



design & verification
conference & exhibition

February 22-24, 2006

Embedding Formal Techniques To Direct Test Suites:
A Path To Rapid Exhaustive
Functional Verification

Jeff Li

Superior Logic Corporation

Clem Meas

quickSTART Consulting

Agenda

- Where does VPE fit in
 - Exhaustive/prefect/complete & flexible
 - Support current methodologies
 - Complementary rather than overlapping
- How to use VPE
 - Easy input: no coding / drawing required
 - Fast runtime: minutes (not hours) each ...
 - Clean output: familiar to everybody
- Examples: beyond what's in verification plans

1) "Real World" Exhaustive Verification

superior-logic.com
verificationplan.com

- Fit into tight budget / schedule
 - Exhaustive+fast for all aspects or user selected aspects
- Be flexible (on-demand)
 - Allow late and partial switch to/from exhaustive
 - Little wasted/replicated work due to the late switch
- Share code/execution with non-exhaustive verification!
 - Directed or/and randomized simulation/emulation/lab
 - Testbenches, HVL scoreboards? Reference models? ...
 - Minimal learning curve

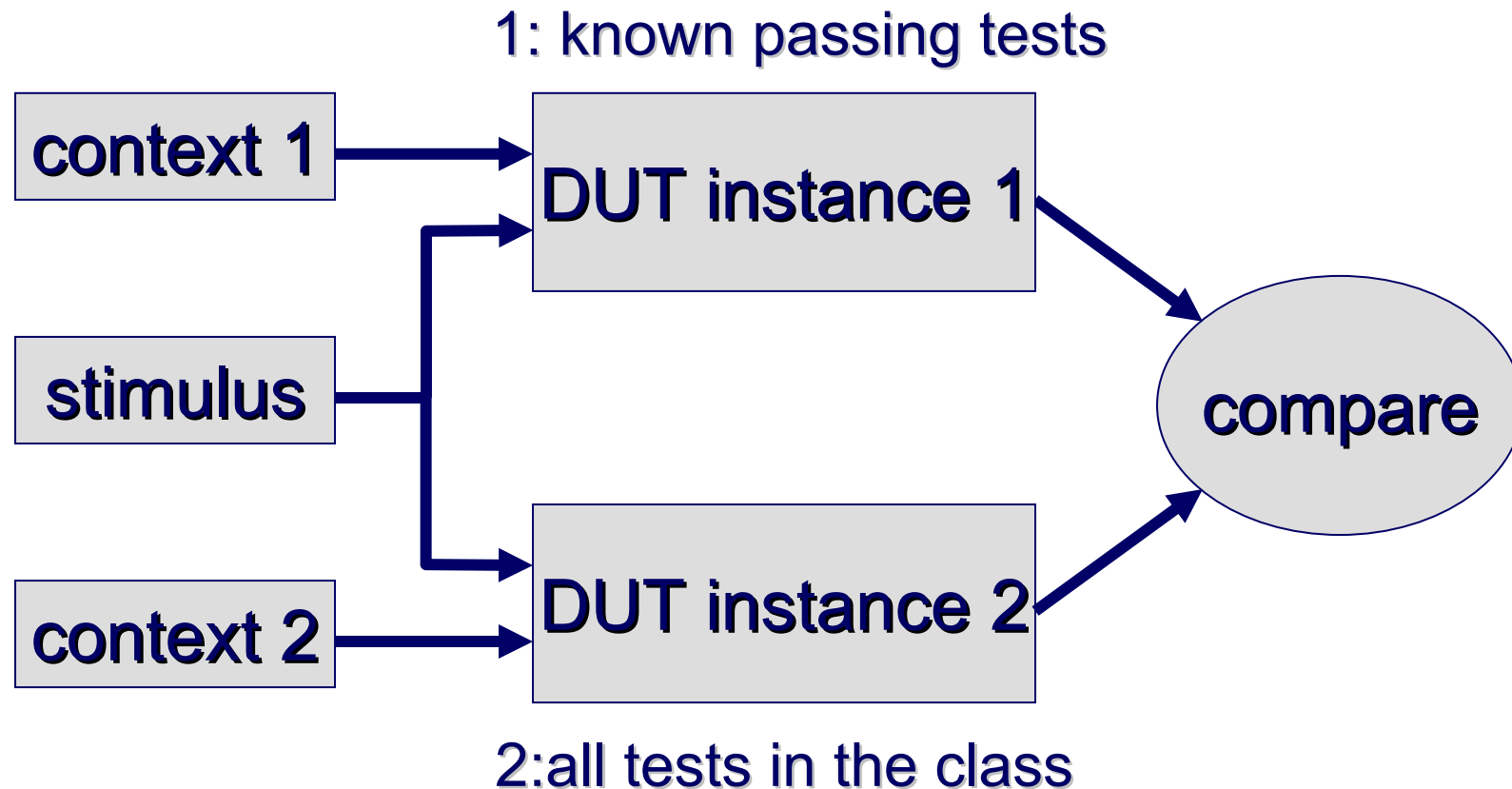
2) Efficiency: Low Redundancy

- Mixing: complementary and overlapping?
 - Chip level and block level
 - Simulation, emulation, post-silicon and formal
 - Testbenches and assertions
 - Similar tests
 - ...
- Need clean boundaries between them
 - Easier if each does 100% of something
- VPE's exhaustiveness enables reducing overlap
 - Between VPE and the old ones
 - Between the old ones

New in VPE: Similarity Class of Tests

- Definition: All tests in the class pass if one (or some) passes
 - Specific to a design under verification (DUT)
 - Response to each test satisfies the PASSing condition
 - Stimulus of each test satisfies a condition (per spec.)
- Same basic idea as coverage or corner test
 - Each coverage point needs to be hit only once (?)
 - Many tests are not needed if a corner test is used (?)
- One mathematical difference from bounded model checking
 - A test in the similarity class must be known passing

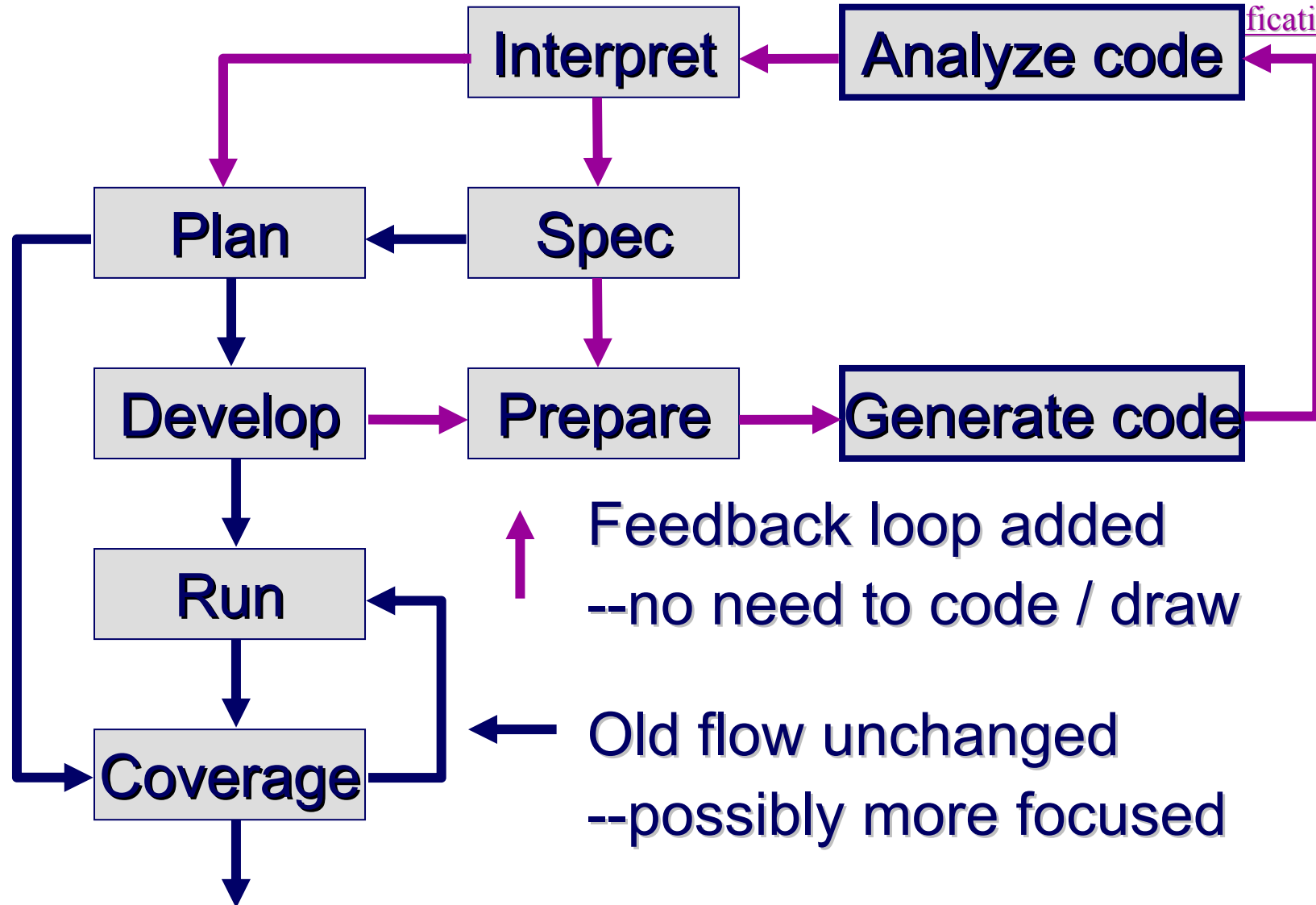
Model for proving a similarity class



Very simple code like parameterized directed tests

Work well with any existing flow

superior-logic.com
verificationplan.com



Feedback loop added
 --no need to code / draw

Old flow unchanged
 --possibly more focused

Full duplex Ethernet core

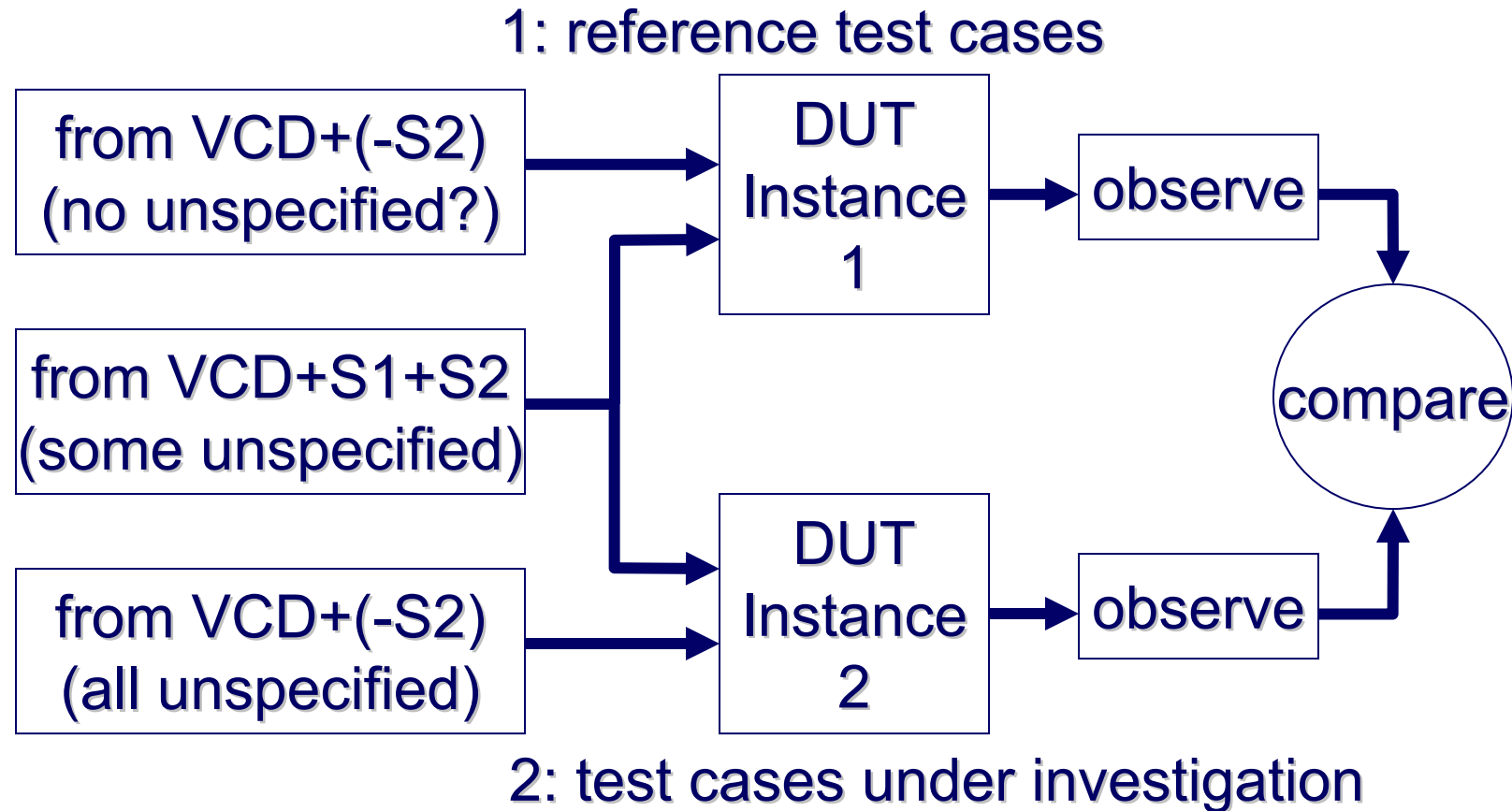
- **No tests** with 2-way traffic are included in the test suite
 - Significantly lower verification workload
- Receiving block and transmitting block **interact**
 - Sharing the buffer space and memory interface
 - Incoming traffic is able to control transmission
- VPE (Verification Plan Evaluator) proves OK
 - Incoming traffic does not disturb outgoing traffic
 - Receiving block's state does not, either
 - Considered all traffic patterns and all states
- Runtime <1 minute with >50K bits in unspecified values

Simple Kind of Similarity Classes

- A reference test per class: R (in test suite, to be passing)
- Passing: a specific part of the response is the same as R's
- Stimulus (w/ initial state): a specific part is the same as R's
 - Expected to cause the same passing condition
 - The remaining part shows the differences among tests
- R can be unknown if the 2 "specific parts" are known
 - For randomized stimulus generation
 - The 2 "specific parts" are similar in many classes
 - Process LOTS of related classes in one shot
- Differences among tests are shown as unspecified values

Generated comparison model

superior-logic.com
verificationplan.com

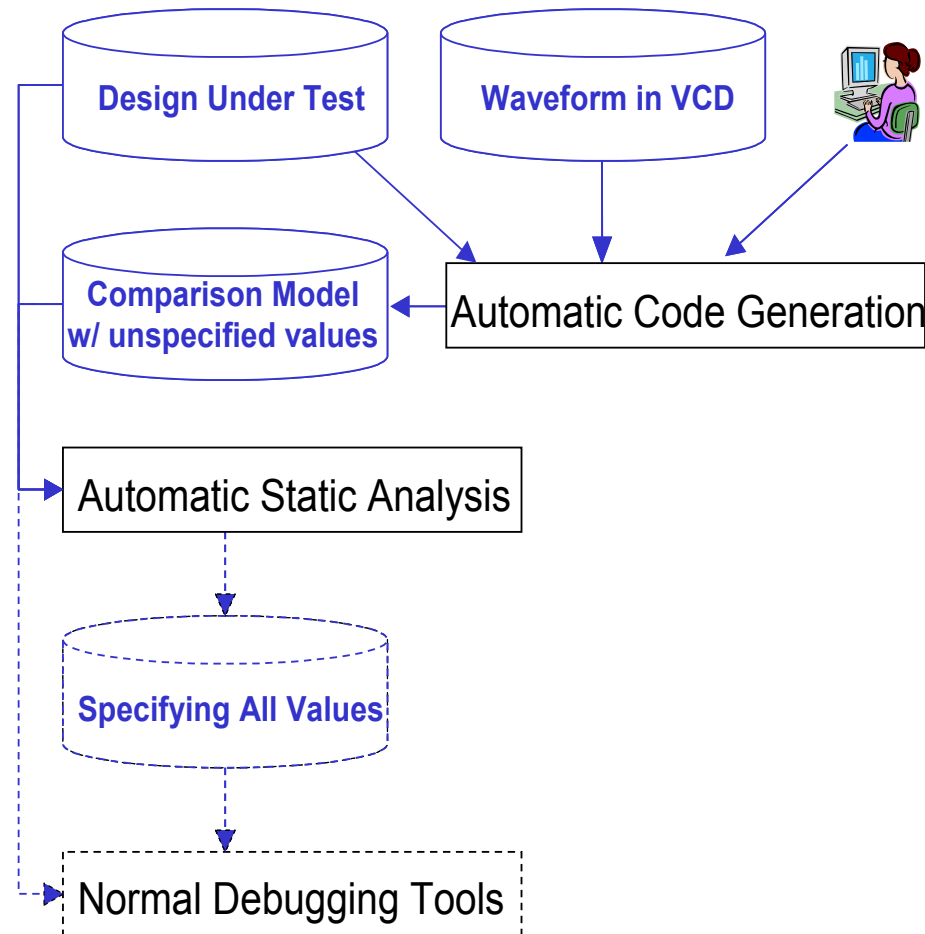


No required coding / drawing

- Comparison model generation based on VCD of:
 - Primary input values of DUT
 - Initial values of memory elements in DUT
- User's selection from VCD (waveform):
 - S1: unspecified values in reference test cases
 - S2: required consistency between reference test cases and test cases under investigation
 - Relevant response to observe / expect
 - at internal signals or primary outputs
 - support timing flexibilities (with waiting loops)

Files for the two automatic steps

- Input:
 - Verilog RTL (DUT)
 - VCD waveform
- Intermediate:
 - Comparison model in Verilog with unspecified values
- For debugging:
 - Assign constants to unspecified values



Low verification cost in all 3 steps

- Code development: automated (less bugs)
- Runtime speedup over simulation: many thousand orders of magnitude (possibly $> 10^{100,000}$)
 - Minutes each run even for very large designs
 - Many thousand bits in unspecified values
- Quick debugging (flaw in code or in doc / plan?)
 - Old debugging environments
 - Short test cases

Summary: Verification Plan Evaluator

superior-logic.com
verificationplan.com

- Expand verification to cover **ALL critical functional bugs**
 - Catch negligence in verification plan and in design specification
 - Flexible definition of what's critical
 - Use both stimulation and observation sides (highest standard)
- Work with **ALL functional verification flows/languages**
 - Only need VCD and DUT source in Verilog
 - Use 2 DUT instances (RTL or gate) and unspecified values
 - Massively expand coverage (not code/functional coverage)
- Low cost for **ALL costly (time/resource consuming) steps**
 - Automated code development (no required coding/drawing)
 - Dramatic runtime speedup (comparing to simulation)
 - Fast debugging without new tools (due to short test cases)

Example: an input is neglected

superior-logic.com
verificationplan.com

$m0 = \sim b \ \& \ \sim a;$

$m0 = \sim c \ \& \ \sim b \ \& \ \sim a;$

$m1 = \sim b \ \& \ a;$

$m1 = \sim c \ \& \ \sim b \ \& \ a;$

$m2 = b \ \& \ \sim a;$

$m2 = \sim c \ \& \ b \ \& \ \sim a;$

$m3 = b \ \& \ a;$

$m3 = \sim c \ \& \ b \ \& \ a;$

- Design spec. does not have "c"
- Verification plan includes 4 tests
- The difference between the above two implementations is not caught

	c	b	a
T0:	0	0	0
T1:	0	0	1
T2:	0	1	0
T3:	0	1	1

What to select [with example]

- Relevant response: define critical bugs [all m^*]
 - Not to check non-critical parts of response
- S2: relevant part of the test cases [a,b]
 - Determines values in the “relevant response”
 - Outside S2 (-S2): supposed background noise [c]
 - Can “background noise” cause serious trouble?
- S1: difference among planned test cases [a,b]
 - If inside S2: cause difference in relevant response
 - If outside S2: any of them can be selected from plan

User Input (v 0.1)

Red:
required

Other:
optional
(no typing
required)

top design source file name: demux.v

top design module name: deMux

main clock name: clk

VCD file name: demux.vcd

read VCD only between times 0 and 5

instance 1 input waveform: a = 1'b0;
b = 1'b0;
c = 1'b0; (select to turn into wild cards)
(also the table in input1.txt)

instance 2 input waveform: a = 1'b0;
b = 1'b0;
c = 1'b0; (select to tie across 2 instances)
(also the table in input2.txt)

instance 1 initial state: .m3 = 1'b0;
.m0 = 1'b1;
m1 = 1'h0; (select to turn into wild cards)
(also the column in init1.txt)

instance 2 initial state: .m3 = 1'b0;
.m0 = 1'b1;
m1 = 1'h0; (select to tie across 2 instances)
(also the column in init2.txt)

observe response after signal gets value between cycles and

shared observation contents: 1 1 .m0 1 bit
1 1 .m1 1 bit
1 1 m2 1 bit;

testbench file name: demuxvpe.v