2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Coverage Models for Formal Verification

ORACLE® Oracle Labs

Xiushan Feng
Oracle Labs, Austin TX

Xiaolin Chen, Abhishek Muchandikar

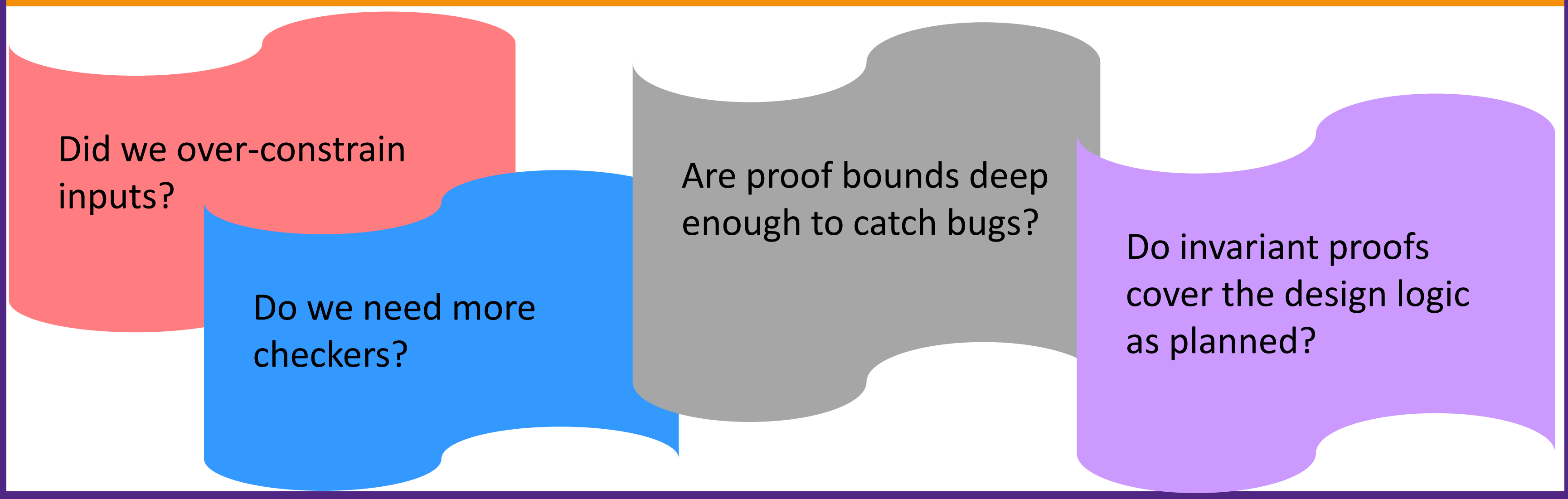
Synopsys Inc., Mountain View, CA

SYNOPSYS®

Abstract

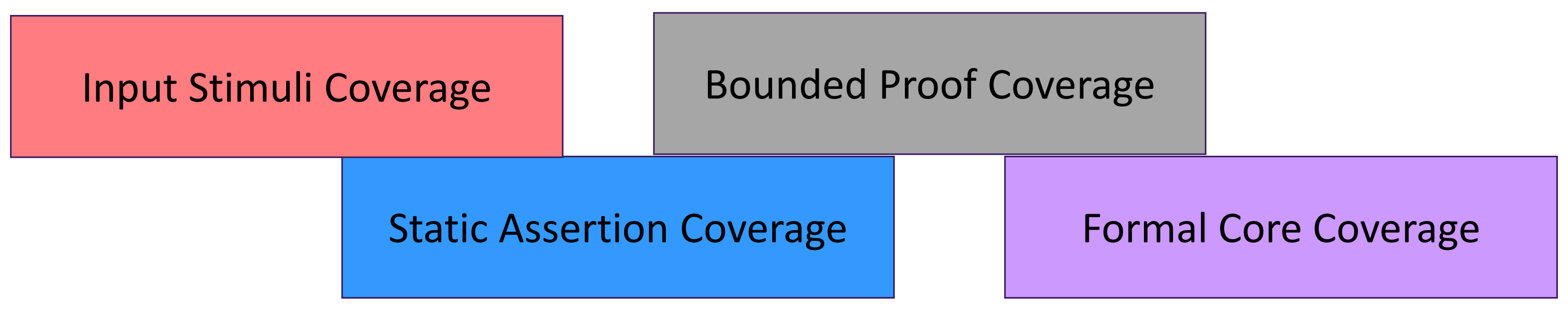
As formal verification engineers, the authors always face challenges to accurately access the current status of test benches. Many questions need to be answered at certain stages of a project. E.g., do we need more assertions? Did we over-constrain inputs that caused the drop of an important design scenario? Are proof bounds for bounded proofs good enough to catch potential design bugs? For the properties that are fully proven, do they cover the design logic that were intended to cover? We cannot get answers to these four most-asked questions without extracting information from formal engines, which is not feasible for general users. However, like coverage metrics from simulation-based verification, formal verification coverage models can be defined and used as metrics to measure formal verification progress and completeness. Some academic research on formal verification coverage and commercial formal verification tools are starting to support some coverage usages in the past two years. However, none of them clearly specified what engineers would really need and provided a good way to present formal coverage results in a standard way.
In this paper, the authors will introduce formal verification coverage models and their usages by real-life examples. The four most-asked questions finally have reasonable and acceptable answers supported by metrics.

Are We There Yet?


Did we over-constrain inputs?
Do we need more checkers?
Are proof bounds deep enough to catch bugs?
Do invariant proofs cover the design logic as planned?

Coverage Rises Up to the Challenge

- Objective metrics of a formal verification test bench
- Coverage Closure methodology same as simulation based verification closure

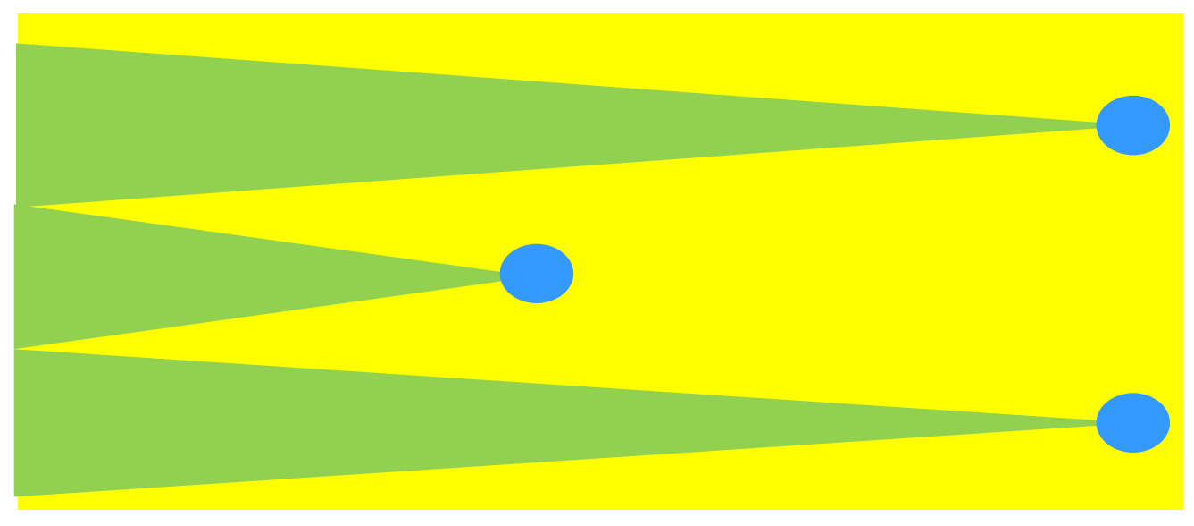


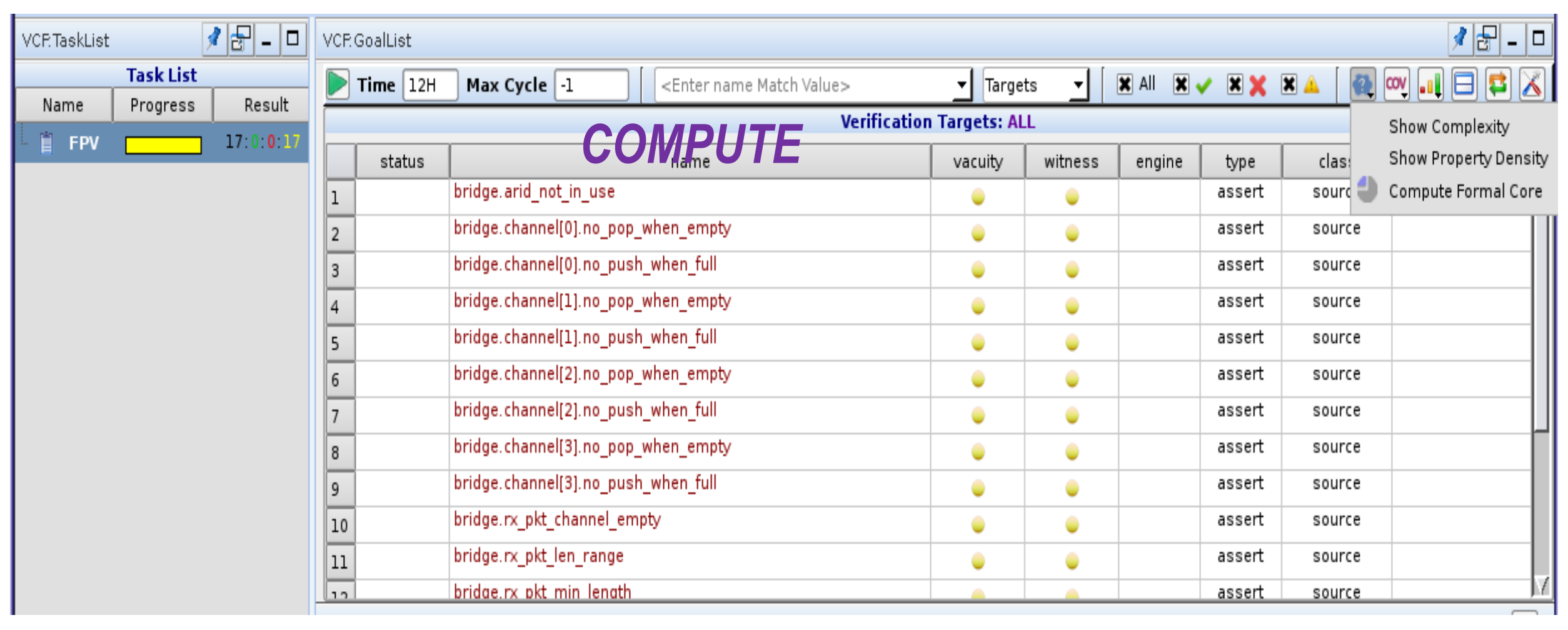
Static Assertion Coverage: *Do we need more checkers?"*

Coverage Model to check whether measure whether enough assertions have been written to cover the design spaces that we intend to check

$$COI\,(tb) = \frac{\sum(\bigcup_0^{n-1} coi(ast_k))}{\sum total}$$

- ast_k is one of n assertions of testbench tb. n, k is an integer
- $\sum$ is an operator to get the number of coverage targets
- $\sum$ total is the total number of coverage targets within tb
- $COI()$ is the function to compute cone of influence of an assertion



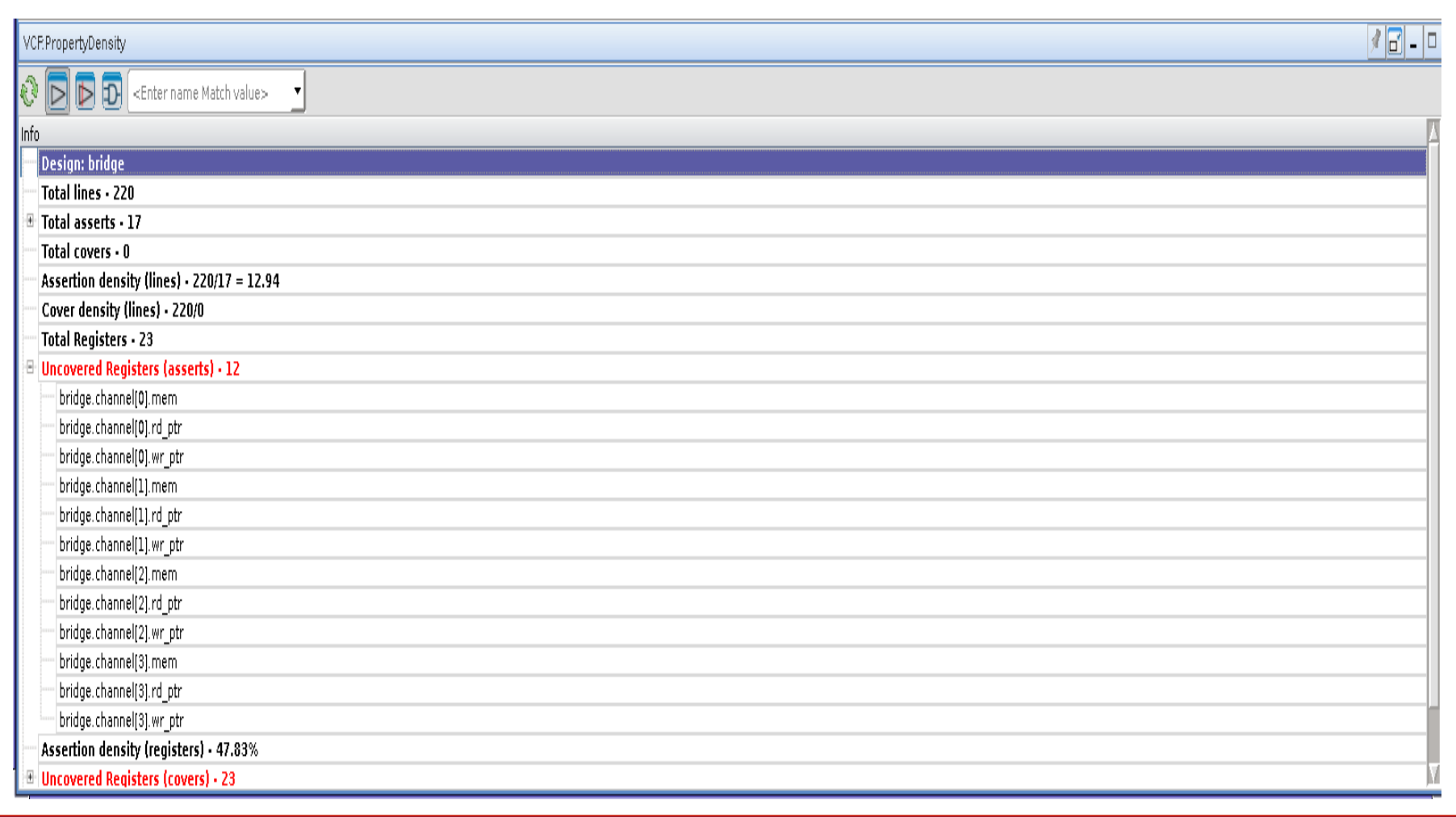


Name	Progress	Result
FPV		17:0:0:17

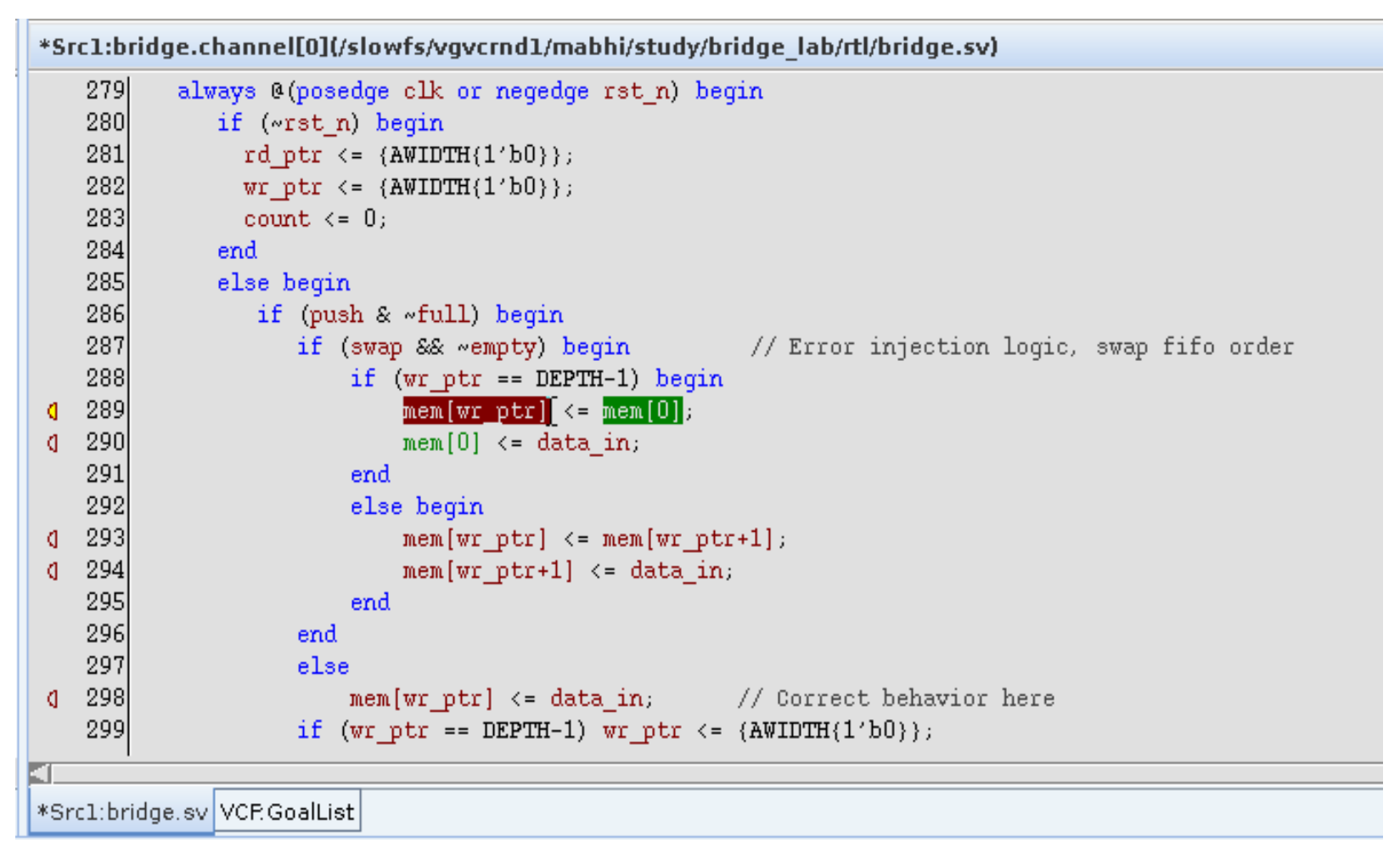
Time 12H Max Cycle -1 <Enter name Match Value> Targets

Verification Targets: ALL

	status	name	vacuity	witness	engine	type	class
1		bridge.arid_not_in_use	●	●		assert	source
2		bridge.channel[0].no_pop_when_empty	●	●		assert	source
3		bridge.channel[0].no_push_when_full	●	●		assert	source
4		bridge.channel[1].no_pop_when_empty	●	●		assert	source
5		bridge.channel[1].no_push_when_full	●	●		assert	source
6		bridge.channel[2].no_pop_when_empty	●	●		assert	source
7		bridge.channel[2].no_push_when_full	●	●		assert	source
8		bridge.channel[3].no_pop_when_empty	●	●		assert	source
9		bridge.channel[3].no_push_when_full	●	●		assert	source
10		bridge.rx_pkt_channel_empty	●	●		assert	source
11		bridge.rx_pkt_len_range	●	●		assert	source
12		bridge.rx_pkt_min_length	●	●		assert	source


VCF:PropertyDensity

```
Info
Design: bridge
Total lines - 220
Total asserts - 17
Total covers - 0
Assertion density (lines) - 220/17 = 12.94
Cover density (lines) - 220/0
Total Registers - 23
Uncovered Registers (asserts) - 12
    bridge.channel[0].mem
    bridge.channel[0].rd_ptr
    bridge.channel[0].wr_ptr
    bridge.channel[1].mem
    bridge.channel[1].rd_ptr
    bridge.channel[1].wr_ptr
    bridge.channel[2].mem
    bridge.channel[2].rd_ptr
    bridge.channel[2].wr_ptr
    bridge.channel[3].mem
    bridge.channel[3].rd_ptr
    bridge.channel[3].wr_ptr
Assertion density (registers) - 47.83%
Uncovered Registers (covers) - 23
```


*Src1:bridge.channel[0](/slowfs/vgvcrnd1/mabhi/study/bridge_lab/rtl/bridge.sv)

```
always @(posedge clk or negedge rst_n) begin
   if (~rst_n) begin
     rd_ptr <= {AWIDTH{1'b0}};
     wr_ptr <= {AWIDTH{1'b0}};
     count <= 0;
   end
   else begin
      if (push & ~full) begin
         if (swap && ~empty) begin         // Error injection logic, swap fifo order
            if (wr_ptr == DEPTH-1) begin
               mem[wr_ptr] <= mem[0];
               mem[0] <= data_in;
            end
            else begin
               mem[wr_ptr] <= mem[wr_ptr+1];
               mem[wr_ptr+1] <= data_in;
            end
         end
         else
            mem[wr_ptr] <= data_in;      // Correct behavior here
         if (wr_ptr == DEPTH-1) wr_ptr <= {AWIDTH{1'b0}};
```

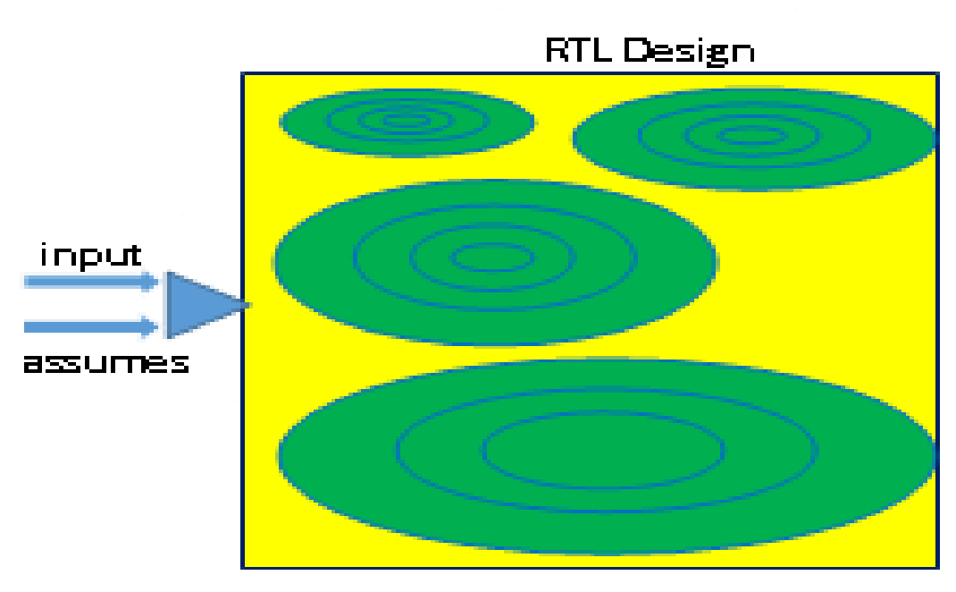
*Src1:bridge.sv VCF:GoalList

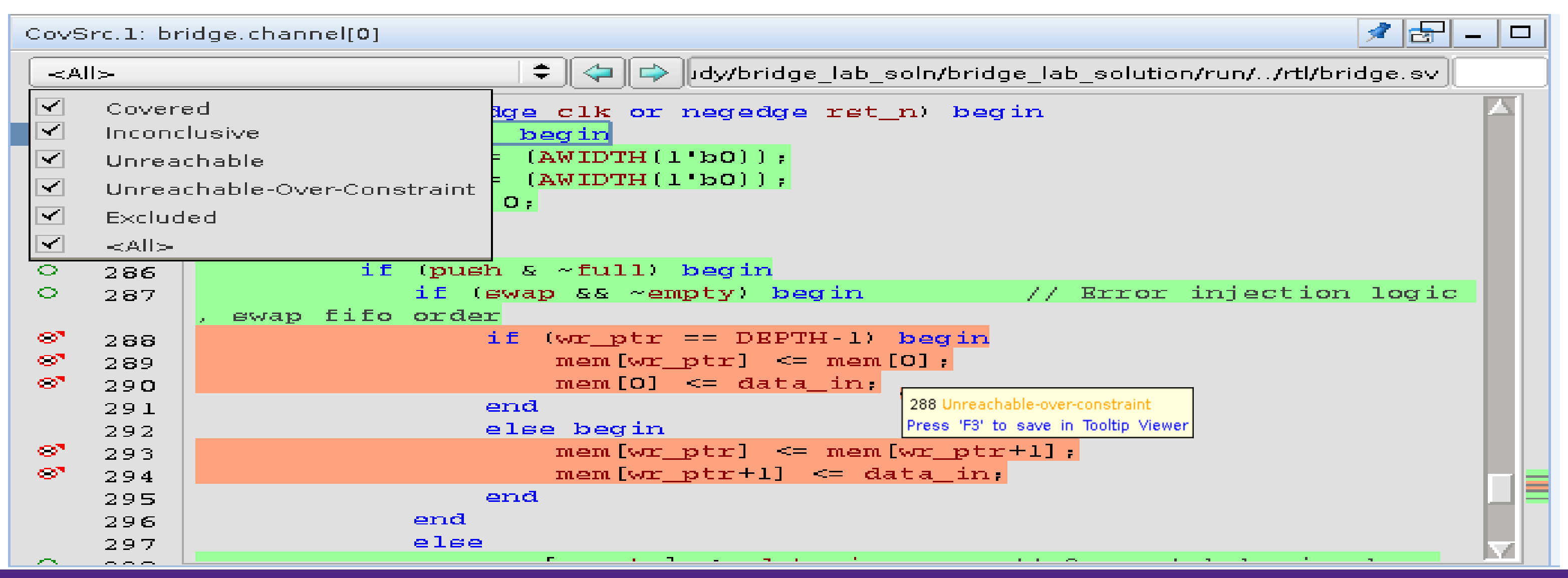
Stimuli Coverage : *Did I over-constraint my inputs?"*

Coverage Model focuses on how formal tools drive inputs to reach RTL design space under the current input constraints

$$Stimuli(tb) = \frac{\sum(\bigcup_0^{\infty}(C_i))}{\sum total}$$

- C_i is covered target at cycle i. $\bigcup_0^{\infty}(C_i)$ is the greatest fixpoint (GFP) of all reachable targets. $\sum(set)$ is the number of items inside set
- $\sum total$ is the total number of coverage targets within tb





udy/bridge_lab_soln/bridge_lab_solution/run/../rtl/bridge.sv

```
dge clk or negedge rst_n) begin
 begin
= {AWIDTH{1'b0}};
= {AWIDTH{1'b0}};
 0;

286          if (push & ~full) begin
287             if (swap && ~empty) begin          // Error injection logic
, swap fifo order
288                if (wr_ptr == DEPTH-1) begin
289                   mem[wr_ptr] <= mem[0];
290                   mem[0] <= data_in;
291                end
292                else begin
293                   mem[wr_ptr] <= mem[wr_ptr+1];
294                   mem[wr_ptr+1] <= data_in;
295                end
296             end
297             else
```

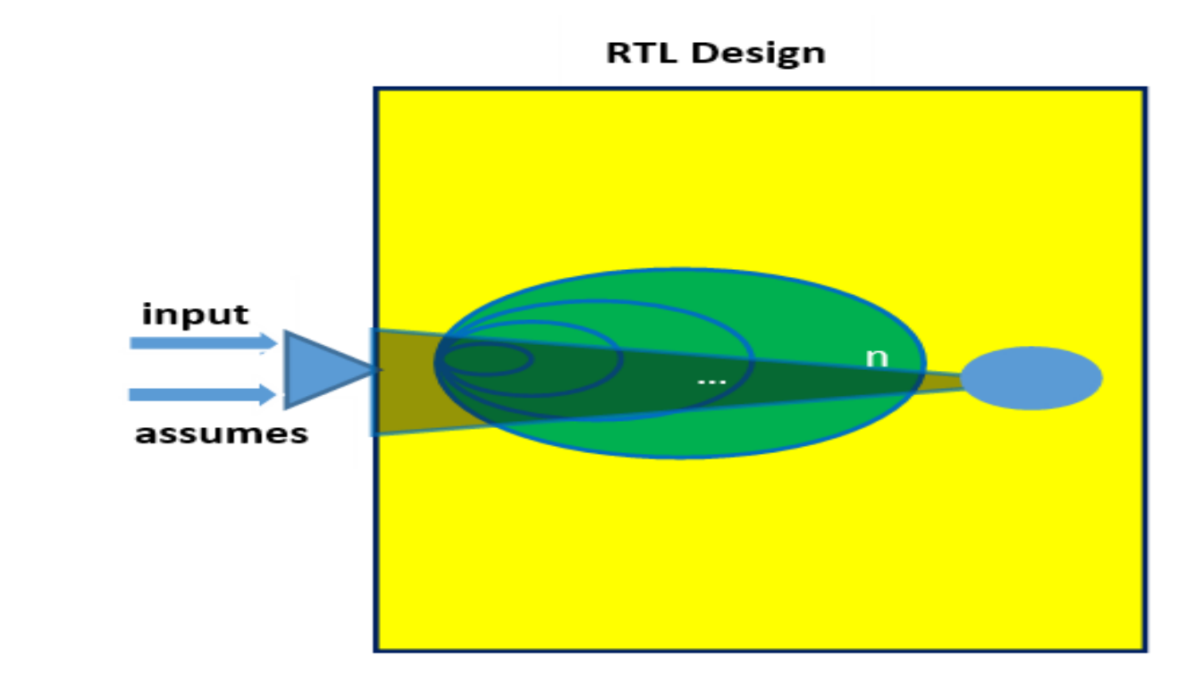
288 Unreachable-over-constraint
Press 'F3' to save in Tooltip Viewer

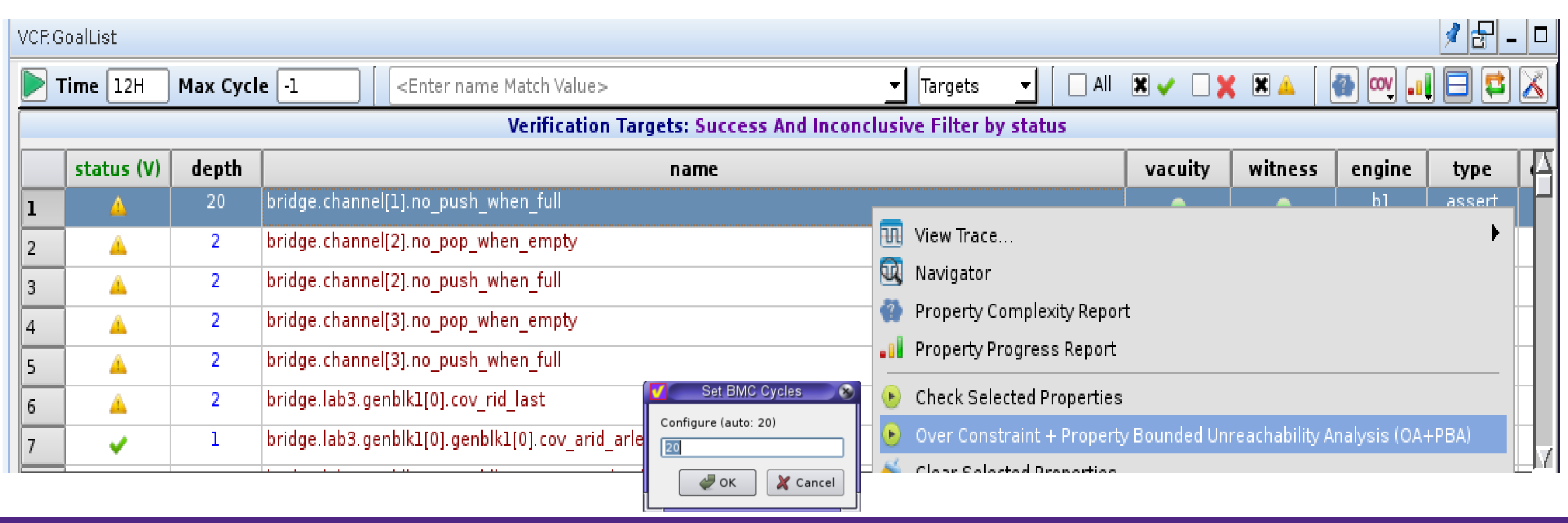
Bounded Proof Coverage : *Are bounded proofs good enough ?"*

Coverage Model focuses on how formal tools drive inputs to reach RTL design space under the current input constraints

$$bounded\_proof\,(ast\_set_k) = \frac{\sum((\bigcup_0^{n}(C_i))\cap coi(ast_i))}{\sum coi(ast_i)}$$

- $ast_i \in ast\_set_k$ i, k is an integer
- ast_i is an assertion that has a proof bound n; n, k is an integer
- $coi()$ is the function to compute cone of influence of an assertion




VCF:GoalList
View Trace...
Navigator
Property Complexity Report
Property Progress Report
Check Selected Properties
Over Constraint + Property Bounded Unreachability Analysis (OA+PBA)
Set BMC Cycles
OK
Cancel

Time 12H Max Cycle -1 <Enter name Match Value> Targets

Verification Targets: Success And Inconclusive Filter by status

	status (V)	depth	name	vacuity	witness	engine	type
1	⚠	20	bridge.channel[1].no_push_when_full	●	●	h1	assert
2	⚠	2	bridge.channel[2].no_pop_when_empty				
3	⚠	2	bridge.channel[2].no_push_when_full				
4	⚠	2	bridge.channel[3].no_pop_when_empty				
5	⚠	2	bridge.channel[3].no_push_when_full				
6	⚠	2	bridge.lab3.genblk1[0].cov_rid_last				
7	✓	1	bridge.lab3.genblk1[0].genblk1[0].cov_arid_arle				

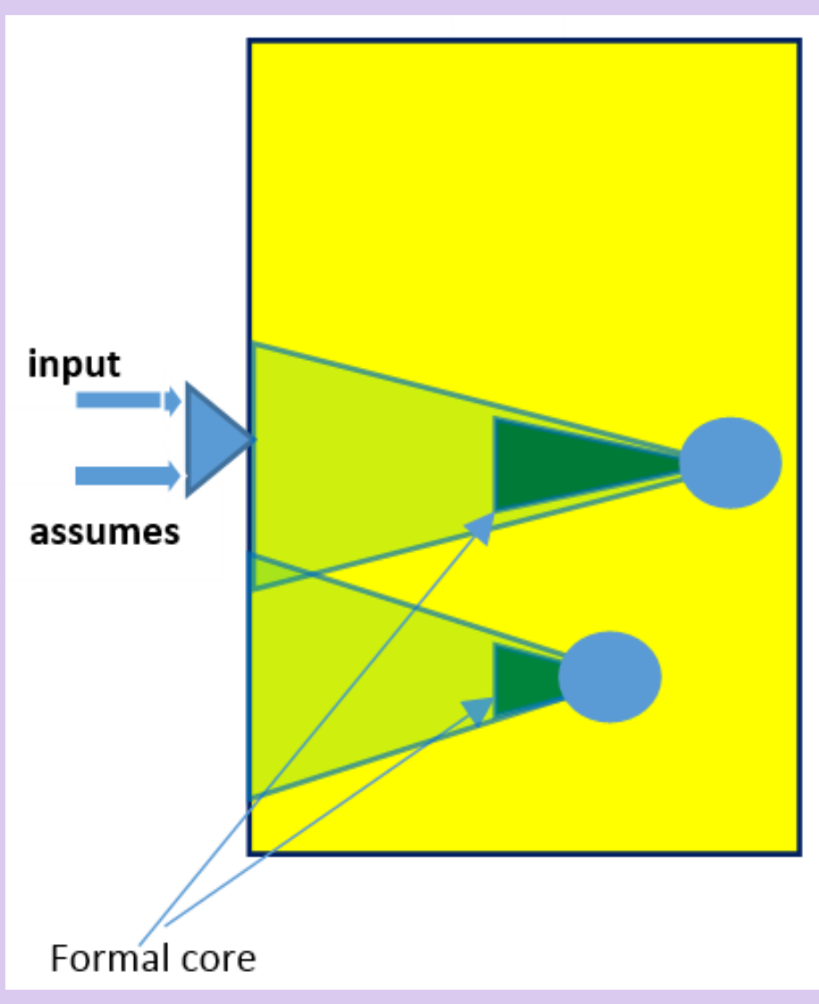
Configure (auto: 20)
20

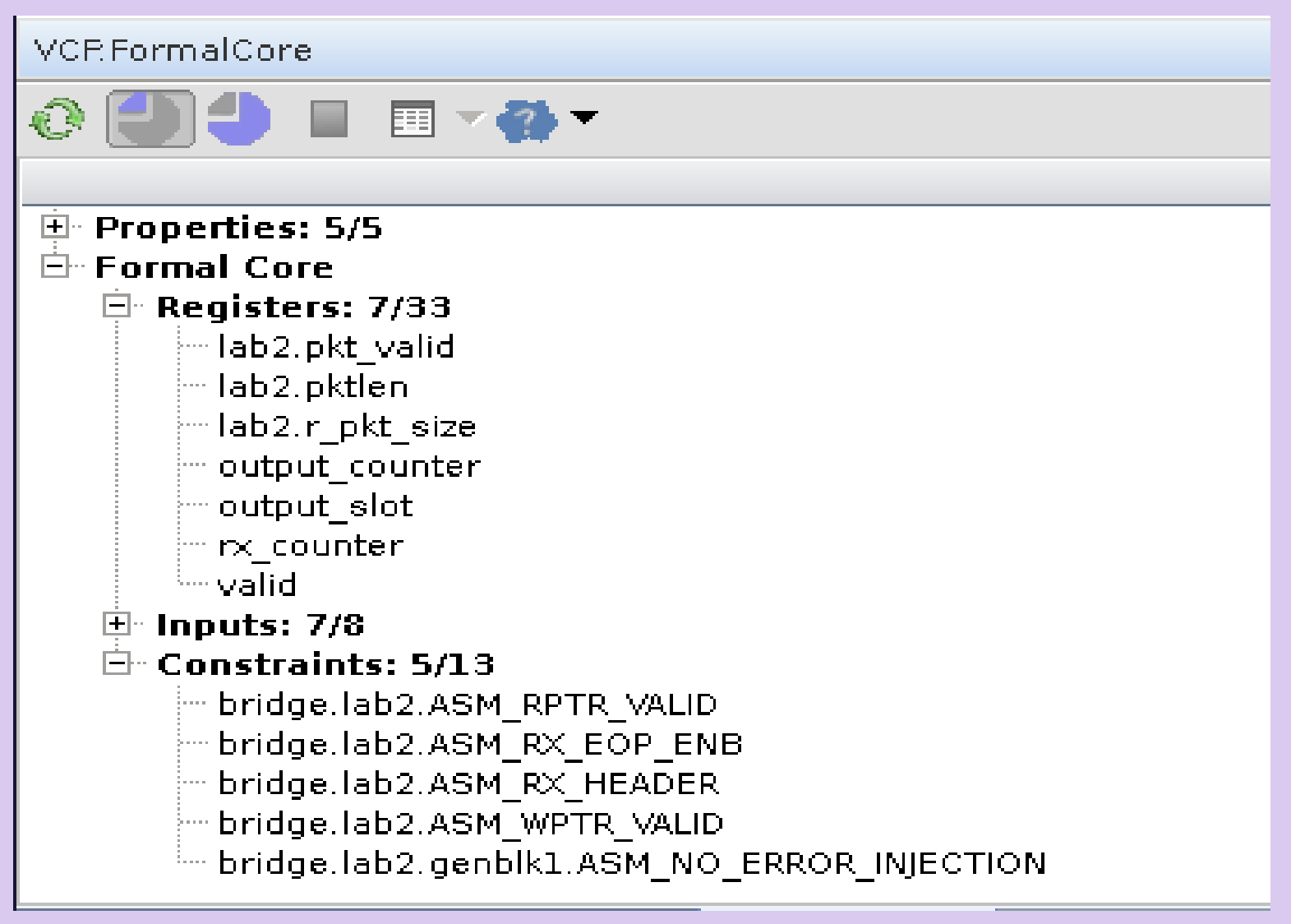
Formal Core Coverage : *Do proofs cover the design logic that were intended to cover ?"*

Coverage Model analyzes If there are design bugs outside the formal proof core but still within the COI of the assertion

$$proof\_core\,(ast\_set_k) = \frac{\sum(\bigcup proof\_core(ast_i))}{\sum coi(ast_i)}$$

- $ast_i \in ast\_set_k$ i, k is an integer
- $proof\_core(ast_i)$ is the set of targets actually used by formal engines to prove ast_i
- $coi()$ is the function to compute cone of influence of an assertion




VCF:FormalCore

```
+ Properties: 5/5
- Formal Core
   - Registers: 7/33
        lab2.pkt_valid
        lab2.pktlen
        lab2.r_pkt_size
        output_counter
        output_slot
        rx_counter
        valid
   + Inputs: 7/8
   - Constraints: 5/13
        bridge.lab2.ASM_RPTR_VALID
        bridge.lab2.ASM_RX_EOP_ENB
        bridge.lab2.ASM_RX_HEADER
        bridge.lab2.ASM_WPTR_VALID
        bridge.lab2.genblk1.ASM_NO_ERROR_INJECTION
```

Conclusion

Formal verification without coverage closure is the same as doing simulation without coverage closure. With the increasing usage of formal verification for circuit design, we expect these formal verification coverage models will become standard models for formal tools and are used by formal verification sign-off process.