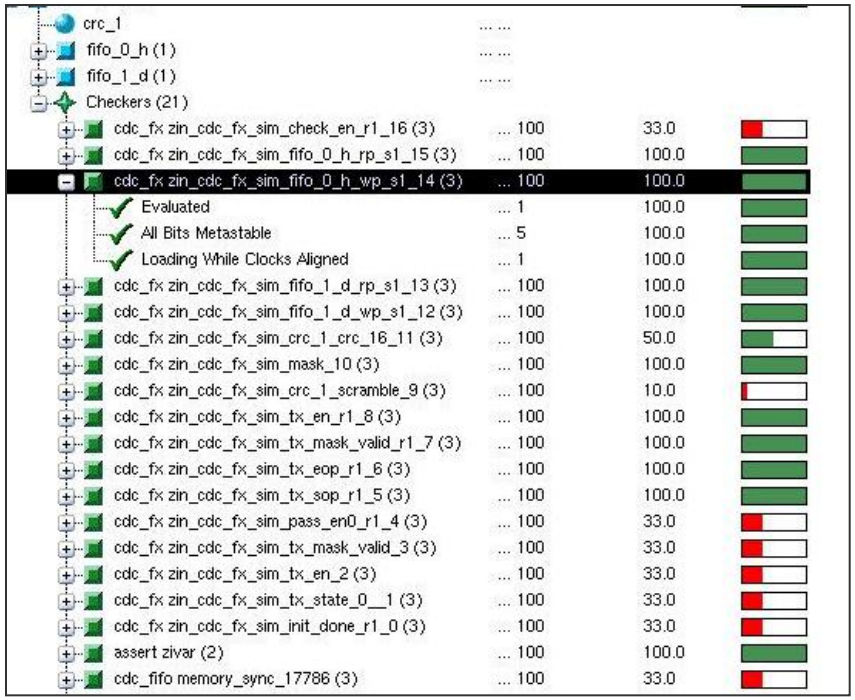
Questa CDC Automatically Adds Metastability Injection Assertions

- Only adds metastability injection assertions to those synchronizers that might cause reconvergence issues
- Random delay (+1 or -1 cycle) is only inserted when metastability is possible
- Metastability injection assertions automatically collect coverage data

Automatic metastability injection in simulation is the only way to effectively verify your CDC reconvergence issues

Metrics for Metastability Injection Assertions

Using assertions for metastability injection enables coverage metrics to be collected



crc_1	... ...	
fifo_0_h (1)	... ...	
fifo_1_d (1)	... ...	
Checkers (21)		
cdc_fx zin_cdc_fx_sim_check_en_r1_16 (3)	... 100	33.0
cdc_fx zin_cdc_fx_sim_fifo_0_h_rp_s1_15 (3)	... 100	100.0
cdc_fx zin_cdc_fx_sim_fifo_0_h_wp_s1_14 (3)	... 100	100.0
Evaluated	... 1	100.0
All Bits Metastable	... 5	100.0
Loading While Clocks Aligned	... 1	100.0
cdc_fx zin_cdc_fx_sim_fifo_1_d_rp_s1_13 (3)	... 100	100.0
cdc_fx zin_cdc_fx_sim_fifo_1_d_wp_s1_12 (3)	... 100	100.0
cdc_fx zin_cdc_fx_sim_crc_1_crc_16_11 (3)	... 100	50.0
cdc_fx zin_cdc_fx_sim_mask_10 (3)	... 100	100.0
cdc_fx zin_cdc_fx_sim_crc_1_scramble_9 (3)	... 100	10.0
cdc_fx zin_cdc_fx_sim_tx_en_r1_8 (3)	... 100	100.0
cdc_fx zin_cdc_fx_sim_tx_mask_valid_r1_7 (3)	... 100	100.0
cdc_fx zin_cdc_fx_sim_tx_eop_r1_6 (3)	... 100	100.0
cdc_fx zin_cdc_fx_sim_tx_sop_r1_5 (3)	... 100	100.0
cdc_fx zin_cdc_fx_sim_pass_en0_r1_4 (3)	... 100	33.0
cdc_fx zin_cdc_fx_sim_tx_mask_valid_3 (3)	... 100	33.0
cdc_fx zin_cdc_fx_sim_tx_en_2 (3)	... 100	33.0
cdc_fx zin_cdc_fx_sim_tx_state_0__1 (3)	... 100	33.0
cdc_fx zin_cdc_fx_sim_init_done_r1_0 (3)	... 100	33.0
assert zivar (2)	... 100	100.0
cdc_fifo memory_sync_17786 (3)	... 100	33.0

Coverage metrics:

- Ensure that all CDC paths are exercised
- Ensure that all protocol corner cases are stressed
- Enable regression test development for full coverage

Questa CDC Verification Delivers

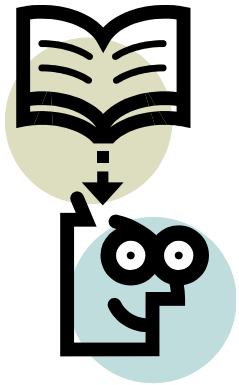
- Structural CDC analysis
  - Automatically recognizes a large set of synchronizers
  - Comprehensive modal analysis
- Protocol verification
  - Automatic generation of CDC protocol assertions
  - These can either be proven with formal analysis or verified through simulation
- Reconvergence verification
  - Complete structural analysis to identify potential reconvergence issues
  - Automatic metastability injection in simulation to verify the design correctly handles reconvergence

Agenda

- Verification Overview
  - Assertion and Functional Coverage
  - Constrained Random
  - Requirements Tracing
  - Algorithmic TB (InFact)
  - Questa CDC
  - Questa Formal
  - Questa Codelink
  - Questa VIP

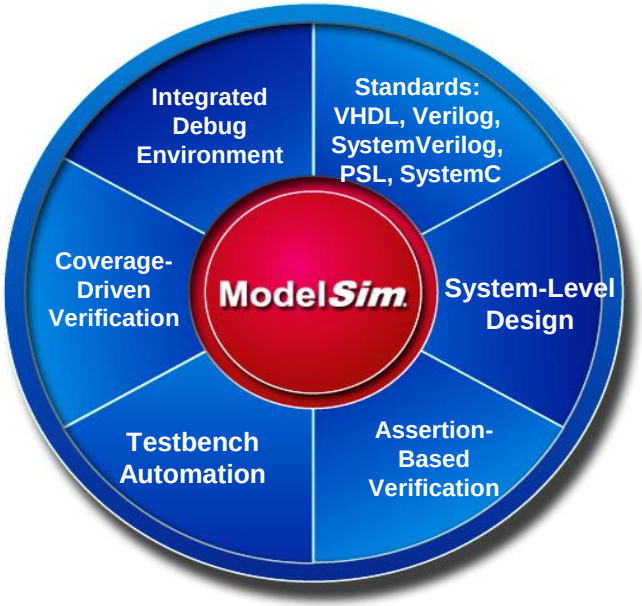
Manually Linking Software with Simulation

- Must convert software binary to memory image file
- No source-level debug
- Slow, ~ 1 to 10 instructions/sec



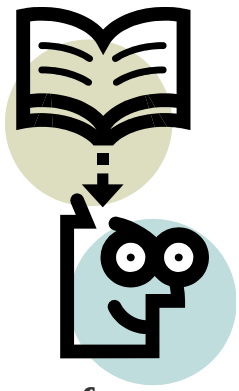
software Developer

Verification Engineer



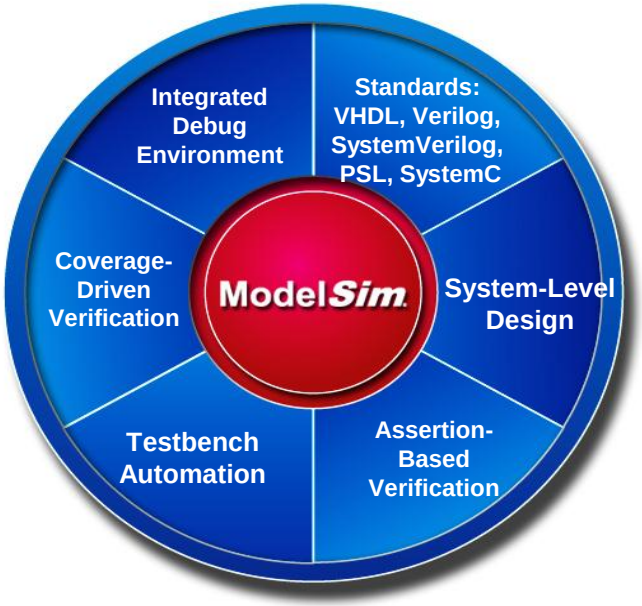
Codelink Automates Software Testbenches

- Loads software binary
- Full source-level debug
- Significantly faster than full-functional model



software Developer

Codelink

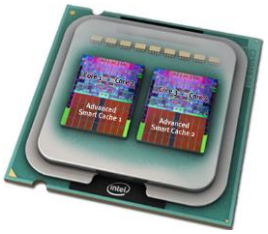


Agenda

- Verification Overview
  - Assertion and Functional Coverage
  - Constrained Random
  - Requirements Tracing
  - Algorithmic TB (InFact)
  - Questa CDC
  - Questa Formal
  - Questa VIP

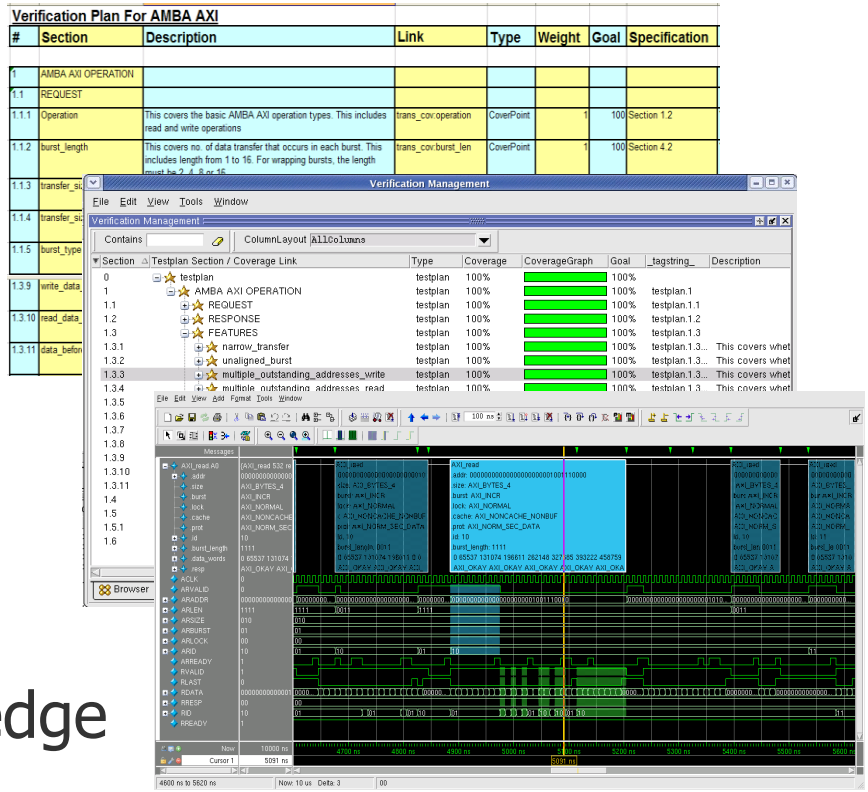
What is Verification IP?

- SoC Design Trends
  - More platform based designs
  - Increasing reuse of Design IP
  - Built around standard interfaces
- Complex verification environment
  - reuse a necessity
- Verification IP
  - Re-usable testbench building blocks
  - Compliant with standard interfaces and protocols
  - Built using standard languages and methodologies



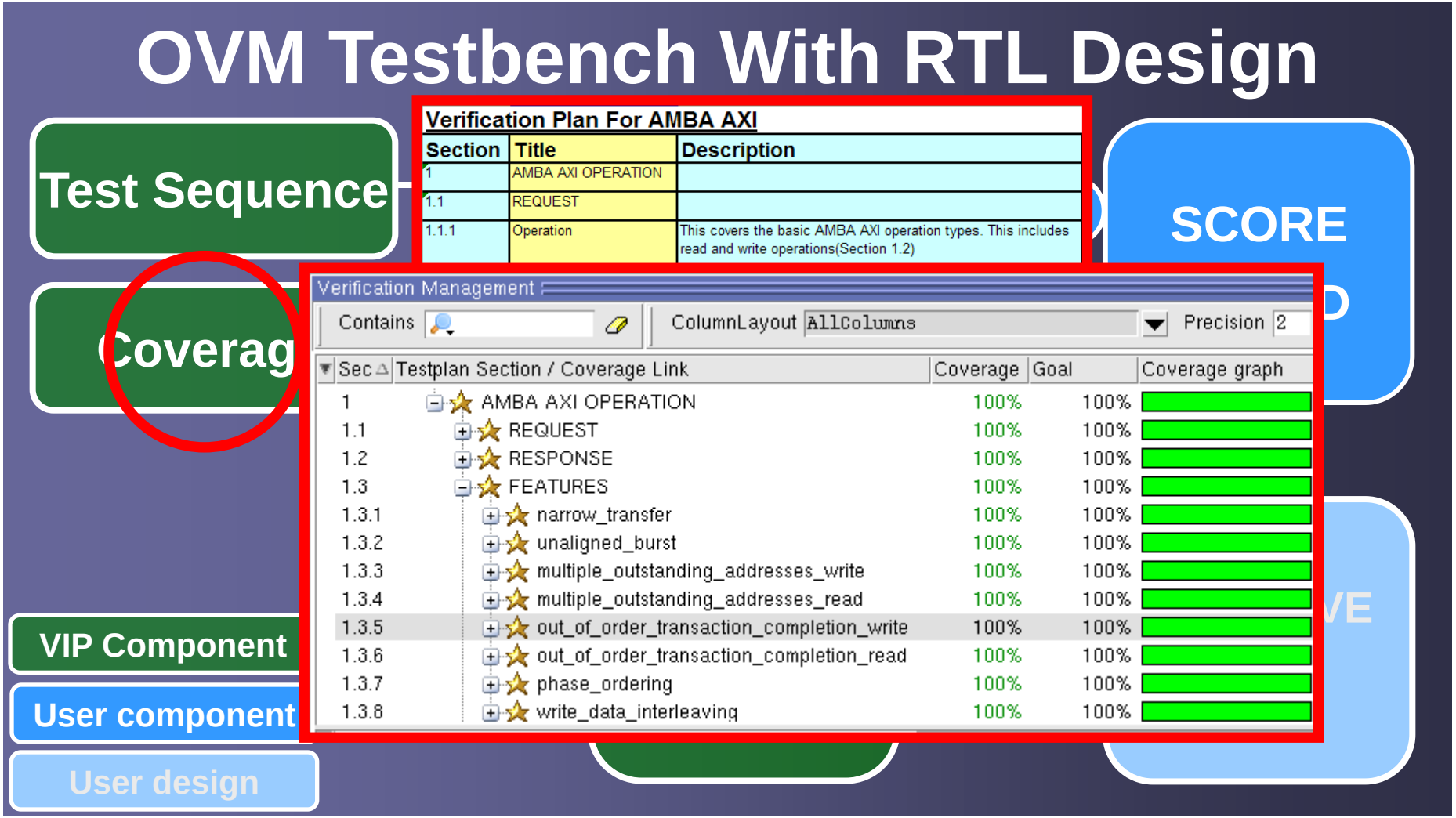
Questa Verification IP

- Comprehensive verification IP
  - Complete protocol coverage and checking
  - Test suite with compliance tests
- Leverage latest verification techniques
  - Verification planning, constrained random, functional coverage
  - Developed for OVM and UVM
- Supports popular and leading edge SoC standards
  - PCIe, USB, AMBA, ethernet, …



Questa Verification IP Features

Complete protocol test sequences and coverage



Questa Verification IP Example

Coverage results for SPI Controller

- Questa Verification IP provides protocol Test Plan, Test Sequences and Coverage
- User adds DUT specific Test Plan, Test Sequences and Coverage
- Questa Verification Management combines the results

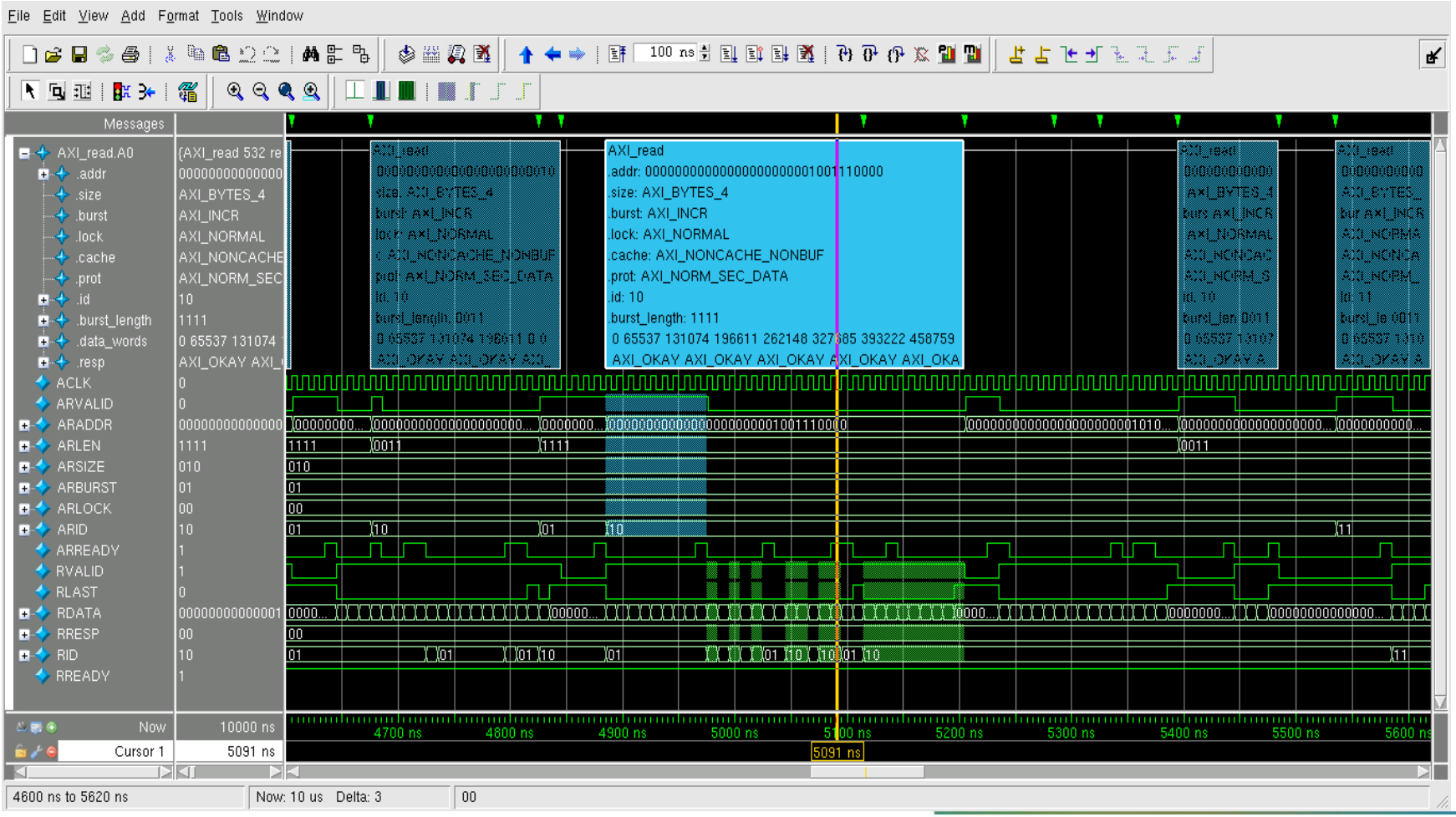
Verification Management Tracker

Sec#	Testplan Section / Coverage Link	Description	Coverage	Coverage graph	% of Goal	Weight	Goal	Unlinked
0	testplan		50.26%		50.26%	1	100%	No
1	SPI Interface		60.15%		60.15%	1	100%	No
2	APB3 Interface		40.65%		40.65%	1	100%	No
2.1	General	General APB3 op...	70%		70.00%	1	100%	No
2.1.1	AMBA 3 APB	AMBA 3 APB	70%		70.00%	1	100%	No
2.1.1.1	Operation	This covers the b...	100%		100.00%	1	100%	No
2.1.1.2	Slave ID	This covers cycle...	100%		100.00%	1	100%	No
2.1.1.3	Wait count	This covers the n...	20%		20.00%	1	100%	No
2.1.1.4	Cycle with slave error	This covers the er...	50%		50.00%	1	100%	No
2.1.1.5	Cycle with slave ID	This covers the sl...	100%		100.00%	1	100%	No
2.1.1.6	Cycle with wait count	This covers the n...	20%		20.00%	1	100%	No
2.1.1.7	Back to back cycles	This covers the n...	100%		100.00%	1	100%	No
2.2	Registers	This covers that w...	78.57%		78.57%	1	100%	No
2.3	Automatic Slave Select	Covers the autom...	50%		50.00%	1	100%	No
2.4	Character Length	This covers the dif...	3.12%		3.12%	1	100%	No
2.5	Divider	Covers the differe...	1.56%		1.56%	1	100%	No
3	Interrupt		50%		50.00%	1	100%	No

Questa Verification IP Features

Protocol stack debug

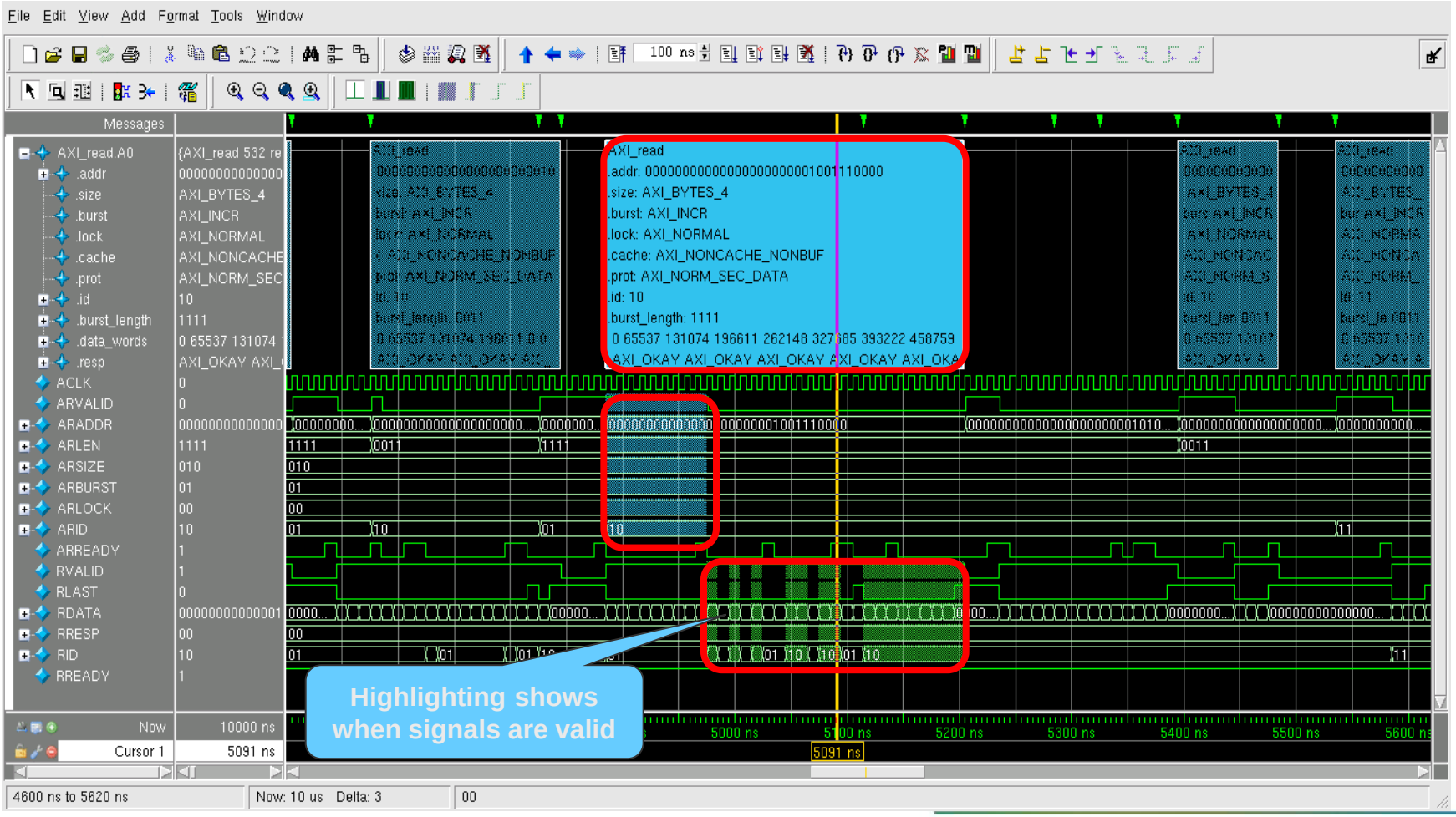
- Quickly understand and analyze bus activity



Questa Verification IP Features

Protocol stack debug

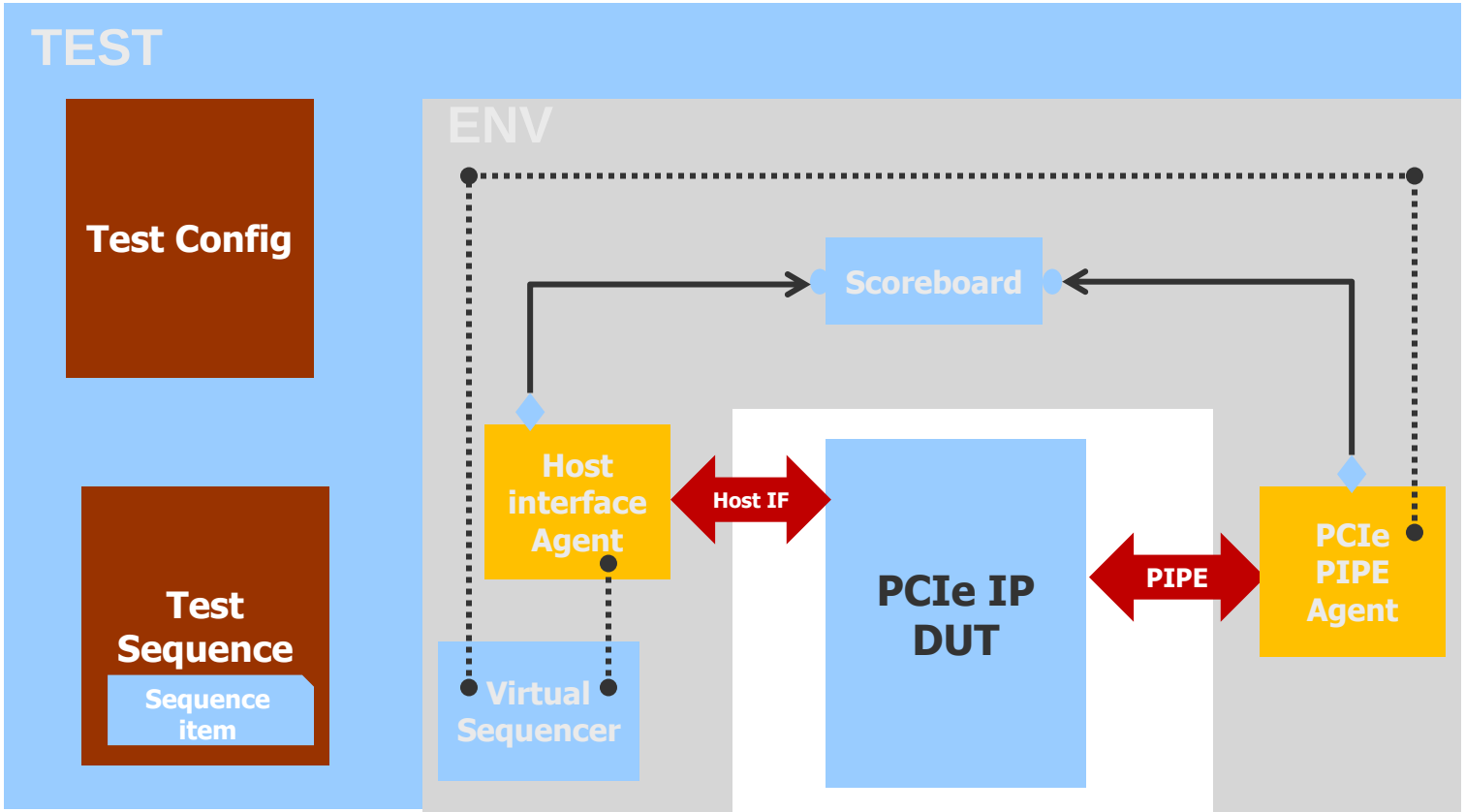
- Highlighting links transaction and signal activity



Questa Verification IP Features

Full support for UVM and OVM

- Connect and reuse standard UVM or OVM components
  - Agents, TLM ports, sequences, sequence items, config

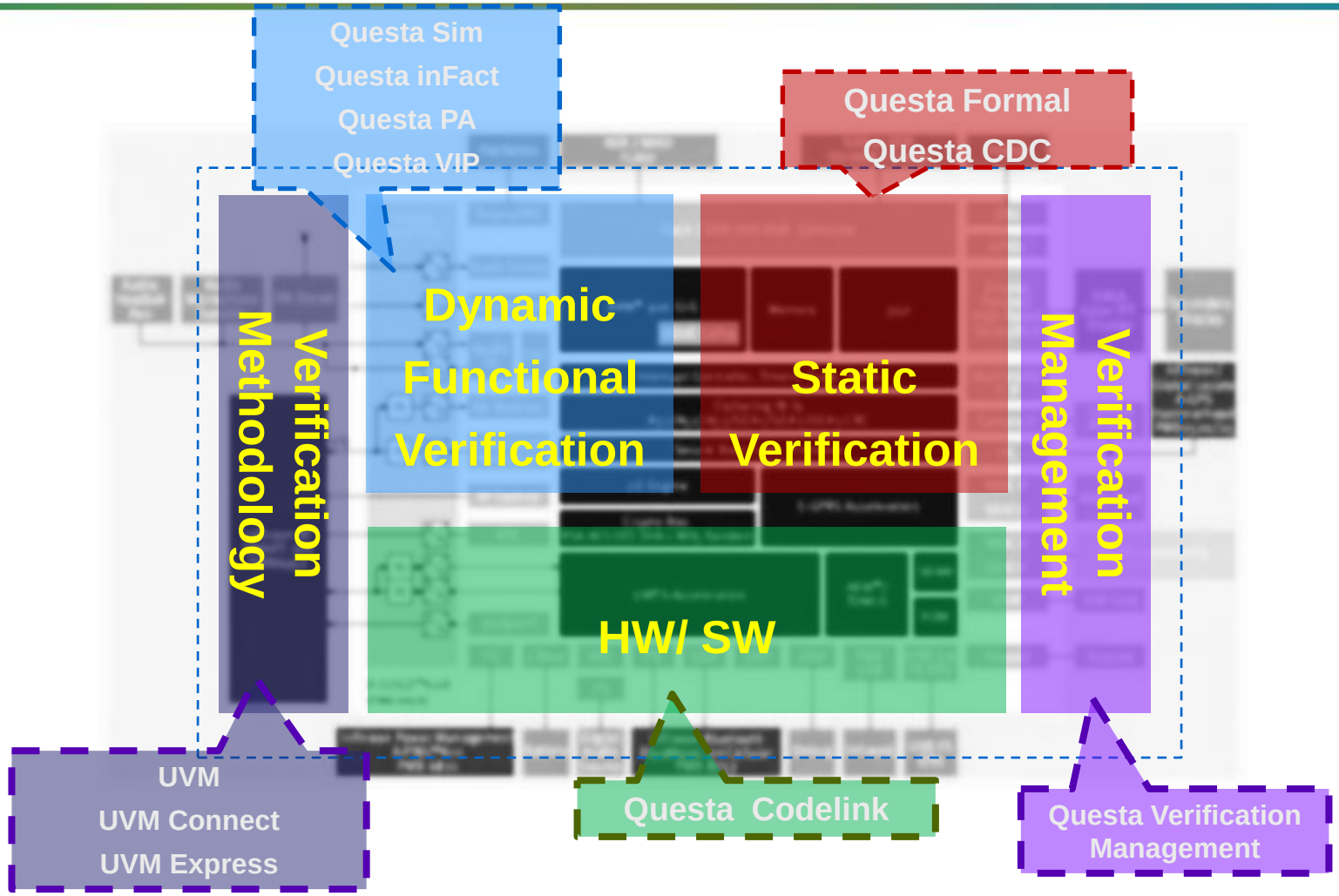


Protocol support

- Supports popular and leading edge SoC standards
  - AMBA
    - APB3, AHB, AXI, AXI4, AXI4-lite, AXI4-stream, AXI-LP
  - PCI Express
    - 1.1, 2.0, 3.0
  - USB
    - 2.0, 3.0, OTG, UTMI, PIPE
  - Ethernet
    - 10/100, 1G, 10G, 40G, 100G
  - SPI 4.2
  - DDR2, DDR3
  - OCP 2.2
  - HDMI
  - I2C, I2S
  - SPI, UART

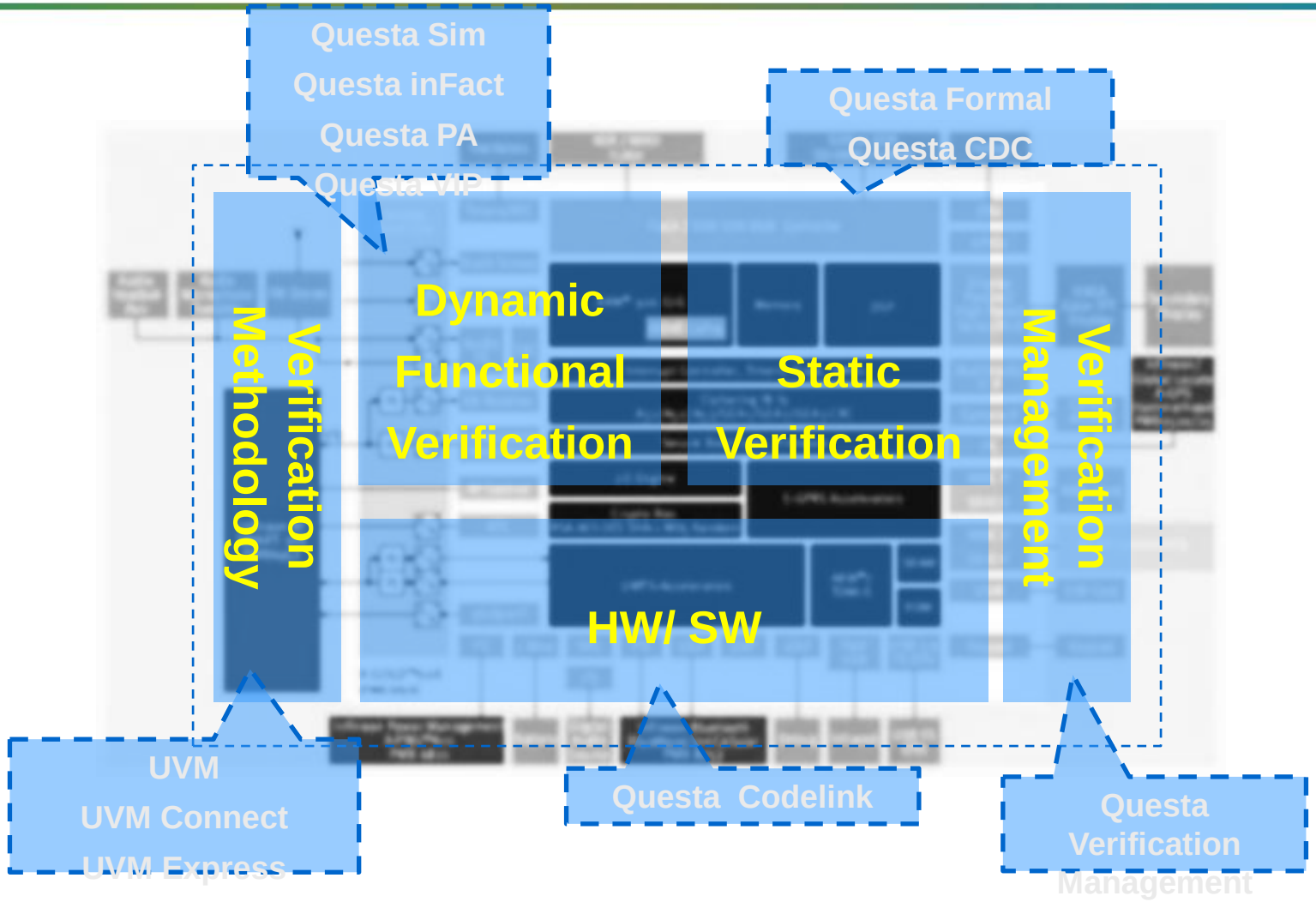
Questa Verification Platform

Best In Class Engines



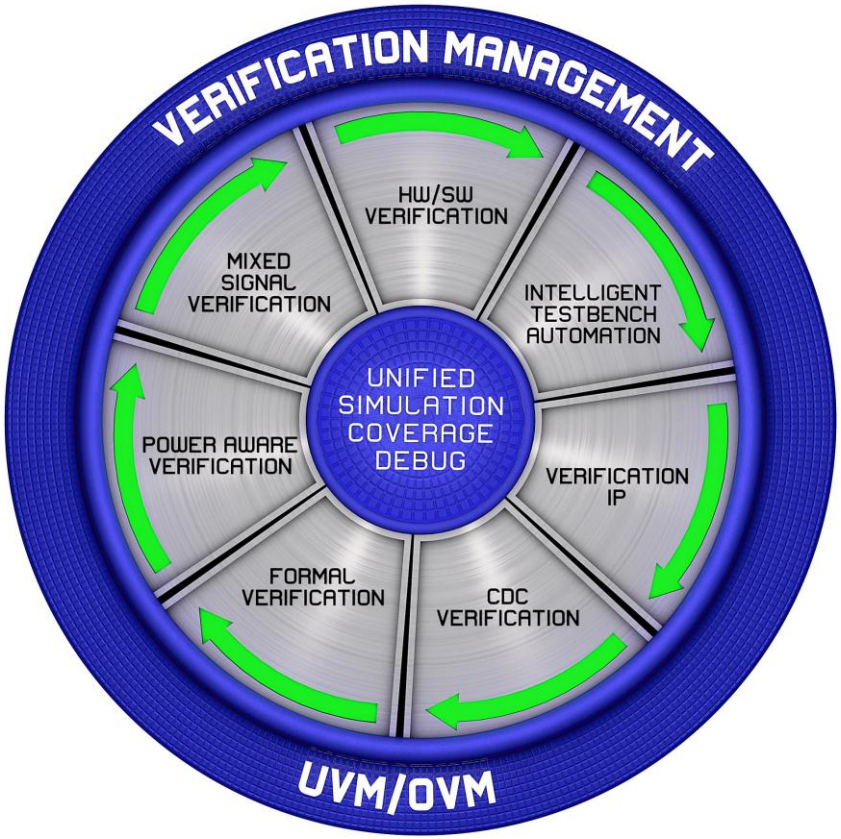
Questa Verification Platform

Best In Class Engines - Unified Front End Analysis & Compile



Questa Verification Platform

- Comprehensive integrated SOC verification platform
  - Best in class engines
  - Integrated
  - Comprehensive debug analysis
- Industry Leading SOC Verification Solutions
  - Coverage Closure Solution
  - Low Power Verification Solution
  - Software Driven Verification Solution
- Standards Leadership
  - Driving the evolution of IEEE standards
  - Major donations to Accellera UVM
  - Accellera UCIS from Mentor UCDB



Agenda

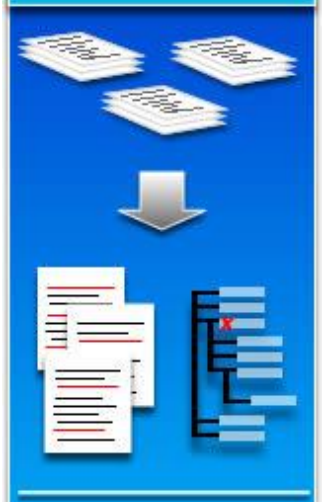
- Verification Overview
  - Assertion and Functional Coverage
  - Constrained Random
  - Requirements Tracing
  - Algorithmic TB (InFact)
  - Questa CDC
  - Questa Formal
  - Questa VIP
- Rule Checking
- Precision

A RTL Reuse Method

Previous Design

- Unknown Quality
- Unfamiliar Design

Step 1 Design Integrity

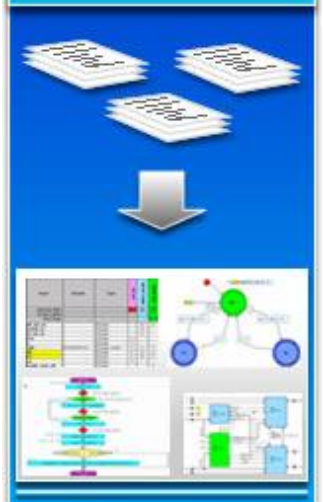


Automate Analysis of RTL Integrity

Step 2 Quality Assessment

Assess RTL to Design Standards

Step 3 Design Visualization



Visualize Behavior and Structure

Ready to Reuse

- Quality Quantified
- Fully Understood

Share Knowledge

Do I Have all the Files?

Design Files	Design Support Files	Tool Output Files
▪ Verilog ▪ VHDL ▪ C & C++ ▪ Memory Content ▪ Constraint Files	▪ Tool Scripts ▪ Design Documents	▪ Netlists ▪ Reports ▪ Projects

- Design Files: describe the design to the tools in a flow

- Design Support Files: help you understand & automate aspects of the design flow

- Tool Output Files: tool-generated files for use as input or that provide information

Missing Files are Found

Design Manager - Project tinybus_completion

File Edit View HDL Tasks Tools Options Window Help

Main

New / Add

Check

Simulate

Synthesize

Document & Visualize

Explore

Tasks

Viewpoints

Design Explorer [Using viewpoint : Default Viewpoint]

Design Unit	Type	Language
TINYBUS_COMPLETION_LIB...		
arbiter	Component	Verilog '95
fifo_8_tester	Component	Verilog '95
tb_interface	Component	Verilog '95
tb_interface_testb...	Component	Verilog '95
tb_interface_tester	Component	Verilog '95
tinybus_controller	Component	Verilog '95
Visualization		
TINYBUS_COMPLETI...		
Design Files		
TINYBUS_COMPLETION_...		
arbiter.v	Source File	2 KB
fifo_8_tester.v	Source File	2 KB
newtbi.v	Source File	7 KB
tb_interface_testben...	Source File	3 KB
tb_interface_tester.v	Source File	17 KB

Design Hierarchy	Design Unit Name	Name
tb_interface_testbench	tb_interface_testbench	tb_interface_testb
tb_interface_testbench	tb_interface_testbench	tb_interface_testb
arb	arbiter	arbiter
DUT	tb_interface	tb_interface
controller	tinybus_controller	tinybus_controller
data_in	tb_interface	data_in : fifo_8
data_out	tb_interface	data_out : fifo_8
tester	tb_interface_tester	tb_interface_teste

Launch Pad | Sub Browser | Tasks/Templates

Project | TINYBUS_COMPLETION_LIB1

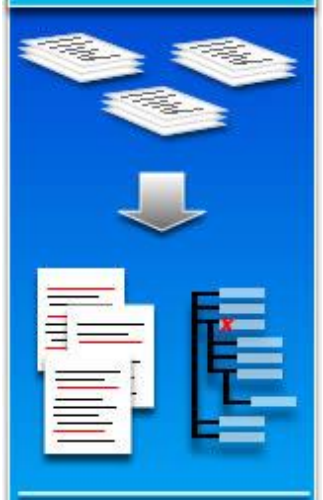
Missing Files are Highlighted in the Hierarchy

A RTL Reuse Method

Previous Design

- Unknown Quality
- Unfamiliar Design

Step 1 Design Integrity



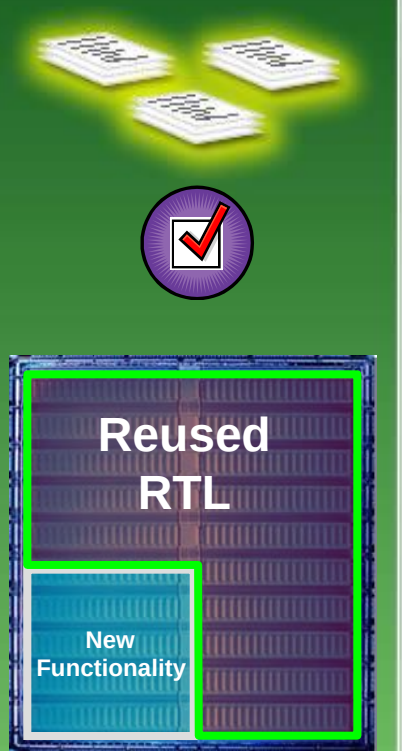
Automate Analysis of RTL Integrity

Step 2 Quality Assessment

Assess RTL to Design Standards

Ready to Reuse

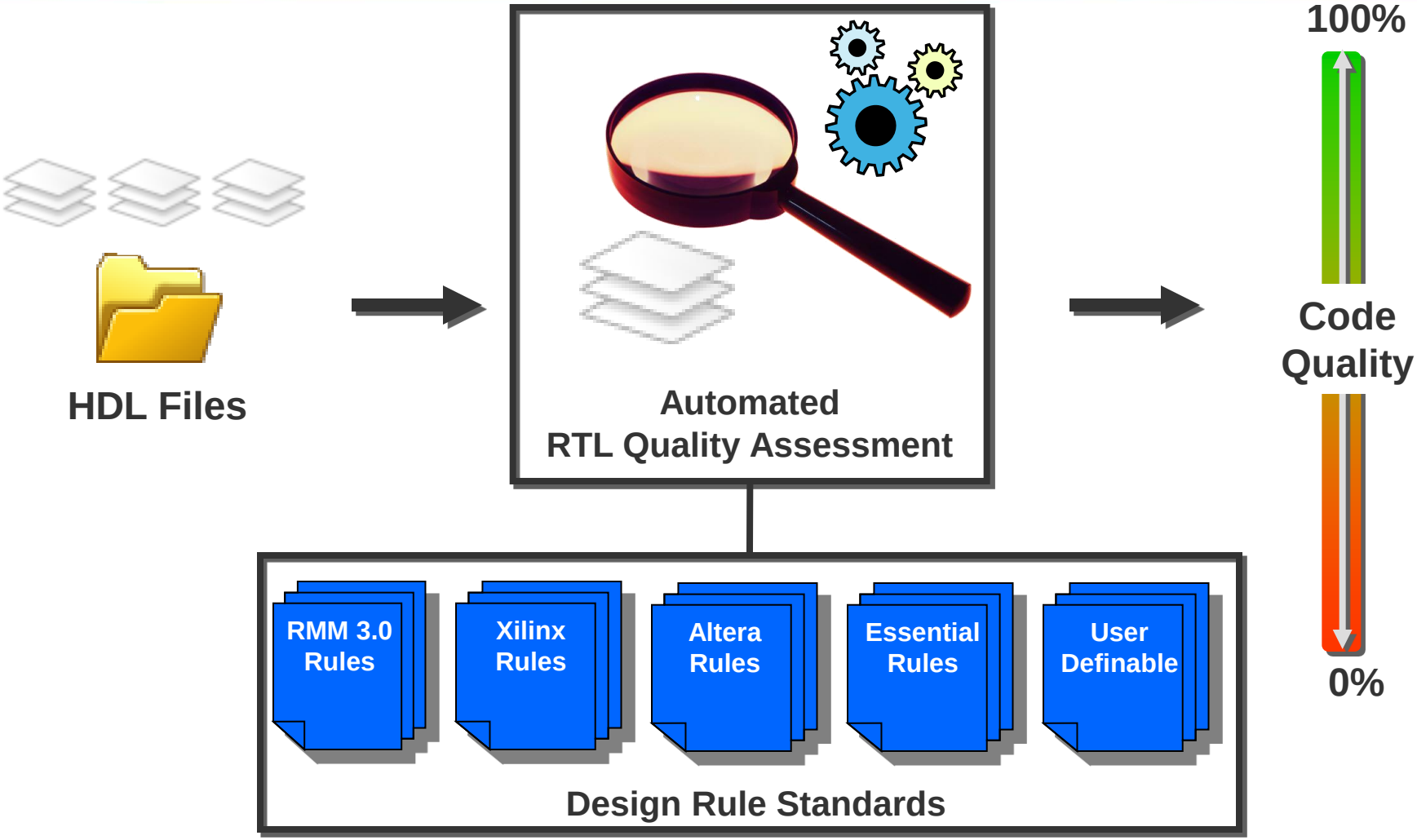
- Quality Quantified
- Fully Understood



Take Emotion Out of the Decision

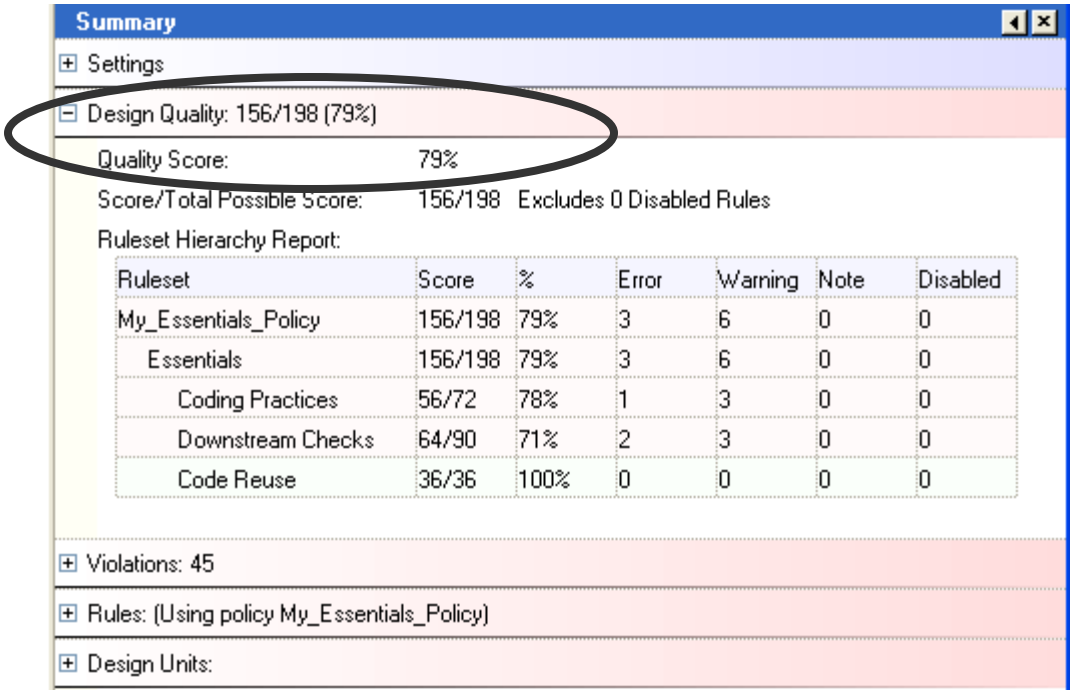
How do you tell people their baby is ugly?

Objective Standards are Key



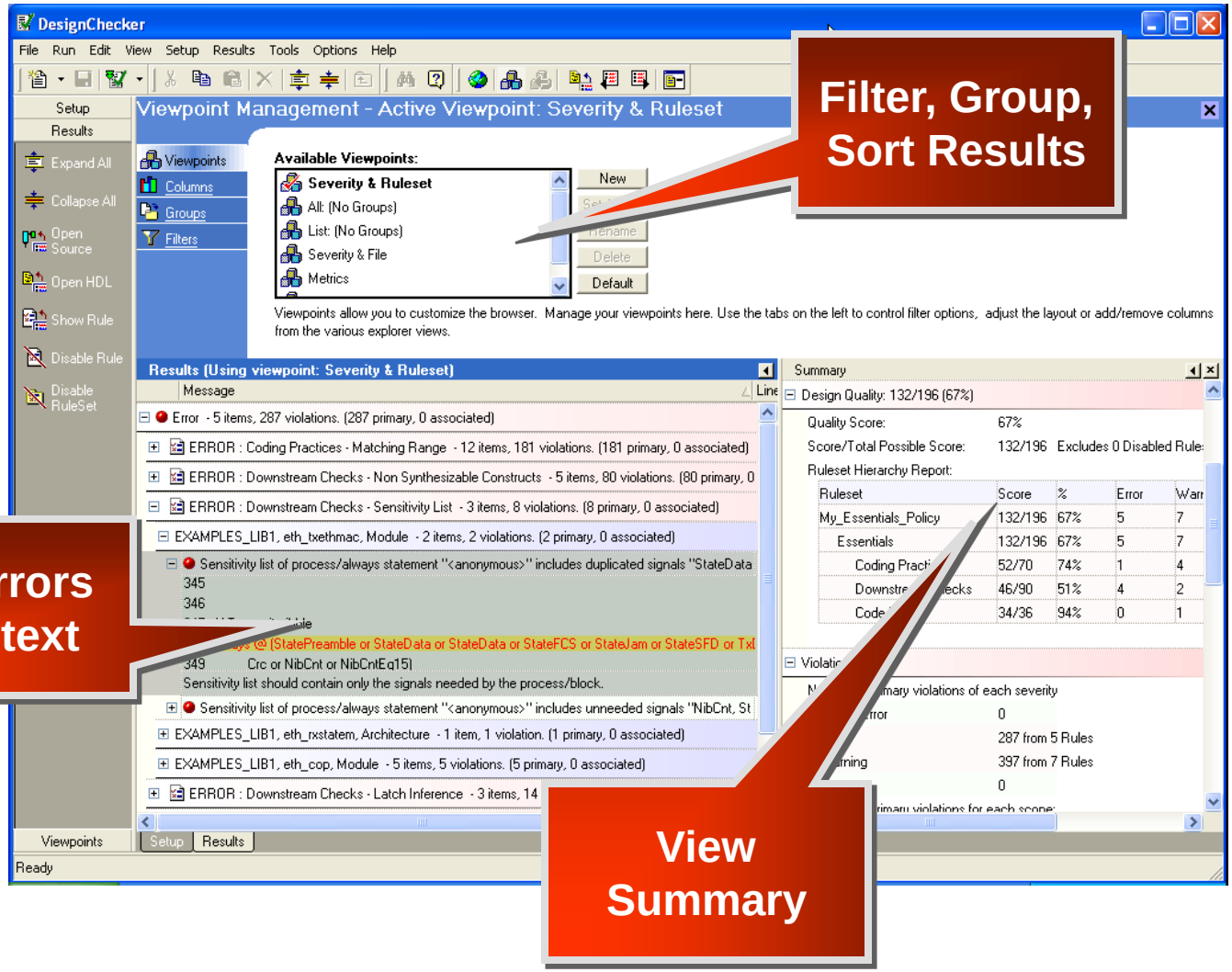
Code Scoring

- All rules have can have a score
- This allows you to see how “good” your code is within seconds
- You can set your own scoring system for your own rules
- Code quality results are automatically created for your checking run (it can optionally be turned off).

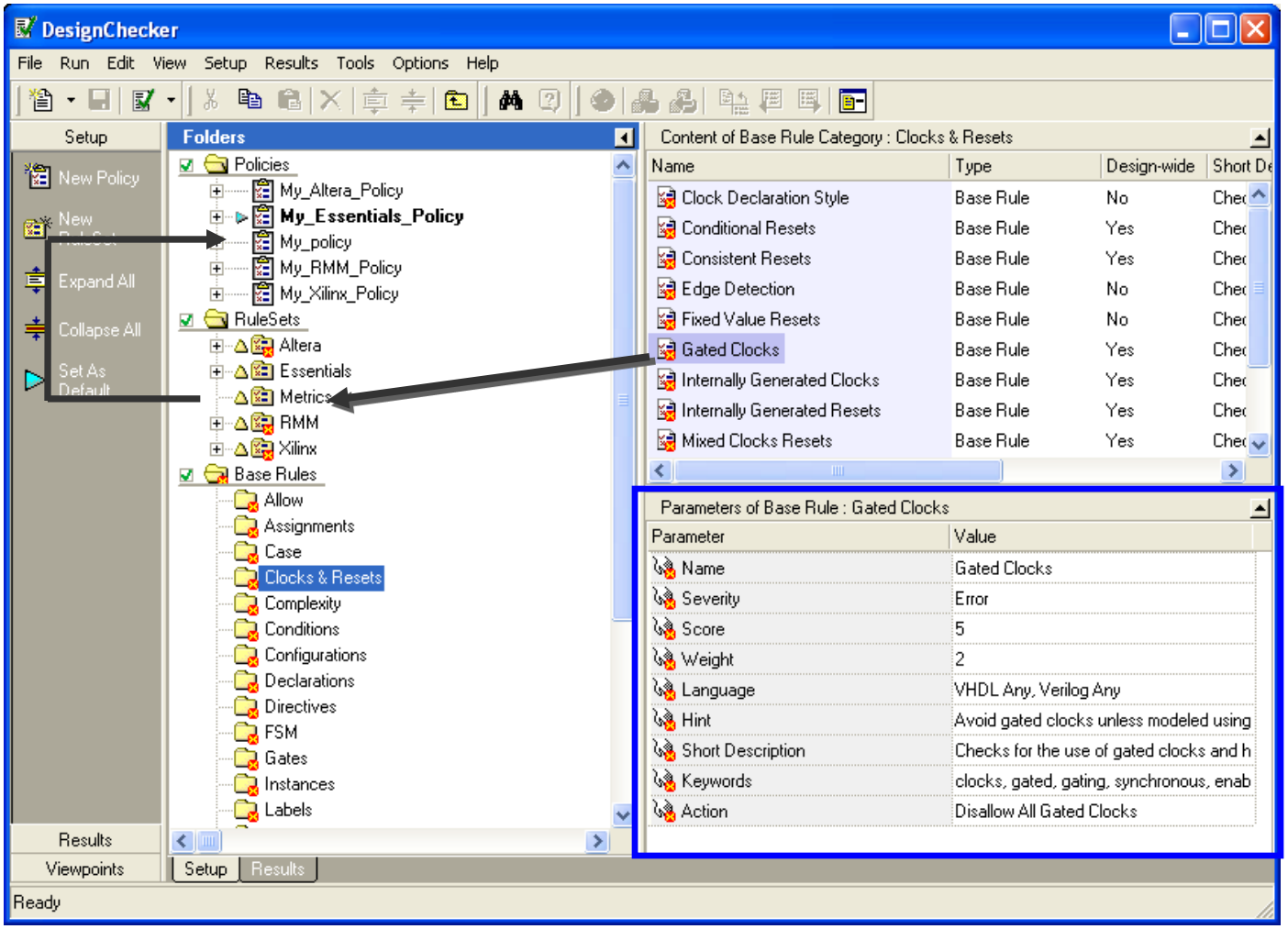


Ruleset	Score	%	Error	Warning	Note	Disabled
My_Essentials_Policy	156/198	79%	3	6	0	0
Essentials	156/198	79%	3	6	0	0
Coding Practices	56/72	78%	1	3	0	0
Downstream Checks	64/90	71%	2	3	0	0
Code Reuse	36/36	100%	0	0	0	0

Assess Reuse Effort



Customize Your Own Rules



The Process:

- Create ruleset(s)
- Drag & drop rulesets & rules into your own rulesets
- Change rule parameters
- Create policies that link to rulesets

Good Coding Practices

- No extra or unused signals
- Isolate or avoid gated clocks
- Avoid internally-generated resets
- Avoid mixed clock edges
- No multiply-driven signals
- Assign a value to a signal before reading it
- Matching comparison/assignment ranges

Ensure expected & good results

Downstream Checks

- No combinational feedback loops
- Avoid latches & inferred registers
- Register outputs
- Avoid or isolate gate-level logic
- Ensure naming compatibility with downstream tools
- Avoid delay times
- Avoid default initialization
- Establish a subset of allowed constructs
- Use complete sensitivity lists

Catch issues before running other tools

Document Quality

- Export Summary report as CSV, TSV, or HTML
- Export Result Table as CSV, TSV, or HTML
- Export rules used in ASCII format

	A	B	C
1	Design Root		
2	Library:	Ethernet	
3	Primary:	eth_fifo	
4	Secondary:	eth_fifo	
5	Master Clocks	clk	
6	Master Resets	rst	
7	Depth:	Single	
8	Violations: 23		
9	Number of violations for each scope:		
10	Errors	0	
11	Notes	0	
12	Warnings	23 from 7 Rule/s	
13	Number of violations of each severity		
		0	0%
		0	0%
		0	0%
		0	0%
		0	0%
		23	100%
		0	0%
		0	0%

Rule Severity	Severity, Ruleset and Rule	Library, Design Unit and Scope	
Warning	Warning : 2.1 - General Naming Conventions - 2.1.9 - Reset name prefix	Ethernet, eth_fifo, Module	Reset signal "reset" violates naming convention: use
Warning	Warning : 2.1 - General Naming Conventions - 2.1.11 - Descending bus order	Ethernet, eth_fifo, Module	Dimension definition "[0:DEPTH-1]", for "fifo", do
Warning	Warning : 3 - Portability - 3.2.1 - Avoid hard-coded numeric values	Ethernet, eth_fifo, Module	Avoid using hard coded numeric values such as "[0
Warning	Warning : 3 - Portability - 3.2.1 - Avoid hard-coded numeric values	Ethernet, eth_fifo, Module	

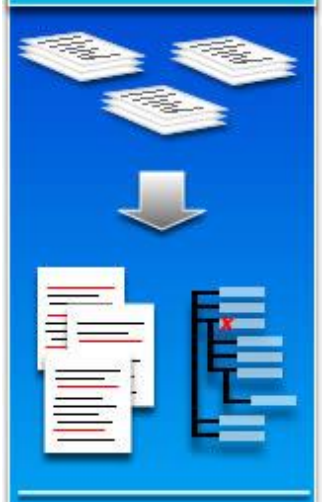
RuleSet: 4 - Clocks & Resets/4.6 - Internally Generated Resets
Rule Name: 4.6.2 - Isolate Condition resets
Base Rule: Conditional Resets
Name: 4.6.2 - Isolate Condition resets
Severity: Warning
Language: VHDL Any, Verilog Any
Hint: If conditional resets are used, isolate into a separate module
Short Description: If conditional resets are used, isolate into a separate module

A RTL Reuse Method

Previous Design

- Unknown Quality
- Unfamiliar Design

Step 1 Design Integrity

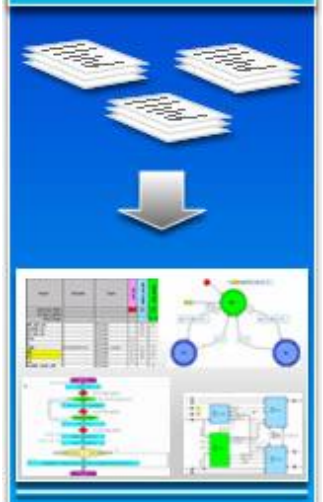


Automate Analysis of RTL Integrity

Step 2 Quality Assessment

Assess RTL to Design Standards

Step 3 Design Visualization



Visualize Behavior and Structure

Ready to Reuse

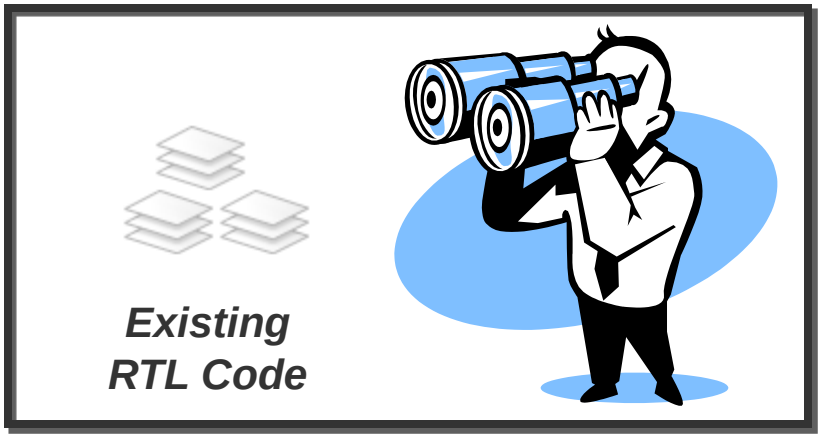
- Quality Quantified
- Fully Understood

Share Knowledge

Design Visualization Challenges
Learning & Integrating Legacy Designs

- Designs are easier to *write* in Text, but harder to *read*
  - Design structure is hard to "see" in text
  - Design functionality is hard to trace across text files
  - Interfaces are often poorly documented

- Re-Drawing by Hand is Inefficient
  - Very Time Consuming
  - Error prone
  - Hard to maintain



Delve Deeper with DesignPad

0: C:/MentorGraphics/HDS2007.1/examples/tutorial_ref/Sequencer_vlg/hdl/fibgen_struct.v (read-only)

File Edit Search View Graphics Document Tasks Options Window Help

0: fibgen_struct.v

Find Block:

Code Browser

Parser Level: Full

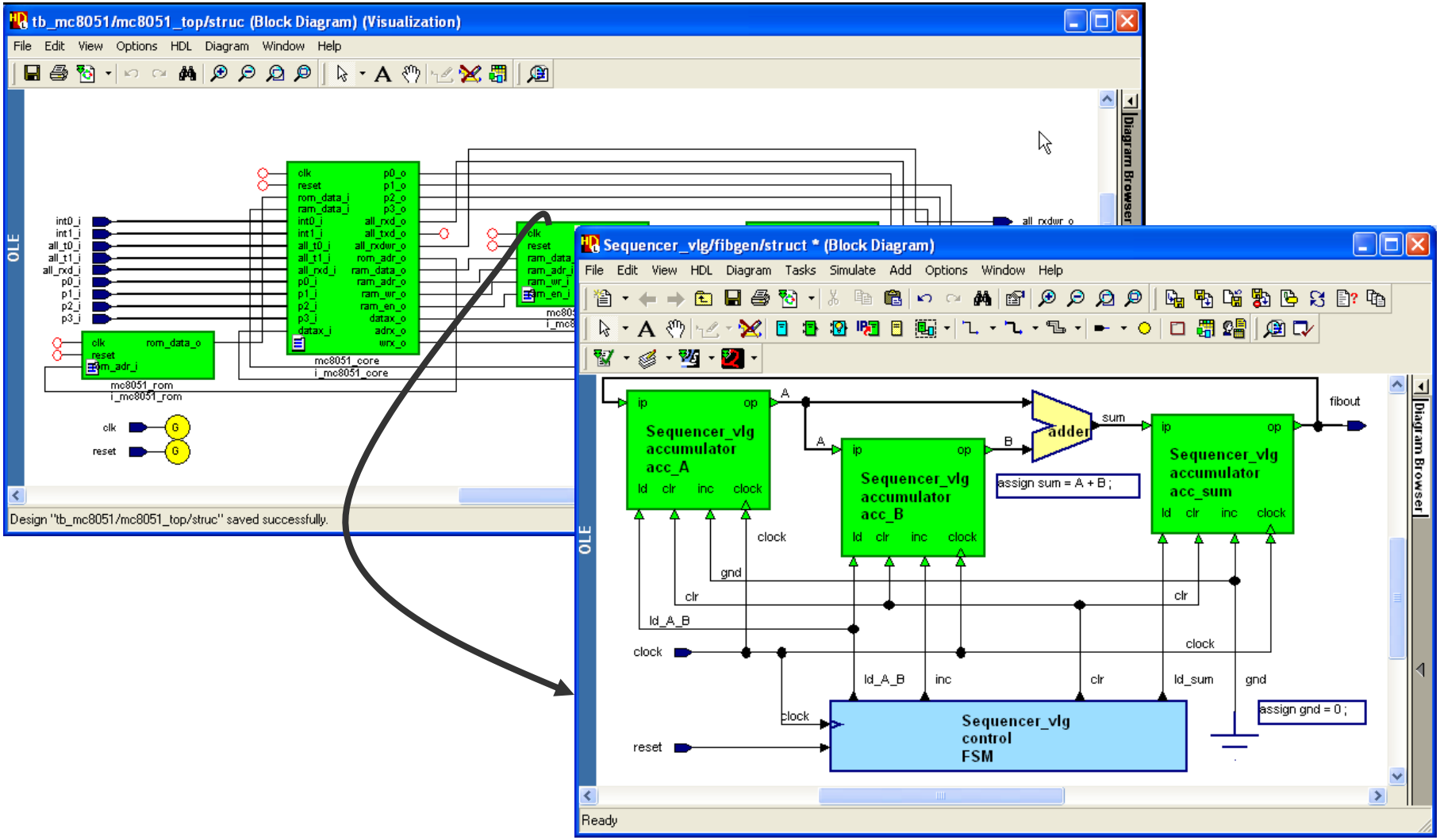
- Top Of Text
- Bottom Of Text
- fibgen
  - ports
  - declarations
    - clock
    - reset
    - fibout
    - A
    - B
    - clr
    - gnd
    - inc
    - ld_A_B
    - ld_sum
    - sum
    - tmp
    - <anonymous>
      - begin-end
  - instances
    - acc_A
    - acc_B
    - acc_sum
    - FSM

```
63
64  accumulator acc_sum(
65     .clock (clock),
66     .clr   (clr),
67     .inc   (gnd),
68     .ip    (sum),
69     .ld    (ld_sum),
70     .op    (fibout)
71  );
72
73  control FSM(
74     .clock  (clock),
75     .reset  (reset),
76     .tmp    (tmp),
77     .clr    (clr),
78     .inc    (inc),
79     .ld_A_B (ld_A_B),
80     .ld_sum (ld_sum)
81  );
82
83  // HDL Embedded Text Block 1 adder
84  assign sum = A + B ;
85  // HDL Embedded Text Block 2 ground
86  assign gnd = 0 ;
87  // HDL Embedded Text Block 3 eb1
88  // eb1 3
89  always @ (posedge clock) begin
90     // Asynchronous Reset
91     if(clr) begin
92        // Reset Actions
93        tmp = 0;
94        end
95     else begin
96
97        // Block 1
98           tmp = tmp+1;
99     end
100 end
101
102 endmodule // fibgen
103
104
```

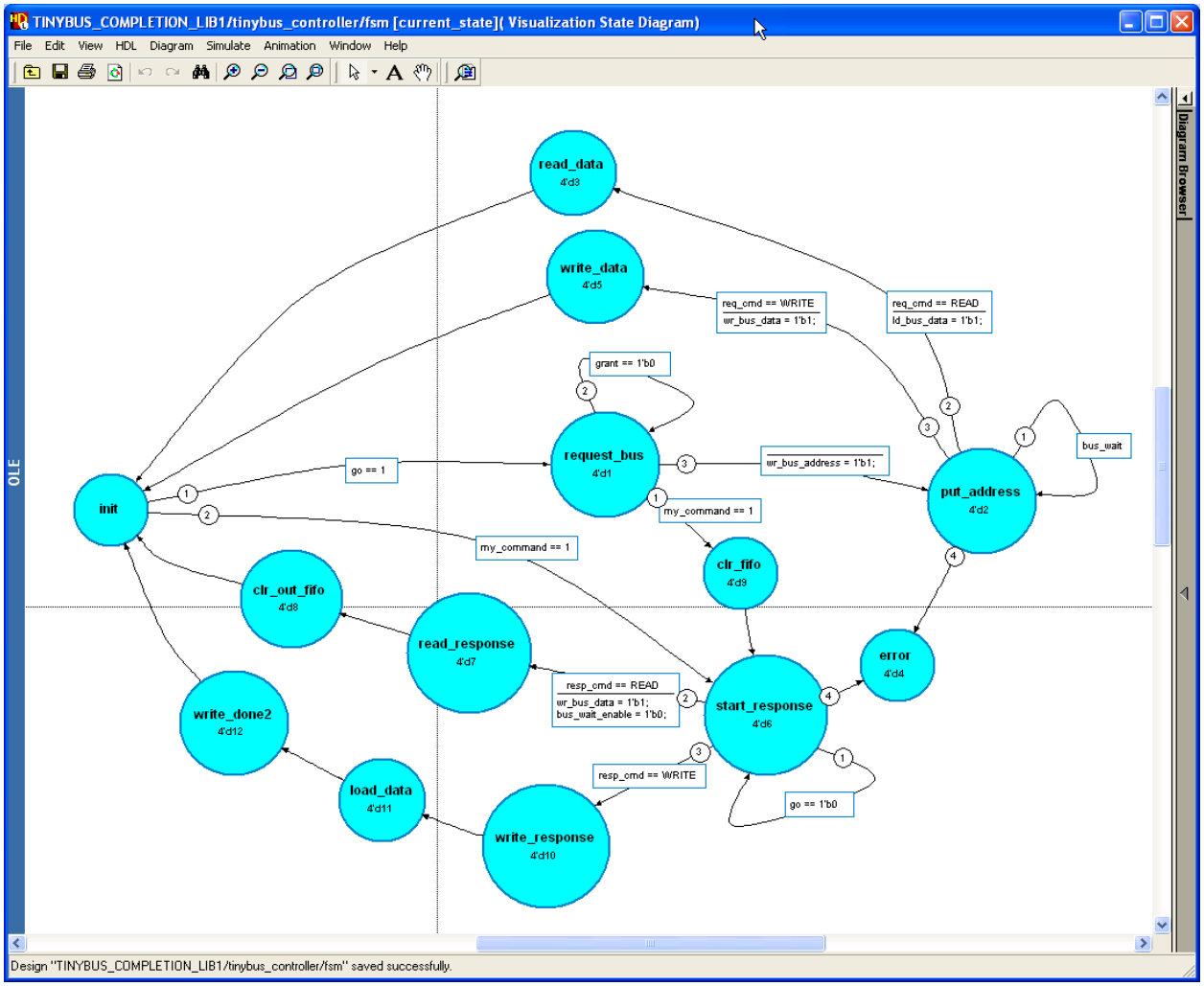
File : fibgen_struct.v | RO | Ins | B | N | Line: 89 / 104 | Col: 0

- Code Browser takes you inside design units
  - Navigate ports
  - Navigate declarations
- View as graphics
- Collapse, open up/down, & split windows to aid navigation
- “Diff” similar files

Structural Levels are Block Diagrams



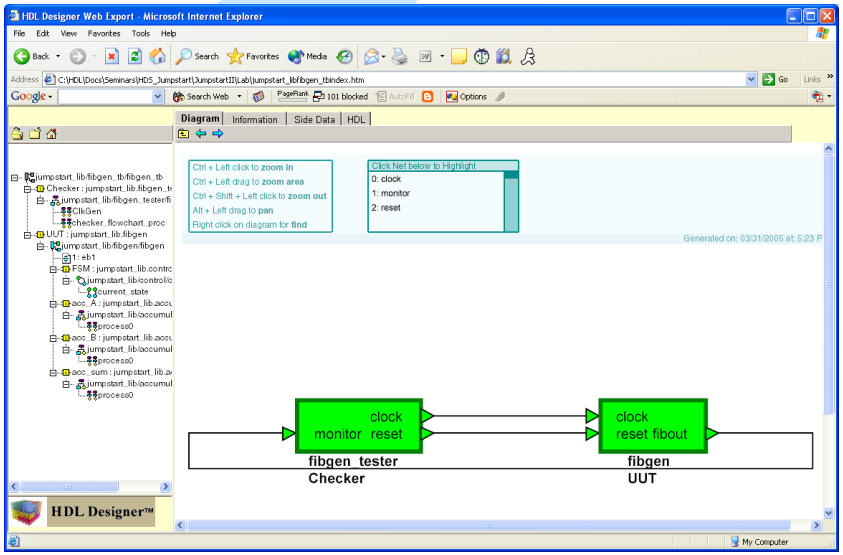
State Machines are Bubbles & Arcs



HTML Export

Automatically create an interactive website with HTML Export:

- Complete control of content
- Navigation matches HDL Designer
- View results in an HTML browser
- Snapshot a project any time
- No access to design database



Great for Design Reviews Too

The Results

HDL Designer Web Export - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address C:\HDL\Docs\Seminars\HDS_Jumpstart\JumpstartII\Lab\jumpstart_libfibgen_tbindex.htm

Diagram | Information | Side Data | HDL

- jumpstart_lib/fibgen_tb/fibgen_tb
  - Checker : jumpstart_lib.fibgen_tester
    - jumpstart_lib/fibgen_tester/fibger
      - ClkGen
      - checker_flowchart_proc
  - UUT : jumpstart_lib.fibgen
    - jumpstart_lib/fibgen/fibgen
      - 1: eb1
      - FSM : jumpstart_lib.control
        - jumpstart_lib/control/contr
          - current_state
      - acc_A : jumpstart_lib.accumul
        - jumpstart_lib/accumulator/
          - process0
      - acc_B : jumpstart_lib.accumul
        - jumpstart_lib/accumulator/
          - process0
      - acc_sum : jumpstart_lib.accun
        - jumpstart_lib/accumulator/
          - process0

```
-- pragma state_vector current_state

--------------------------------------------------------------------
-- Next State Block for machine machine0
--------------------------------------------------------------------
always  @(
   current_state)
begin : machine0_next_state_block_proc
   case (current_state)
      clr_regs:

            next_state = inc_accb;

      inc_accb:

            next_state = load_acc_sum;

      load_acc_sum:

            next_state = load_acc_A_B;

      load_acc_A_B:

            next_state = load_acc_sum;

      default: begin
         next_state = clr_regs;
      end
   endcase

end -- Next State Block

--------------------------------------------------------------------
-- Output Block for machine machine0
--------------------------------------------------------------------
always  @(
   current_state)
begin : machine0_output_block_proc
   -- Default Assignment
   clr = 0;
```

HDL Designer™

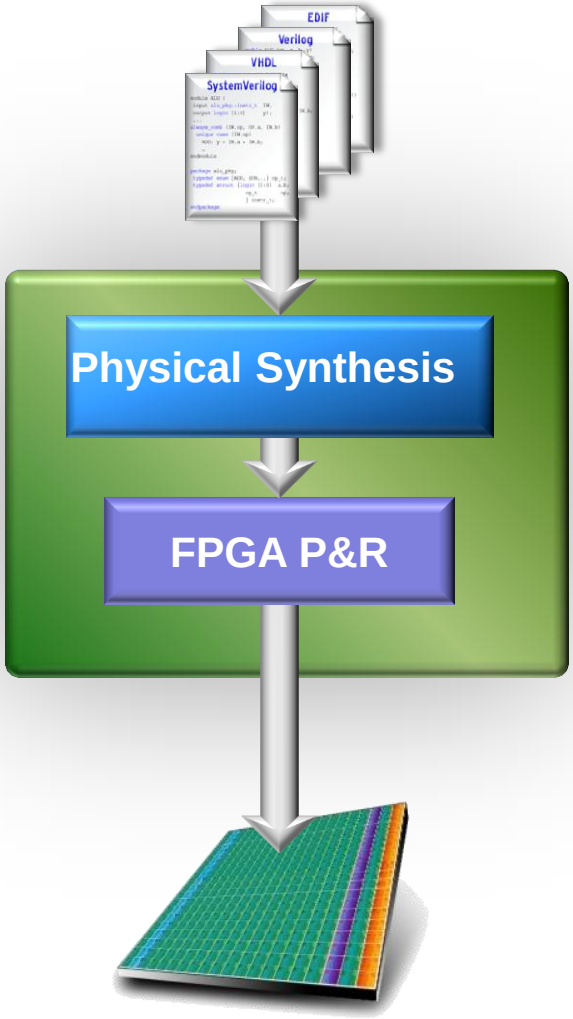
My Computer

Agenda

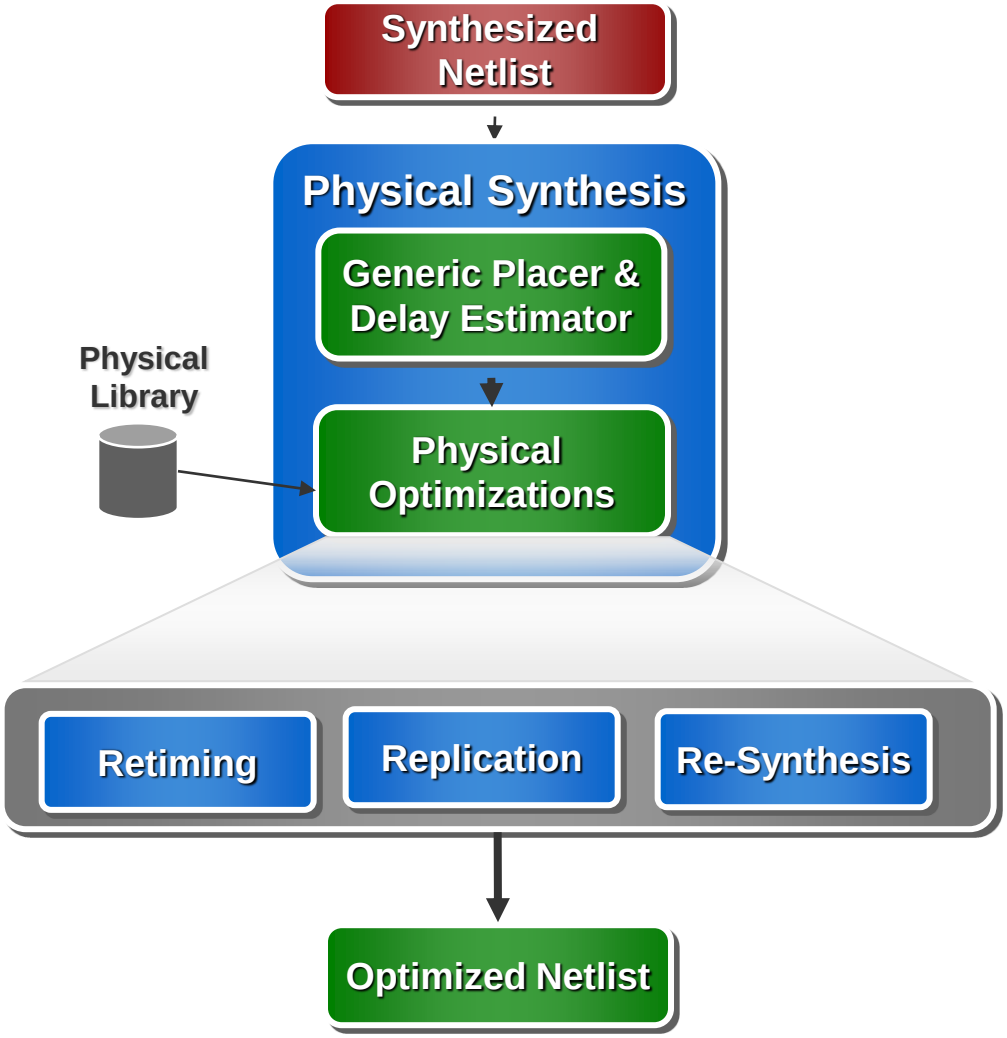
- Verification Overview
  - Assertion and Functional Coverage
  - Constrained Random
  - Requirements Tracing
  - Algorithmic TB (InFact)
  - Questa CDC
  - Questa Formal
- Rule Checking
- Precision

Precision 2009 - Synthesis Leadership

- Leading multi-vendor physical synthesis
  - Average 10 % F_{MAX} Improvement
  - 26 devices supported from all major vendors
- Fully automatic incremental synthesis
  - Up to 60% run-time savings
- Unique resource analysis/management improves QoR
- Industry leading mixed language support
  - Superior SystemVerilog coverage



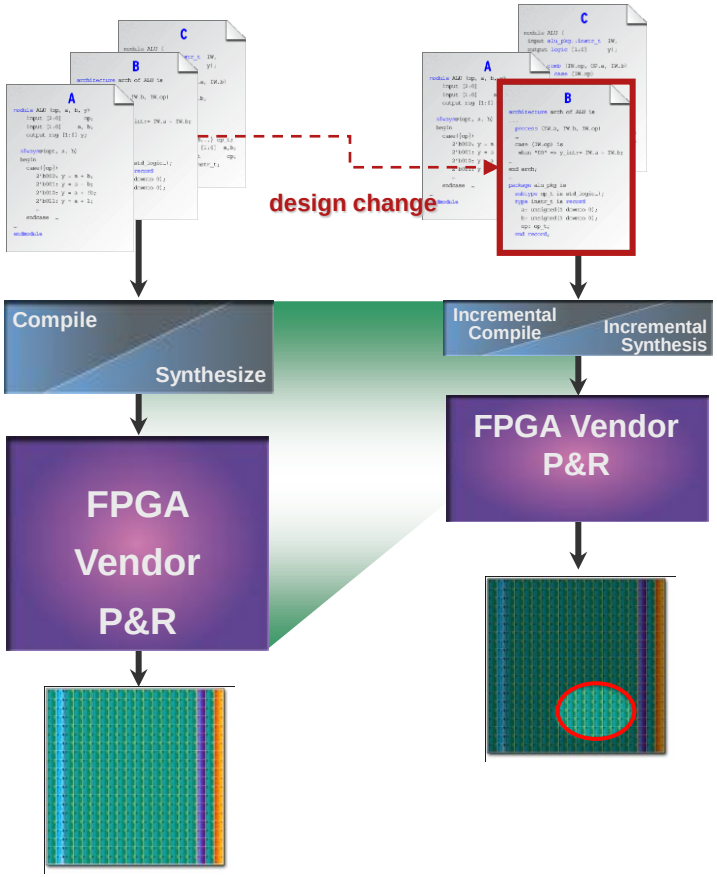
How Physical Synthesis Works



- Advanced Delay Estimation
  - Estimates location
  - Estimates routing resources
  - More accurately estimates net delays
  - Identifies Critical Paths
- Netlist Optimization
  - Retiming, Replication, Re-synthesis
- No placement sent to P&R
  - No DRC/packing violations
  - Maximum flexibility for P&R

Automatic Incremental Synthesis

- Up to 60% runtime savings
  - Design, change dependent
- Industry's only automatic incremental synthesis
  - *No partitioning or prior planning*
  - Maintains QoR with cross-hierarchy optimizations
  - Based on real changes in parse tree
- All FPGA families supported



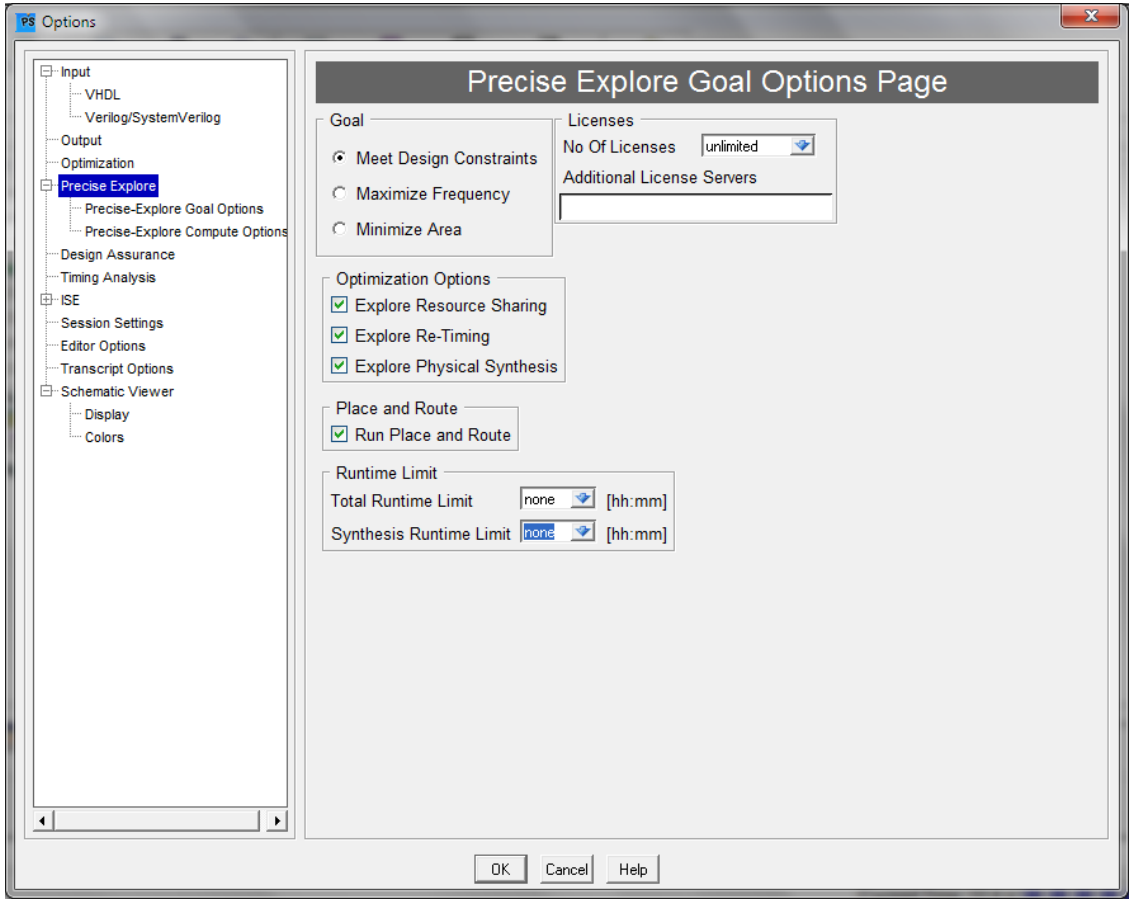
PRECISE-EXPLORE

The Need for Design Exploration

- Each design has unique characteristics
- Very time consuming to try all synthesis options for each individual design

Precise-Explore

- Exploration Goals
  - Meet Constraints
  - Max Frequency
  - Min Area
- Exploration Options
  - Turning off will use selected mode from other options page
- Place & Route
- Runtime Limits



Precise-Explore
Benefits

- Precise-Explore automates the process
- Explore implementation options for different design structures in both synthesis and P&R
- Control for exploration options
- Multiple computing options
- Detailed reporting

Precise-Explore
Product Configuration

- 2012b
  - Controlled feature
  - Requires special license
  - License available upon customer request

- 2012c *
  - Production feature
  - Included in Precision RTL Plus license

* Planned, subject to change

Precision Synthesis 2012b Summary

- Support for the latest FPGA devices & software
- Expanded SystemVerilog coverage
- Explore your FPGA design with Precise-Explore

Mentor Graphics

THANK YOU