The lights in the Tunnel: Coverage Analysis for Formal Verification

Xiushan Feng
Oracle Labs

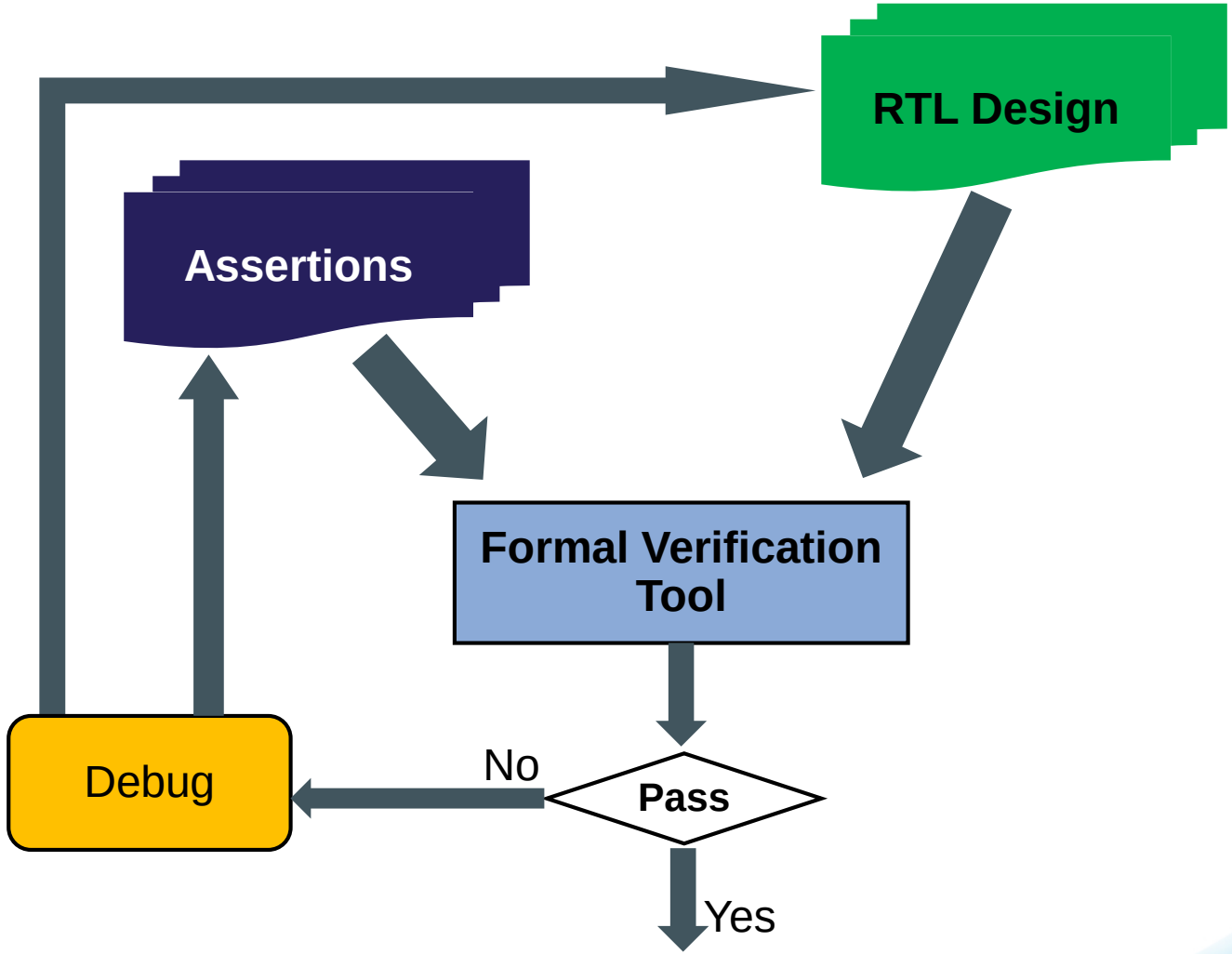
September 29, 2016
SNUG Austin

Agenda

- Background
  - Formal Verification
  - Challenges to Formal Verification Closure
- Proposed Coverage Models for Formal Verification
- Conclusion

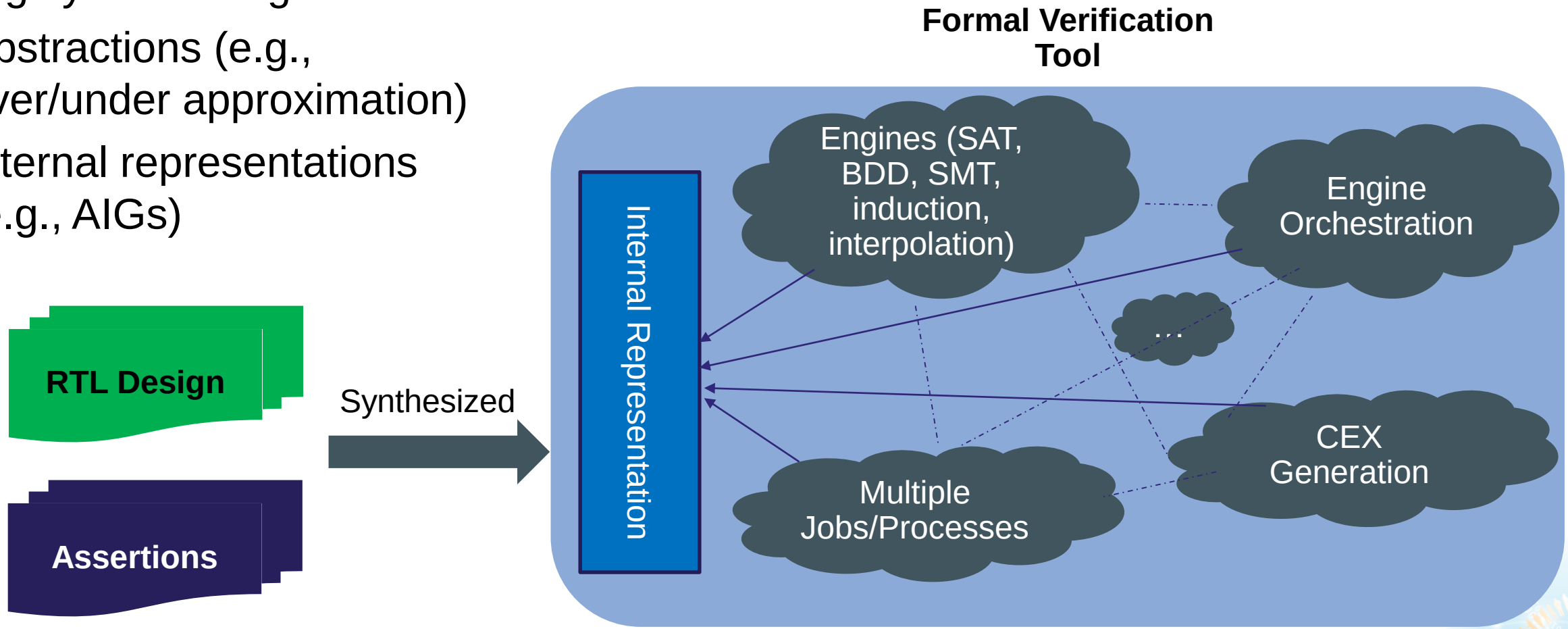
Formal Verification Process

- Assertions
  - Assume/assert/cover/restrict
- Formal verification tool
- Debug



Formal Verification Tool

- Sophisticated algorithms
- Highly-tuned engines
- Abstractions (e.g., over/under approximation)
- Internal representations (e.g., AIGs)



Four Most-asked Questions

- Do we need more assertions?
- Did you over-constrain inputs?
- Are proof bounds for bounded proofs good enough?
- For proven properties, do they cover all the logic to be verified?

Our Solution: Coverage Models for Formal Verification

Proposed Coverage Models

- Static Assertion COI (Cone Of Influence) Coverage
- Input Stimuli Coverage
- Bounded Proof Coverage
- Proof Core Coverage

Static Assertion COI Coverage

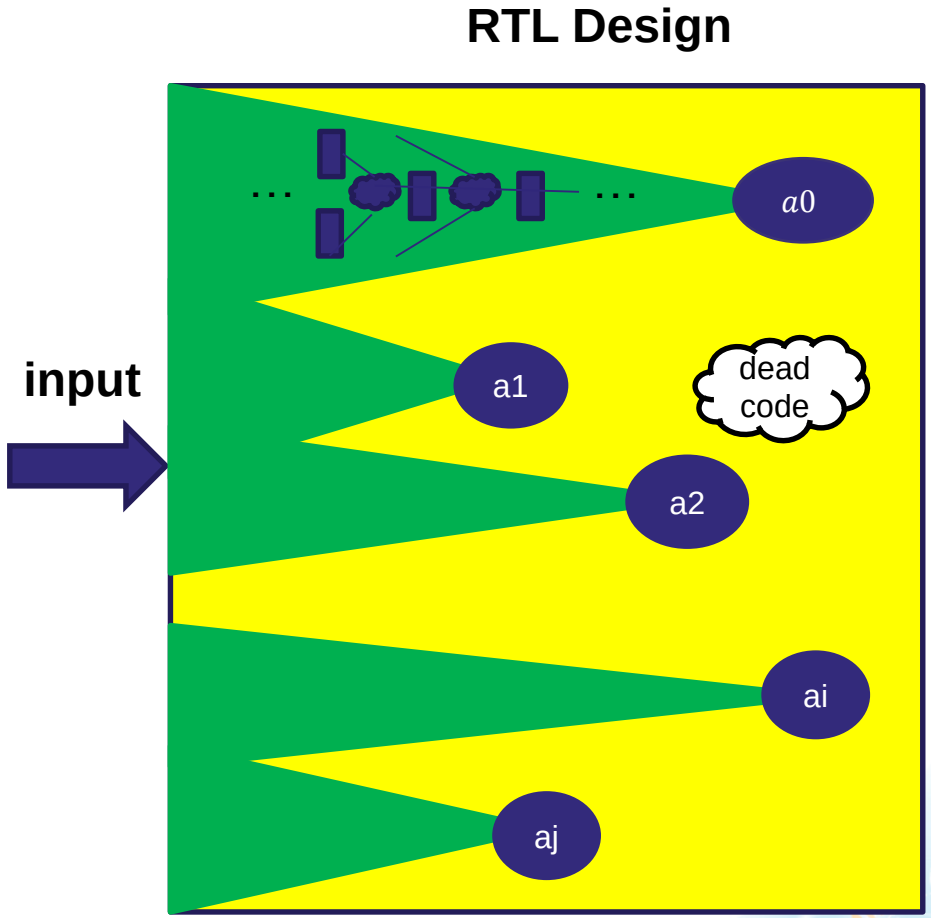
- Definition
  - Static Assertion COI Coverage

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

    - ast_k *is one of* n *assertions of testbench* $tb.n, k$ *is an integer*
    - $\sum$ *is the total number of coverage targets within tb*
    - *coi() is the function to compute cone of influence of an assertion*
  - No assumption is needed for fast runtime
  - Possible targets, line/condition/register, etc.
- For coverage closure, we are more interested in this: $total \setminus \bigcup_0^{n-1} coi(ast_k) \setminus deadcode$

  i.e. targets that are not covered by any assertion

  Deadcode is the set of targets that are un-coverable in the design



Static Assertion COI Coverage Cont.

COI coverage for registers:

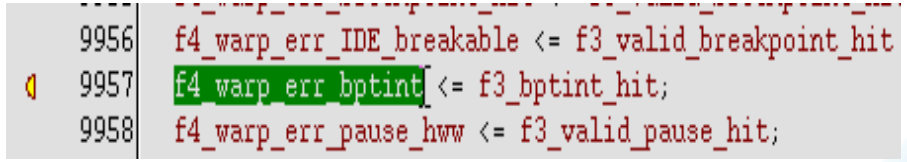
Number of instances analyzed	1612
Total number of registers	7607
Total number of assertions	5423
Total uncovered registers	18% (1375)
Total covered registers	82% (6232)

List of uncovered registers:

...

sub_partition0.f4_warp_err_bptint

...



Proposed Coverage Models

- Static Assertion COI Coverage

- **Input Stimuli Coverage**

- Bounded Proof Coverage

- Proof Core Coverage

Input Stimuli Coverage

- Definition
  - Input Stimuli Coverage

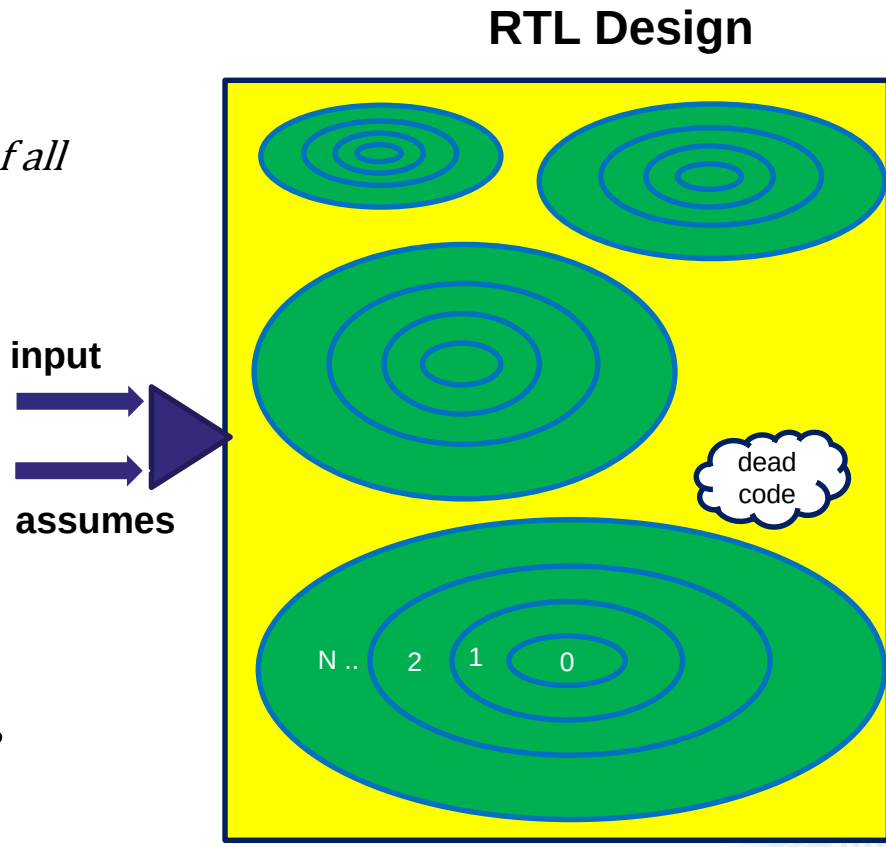
$$Stimuli(tb) = \frac{\sum(\bigcup_0^\infty(C_i))}{\sum total}$$

    - 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*
  - Reachability analysis under existing assumptions
  - No assertion is needed – we want to understand how assumptions constrain test bench
  - Possible targets, line/condition/register, etc…
- For formal coverage closure, this is more useful:

$$unreachable\ under\ stimuli = \ total \setminus (\bigcup_0^\infty(C_i)) \setminus deadcode$$

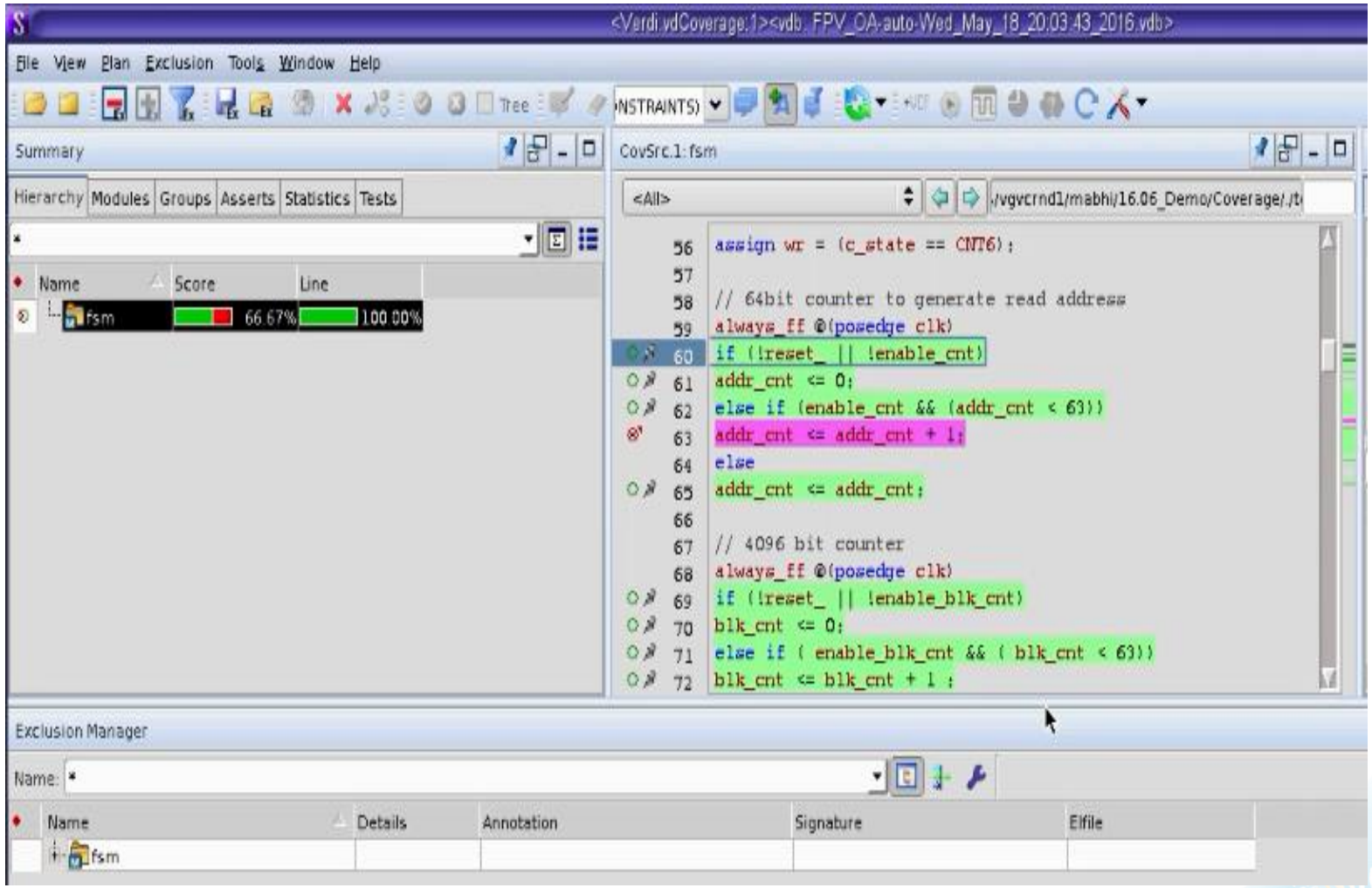
i.e. targets that are not exercised under current input constraints

Deadcode is the set of targets that are un-coverable in the design



Input Stimuli Coverage Cont.

- Practical issues
  - If a fixpoint cannot be computed due to time/mem out, use the current reachable targets under timeout/memout/bound setting
  - Exclusion
    - Deadcode
    - Targets under reset statements



Proposed Coverage Models

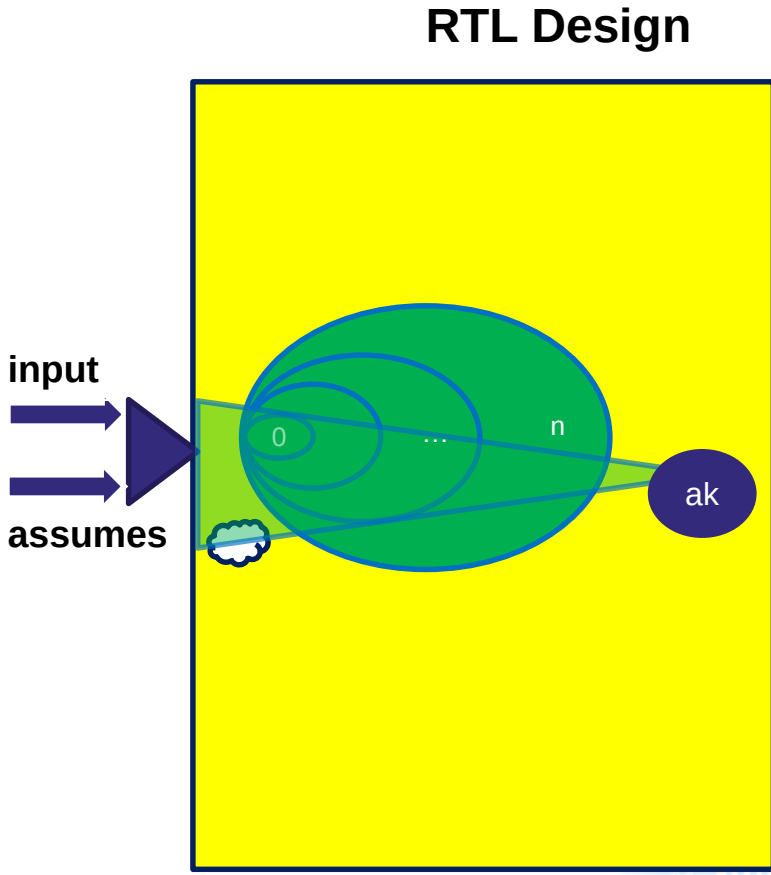
- Static Assertion COI Coverage
- Input Stimuli Coverage
- **Bounded Proof Coverage**
- Proof Core Coverage

Bounded Proof Coverage

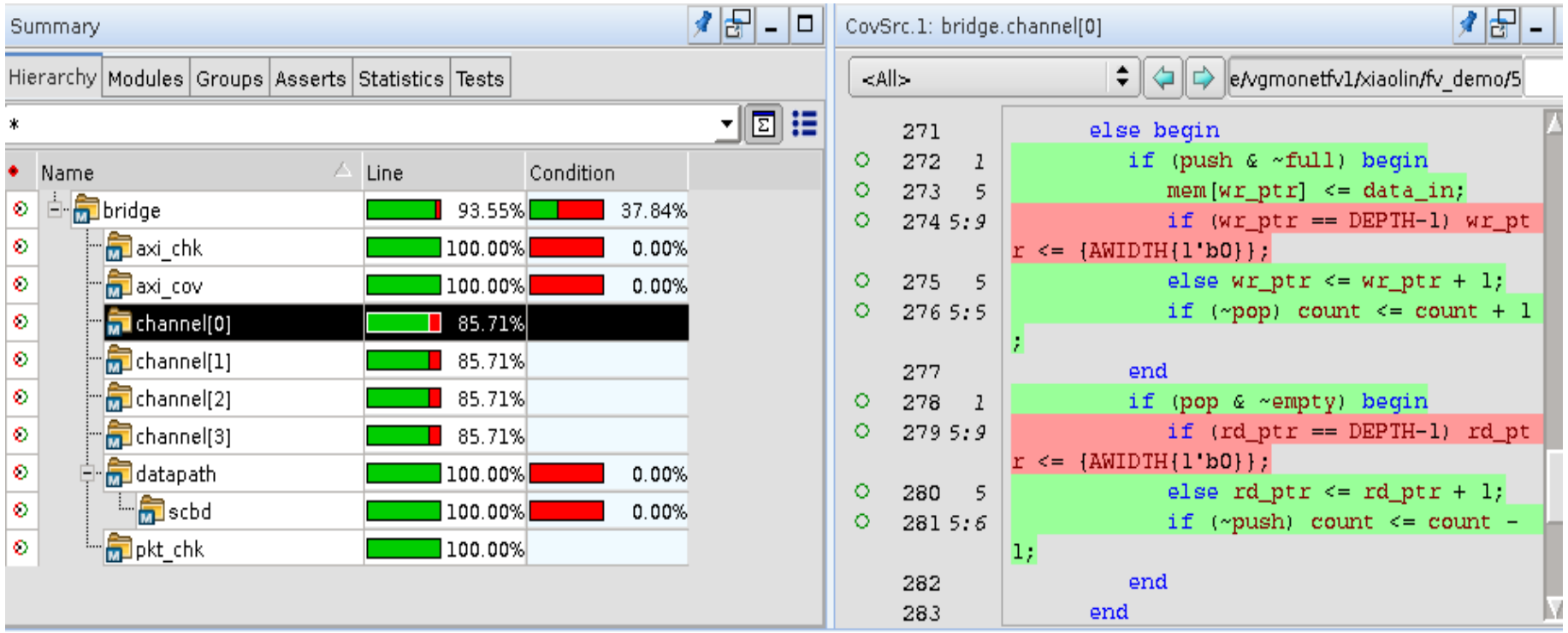
- Target
  - Bounded proof assertions with a minimum bound
  - Or one bound proof
- Given a bounded proof with proof bound *n*, bounded proof coverage can be defined as this:

$$bounded\_proof\ (ast_k) = \frac{\sum((\cup_0^n(C_i)) \cap coi(ast_k))}{\sum coi(ast_k)}$$

  - ast_k *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*
- This coverage will show which is not covered under the current proof bound
- In reality, we are more interested in this:
  - $coi(ast_k) \setminus (\cup_0^n(C_i)) \cap coi(ast_k) \setminus deadcode \setminus unreachable\ from\ Stimuli$



Bounded Proof Coverage Cont.

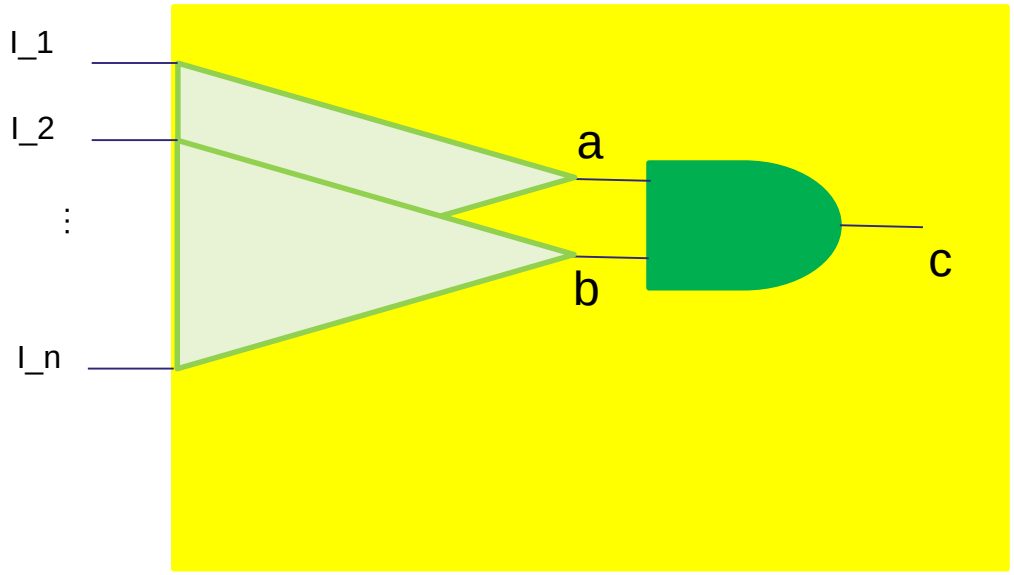


Proposed Coverage Models

- Static Assertion COI Coverage
- Input Stimuli Coverage
- Bounded Proof Coverage
- **Proof Core Coverage**

Proof Core Coverage Introduction

- Why proof core analysis
  - Not all logic inside COI is used to prove an assertion
  - Need to understand what logic is need (or not needed)
- Usages:
  - Detect verification holes
  - Exploit abstraction opportunities



assert c == a & b

If a formal engine is smart, only AND gate is needed to prove it!

For better coverage, can be re-written with inputs

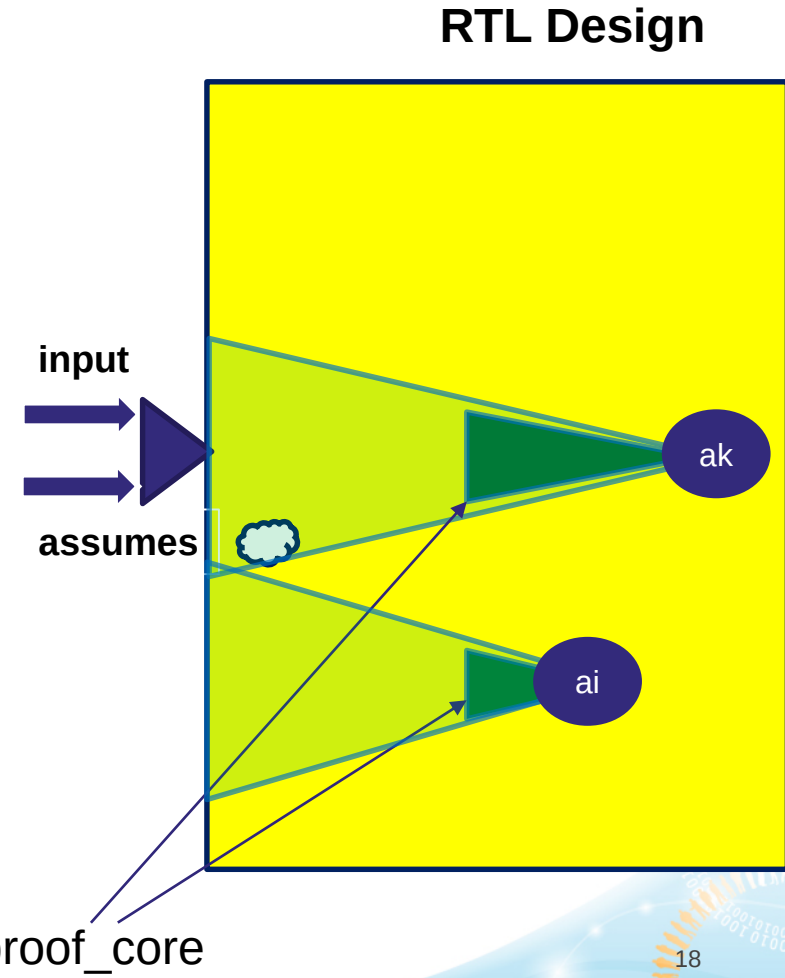
assert c == f(I_1, 1_2, …) & f (I_i,...)

Proof Core Coverage

- Target
  - All proven assertions or a small set of selected proven assertions
- Given a set of proven assertions, proof core coverage can be defined as this:

$$proof\_core\ (ast\_set_k) = \frac{\sum(\cup proof\_core(ast\__i))}{\sum coi(ast_k)}$$

  - $ast_i \in ast\_set_k \;\; i,k \; is \; an \; integer$
  - $proof\_core(ast\__i)$ *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*
- In reality, we are more interested in this:
  - $coi(ast_k) \setminus (\cup\, proof\_core(ast\__i)) \setminus deadcode \;\; \setminus \; unreachable \; from \; Stimuli$



Proof Core Coverage Cont.

```
vcf> report_formal_core -property {bridge.pkt_chk.ast_pkt_len_legal bridge.pkt_chk.ast_pkt_type_legal} -list
Formal core list view
-------------------------------------------------------------------------------------------------------------------------
 RunStatus |   #Registers |  #Inputs | #Constraints |       Status |        Depth |  SubType | Property
-------------------------------------------------------------------------------------------------------------------------
 completed |            0 |        2 |            0 |       proven |            - |        - | bridge.pkt_chk.ast_pkt_len_legal
 completed |            0 |        0 |            0 |       proven |            - |        - | bridge.pkt_chk.ast_pkt_type_legal
-------------------------------------------------------------------------------------------------------------------------
Formal core verbose view
Property      : bridge.pkt_chk.ast_pkt_len_legal
SubType       :     -
Status        : proven
Depth         :     -
#Registers    :     0
#Inputs       :     2
    ARLEN
    ARVALID
#Constraints  :     0

Property      : bridge.pkt_chk.ast_pkt_type_legal
SubType       :     -
Status        : proven
Depth         :     -
#Registers    :     0
```

Message | VCF Shell

Conclusion

- Four most-asked questions answered by four coverage models
  - Doing formal verification without coverage – walk in the darkness
- Usages
  - Identify verification holes
  - Measure progress of formal test bench
  - Provide hints for abstractions to converge and optimize
- Status
  - A few key formal tool vendors listened and provided prototype implementations
  - More support needed to make coverage models usable to broader users

ORACLE

snug
Synopsys Users Group
Austin

Thank You