Discovering the Input Assumptions in Specification Refinement Coverage

P. Basu, S. Das, Pallab Dasgupta, P.P. Chakrabarti

Dept. of Computer Science & Engineering,
Indian Institute of Technology Kharagpur, India

The state of Formal Property Verification

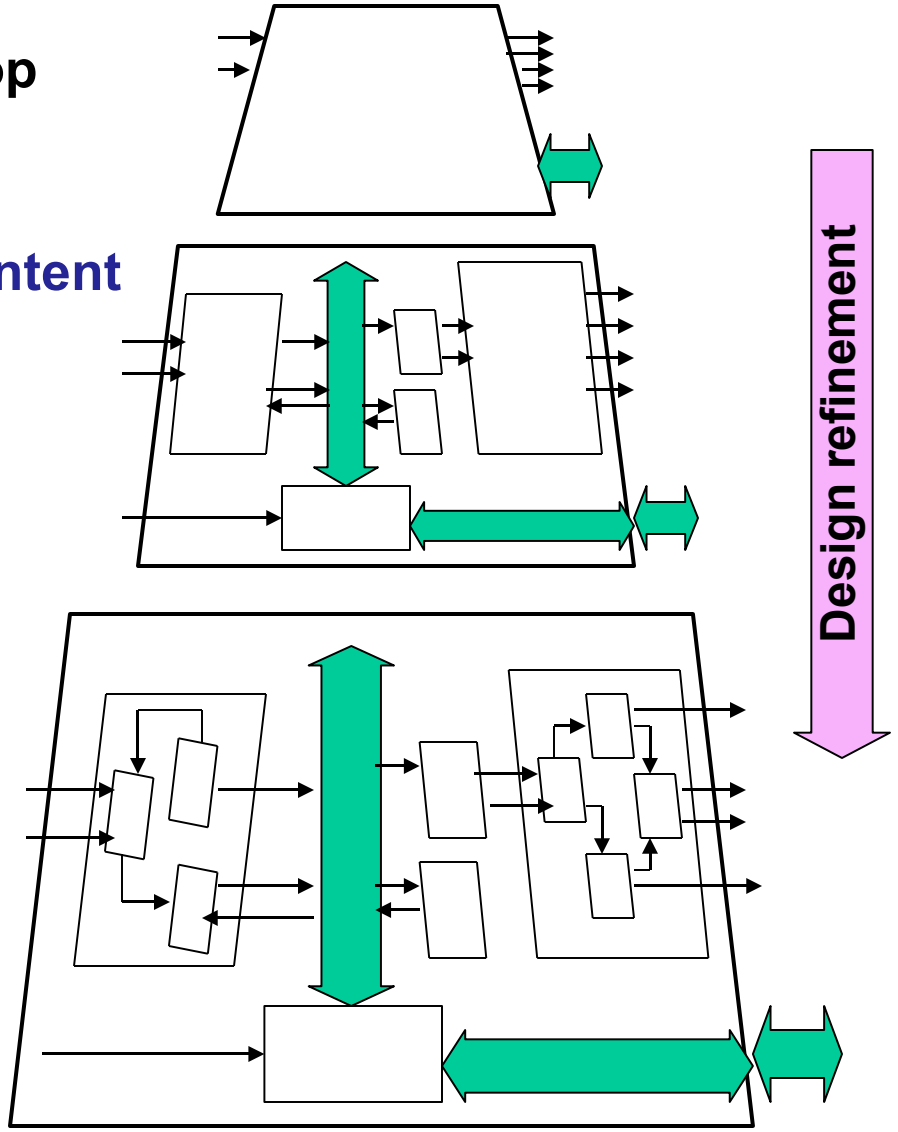
- **We know the benefits**
  - Non-ambiguous specification of the design intent
  - Exhaustive verification of the specified intent
- **We have the languages for specification**
  - Examples include SVA, PSL, OVL
- **… but the technology does not scale**
  - State explosion
  - Significant advances in the engineering of FPV tools
  - … no hope beyond a point due to complexity barrier

The notion of Design Refinement

How are we able to develop complex designs?

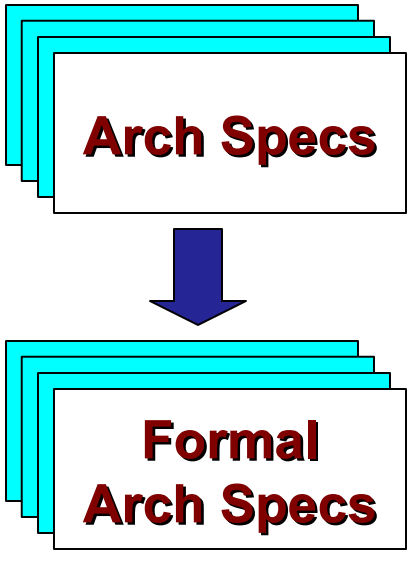
We formulate the design intent

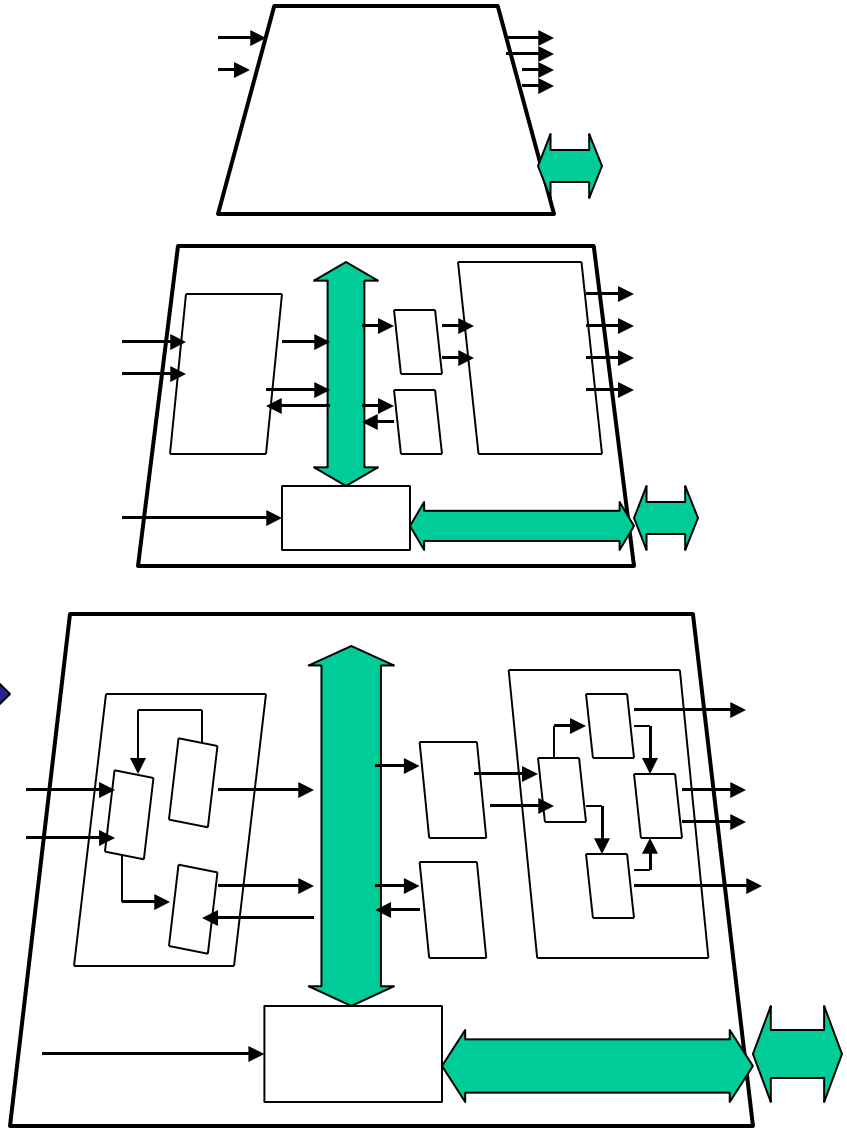
… which is too hard to implement as a single module



Verification of the Design Intent

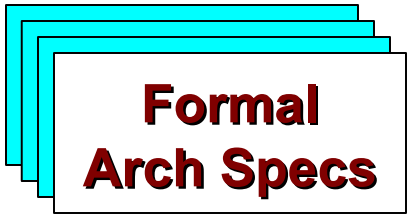
Today we can express the design intent formally …

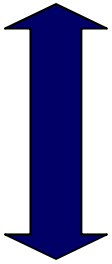




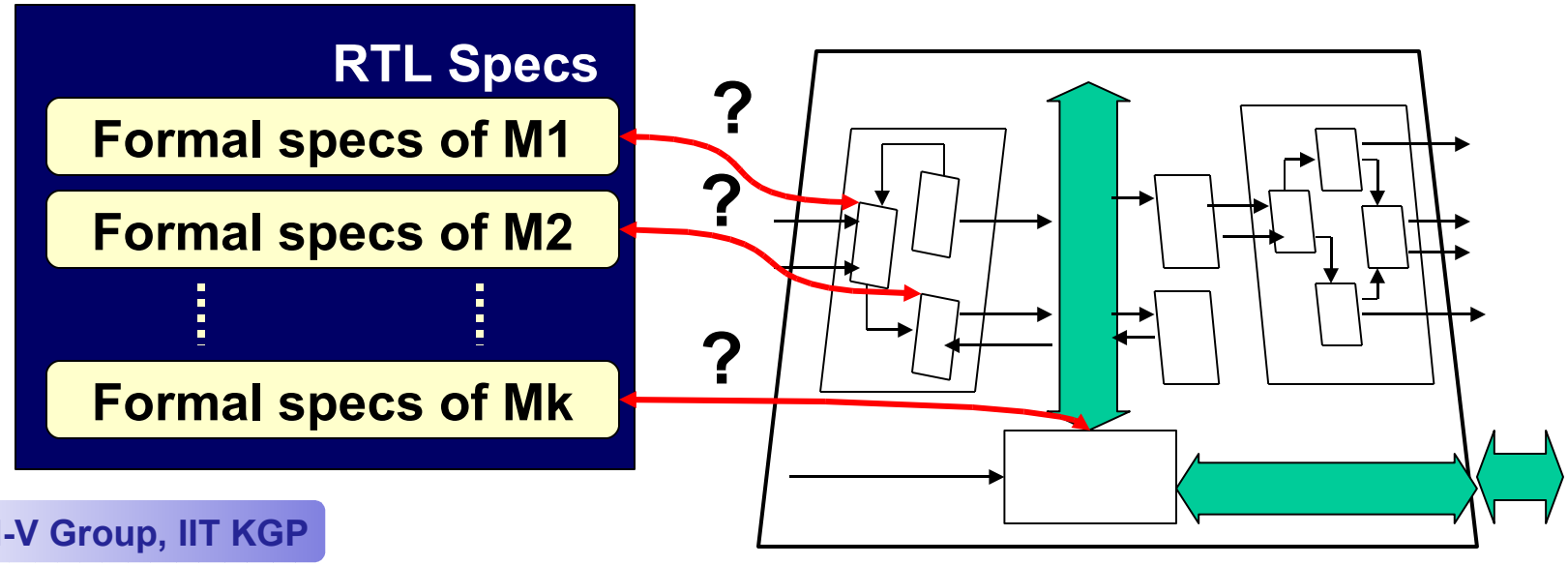
… but we cannot verify formally due to capacity limitations

… so we use FPV locally

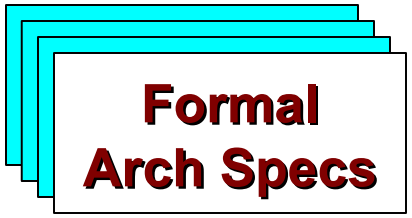


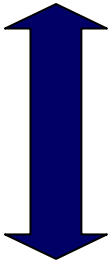


… but this does not guarantee compliance with the Arch. Specs

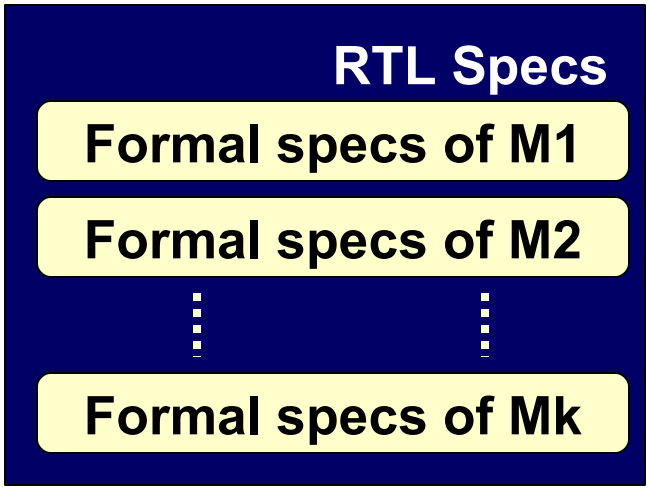


Design Intent Coverage





Does the RTL specs cover the Arch Specs?



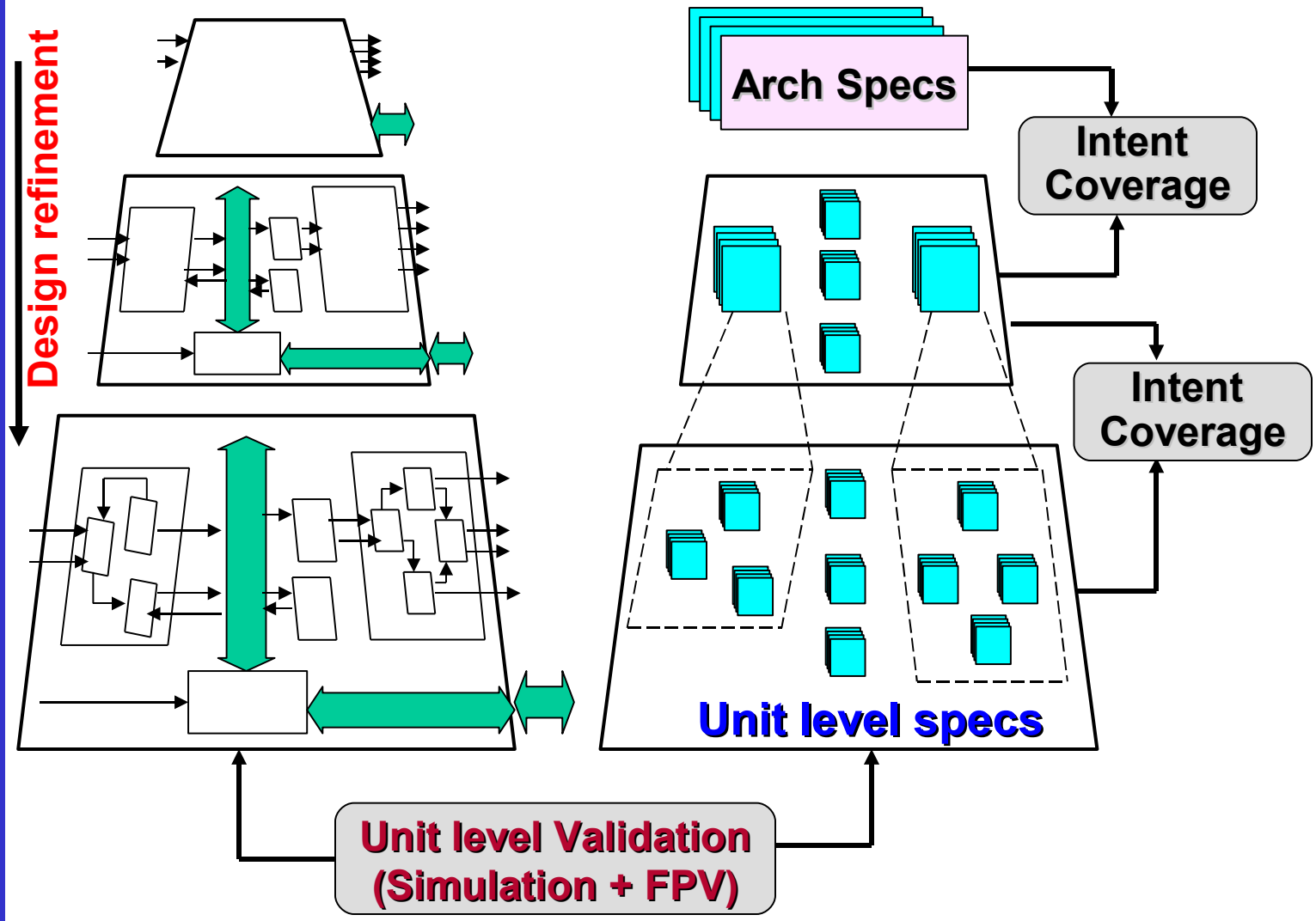
This is not an equivalence checking problem between formal specs

The RTL specs should not admit any run that refutes the Arch Specs

Design Intent Coverage

- **Formal methods for comparing temporal specs at different levels of abstraction**
  - **Formalization and algorithms [DATE'04, ICCAD'04]**
  - **How to present the gap?**
- **The SpecMatcher Tool**
  - **Presents the gap as a set of missing properties**
  - **Main challenge – preservation of syntactic style**
  - **Detailed paper to appear in IEEE TCAD**

The Specification Refinement Flow

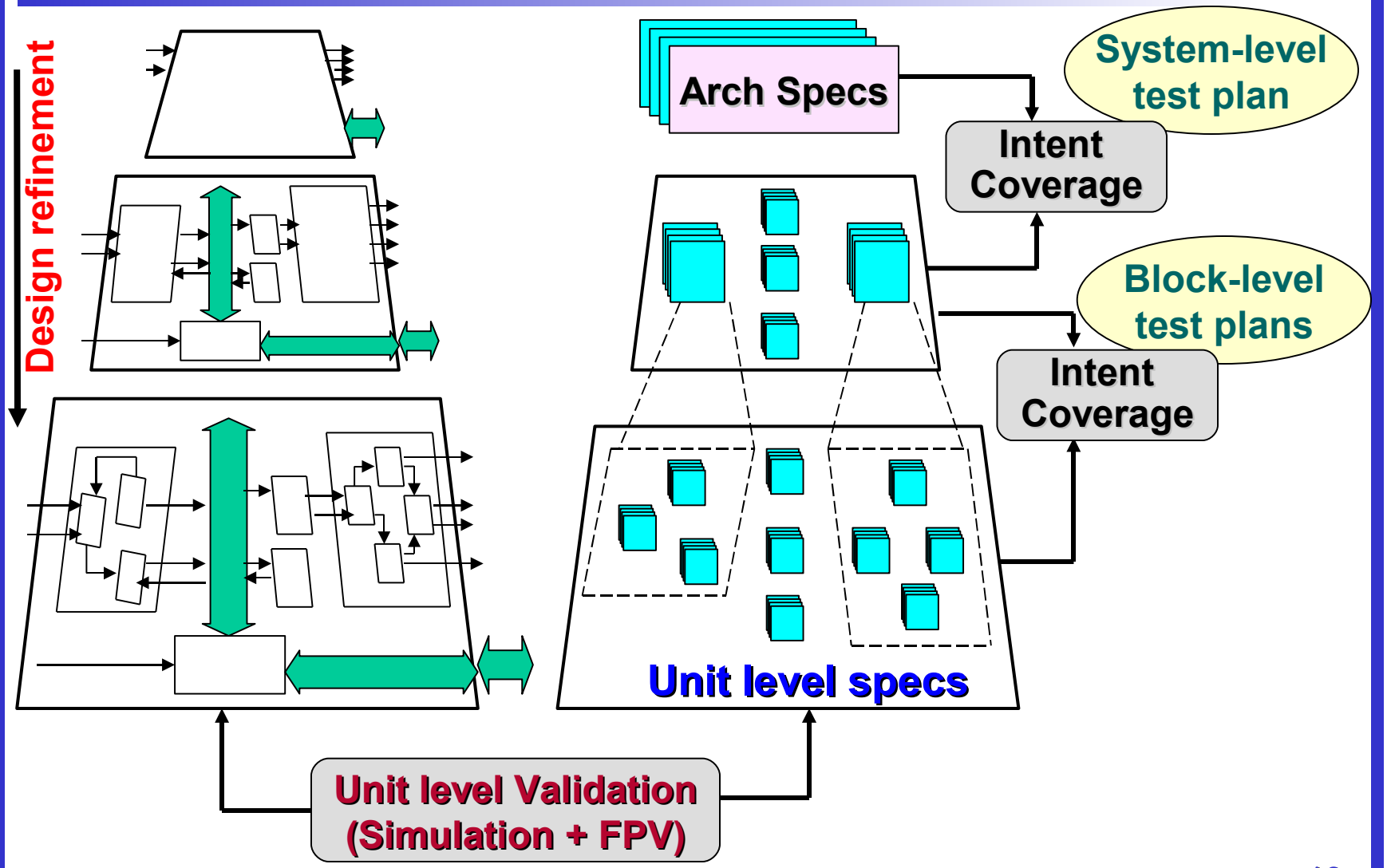


What shall we do with the gap?

- If there is a gap between the original specs and the refined specs, then
  - Intent coverage analysis demonstrates this gap
  - What should the validation engineer do with this gap?
    - Add more properties to close the gap
    - What if no new properties can be conceived?

- Can we use the gap in generating the simulation test plan?
  - Often the gap lies in behaviors corresponding to specific input scenarios
  - *Our goal in this work is to formally find such scenarios and direct simulation towards such coverage points*

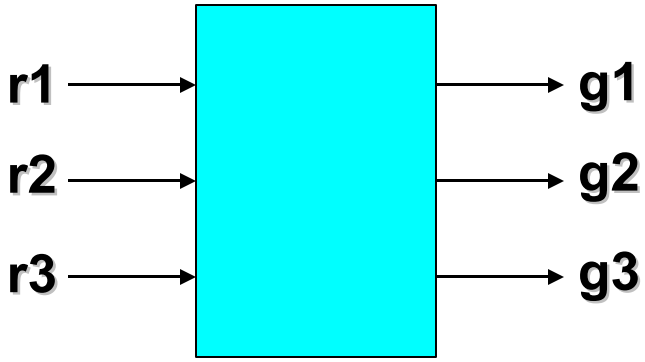
The Specification Refinement Flow



Technical Challenges

- **Finding the scenarios that trigger a given formal property**
  - **Key problem in automatic test generation from properties**
- **Enabling the triggering scenarios during simulation**
  - **Finding the sequence of input constraints and coding them into a constraint random test environment**

A toy example



A: $G(r1 \Rightarrow X\, g1)$

R1: $r1 \wedge \neg\, r3 \Rightarrow X\, g1$

R2: $G(r1 \wedge r2 \Rightarrow X\, g1)$

Given that the DUT satisfies R1 and R2, what are the scenarios that need to be verified in order to guarantee A?

- In the first cycle we should drive $r1 \wedge \neg\, r2 \wedge r3$, because all other relevant cases are covered by R1 and R2
- In the subsequent cycles, the interesting input is $r1 \wedge \neg\, r2$, because the other cases are covered by R2.

Our formal methodology produces the input constraint:

$(r1 \wedge \neg\, r2 \wedge r3) \vee XF(r1 \wedge \neg\, r2)$

Coverage Algorithm

1. Compute $U = A \vee \neg R$

3. If U is not valid then

   (a) Unfold U up to its fixpoint to create two sets of uncovered terms – U_{BF}, the disjunction of terms before fixpoint, and U_{AF}, the disjunction of terms at the fixpoint.

   (b) Eliminate signals belonging to $AP_R – AP_A$ from both using universal elimination

   (c) Eliminate non-inputs from U_{BF} and U_{AF}

   (d) Combine the two – call it I_C

   (e) Return $I_U = \neg I_C$

How do we perform Step 2(c) ?

Elimination of non-inputs

- A property f1 is stronger than a property f2 iff f1 $\Rightarrow$ f2, but the converse is not true

- Given a property, f, defined over input variables, I, and non-inputs, O, we generate a formula S_f over I that is stronger than f.

  - Since S_f is stronger than f, it follows that S_f describes input scenarios that make f vacuously true

  - Therefore we restrict the input space by $\neg S_f$, which covers all non-vacuous runs of f.

  - The rules for strengthening f are presented in the paper

Prototype Implementation

- The algorithms for finding the input constraints are now part of the SpecMatcher tool
  - We have also formalized a methodology for importing the constraints into a constrained random System Verilog test bench
  - Test cases used:
    - Arm AMBA AHB
    - Cache access logic
    - Two Intel test cases

Forthcoming methodology

- **Automatic test generation from formal properties**
  - **Modeling the problem as a game between the test bench and the DUT**
  - **Drives non-vacuous test inputs wrt given property**
  - **Test generator implemented using the Direct-C interface of System Verilog**
  - **Methodology for importing the test generator into a constrained random test bench**

Our home:

http://www.facweb.iitkgp.ernet.in/~pallab/forverif.html

Thank you very much!!