2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION JAPAN

Noise Reduction in Coverage-Based FV

Gilboa Alin, Emilia Katz

intel

accellera SYSTEMS INITIATIVE

Agenda

- Motivation and problem statement
- Stimuli cleanup methodology
- Checker cleanup methodology
- Results
- Summary and next steps

Agenda

- **Motivation and problem statement**
- Stimuli cleanup methodology
- Checker cleanup methodology
- Results
- Summary and next steps

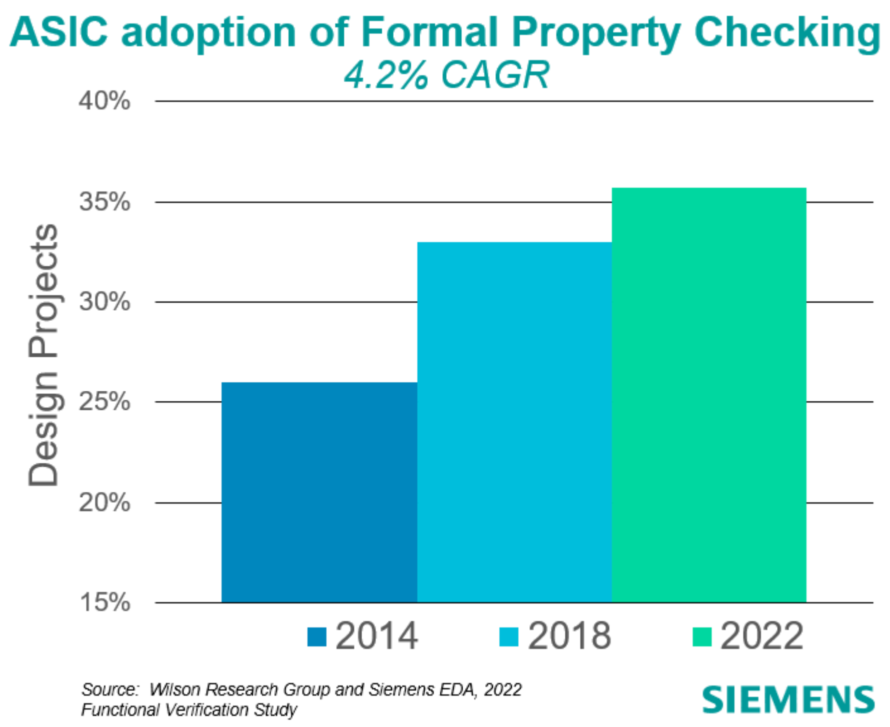
Formal Verification becomes Mainstream

FV usage is increasing

- Harry Foster's report

Impact of FV:

- Early bug finding
- Design exploration
- Deep bug-hunting



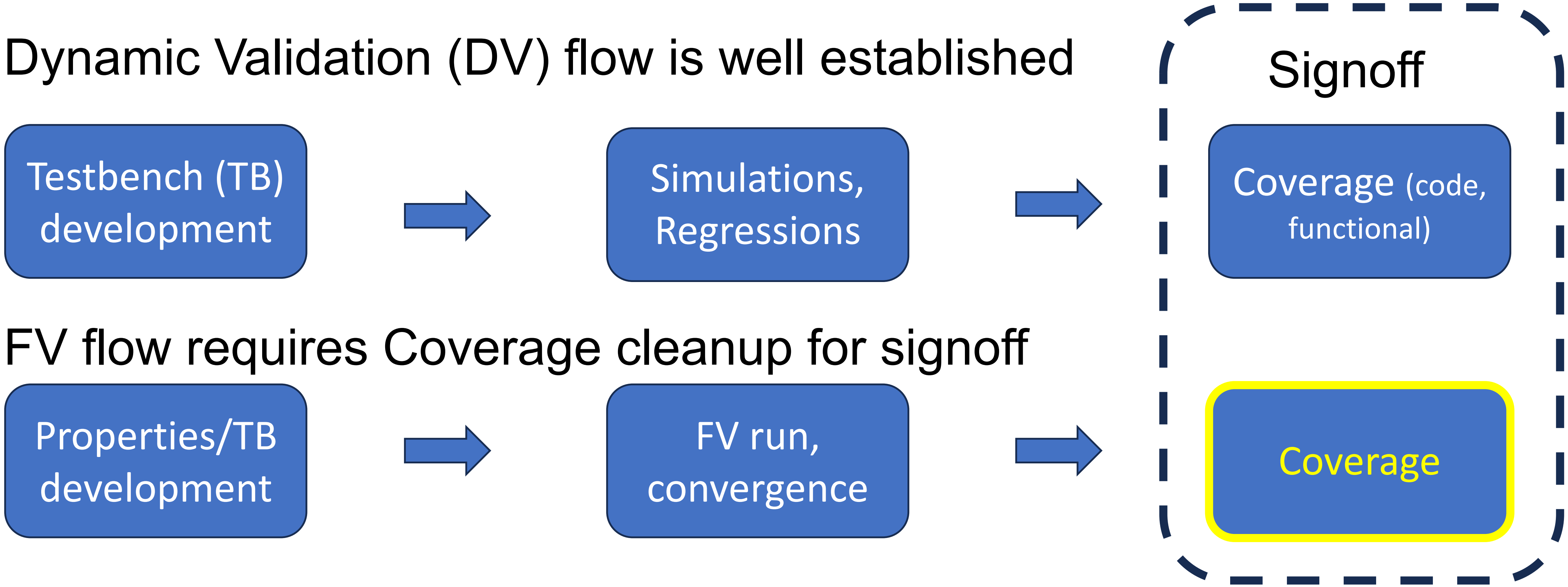
Full responsibility or just an Add-on?

FV Signoff Challenge

Full responsibility |-> Signoff criteria

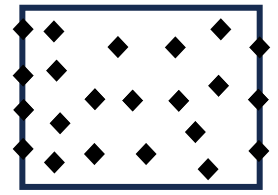
Dynamic Validation (DV) flow is well established

FV flow requires Coverage cleanup for signoff

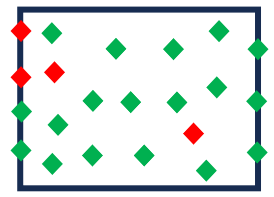

Signoff
Testbench (TB) development
Simulations, Regressions
Coverage (code, functional)
Properties/TB development
FV run, convergence
Coverage

Understanding FV Coverage

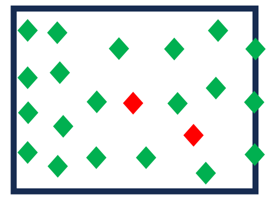
- Coverage can be measured on:
  - inputs/outputs, code statements, branches, expression



- Stimuli coverage: can each cover be covered?
  - Finds overconstraints



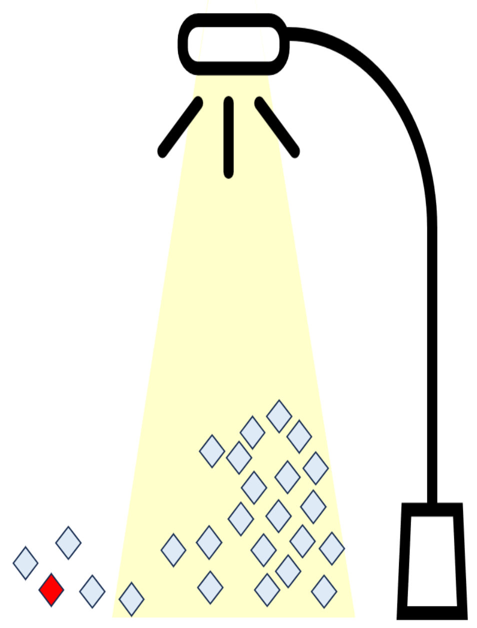
- Checker coverage: verify all behaviors are checked



Best method for design reliability, although not absolute

Coverage Cleanup Issue

- Commercial tools create covers and perform checks
- High volume of covers leads to numerous violations
- Manual inspection of violations:
  - time-consuming and error-prone
- Improper methodology can result in:
  - **Premature termination** of cleanup efforts
  - **'Streetlight effect'** – focusing only on easily visible issues



A robust FV cleanup methodology is essential

Agenda

- Motivation and problem statement
- **Stimuli cleanup methodology**
- Checker cleanup methodology
- Results
- Summary and next steps

Stimuli Cleanup - Traditional

How to efficiently clean thousands or more unreachable covers?

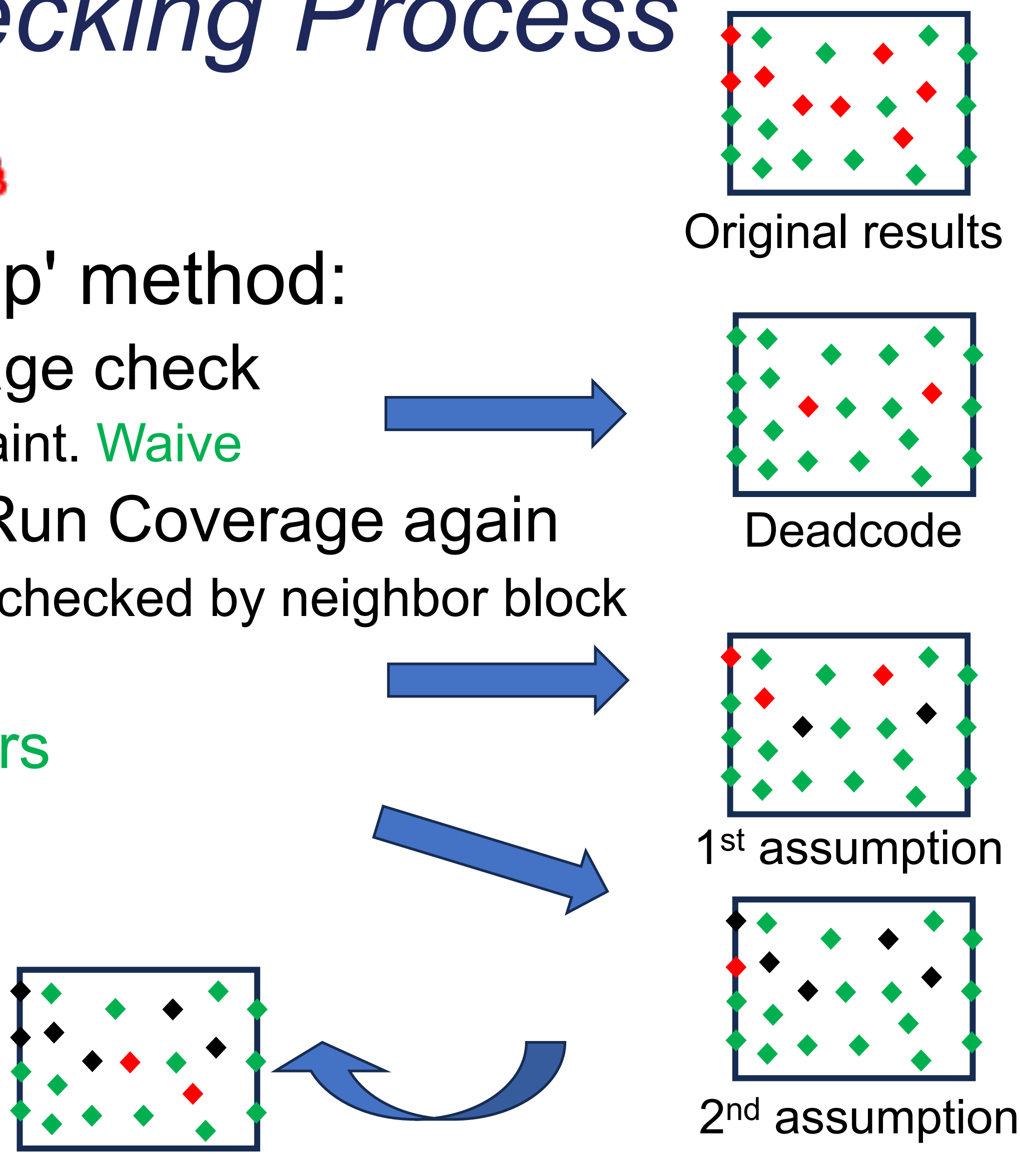
Traditional cleanup method: Covers pattern matching

- Group violations by signal names for collective handling
  - For example, `obsolete_feature`
- Can reduce massively the list of violations
- Remained violations require manual inspection
- Risks errors of waiving wrong covers

- `case1_`**`obsolete_feature`**`_2`
- `if_`**`obsolete_feature`**`_3`
- **`obsolete_feature`**`_replacement_3`
- `temp_cov1`
- `toggle_signal_abc123`

Stimuli Cleanup – *Invert Checking Process*

- Traditional ‘cleaning stimuli failures’
- Adopt the 'stimuli overconstraint cleanup' method:
  1. Remove all assumptions and run Coverage check
     - Unreachable → deadcode and not Overconstraint. Waive
  2. Add assumptions with high confidence. Run Coverage again
     - Assumptions from spec, known restrictions, or checked by neighbor block
     - Unreachable → Waive
  3. Continue adding assumptions and waivers
  4. Remainder, if exists → manual review



Agenda

- Motivation and problem statement
- Stimuli cleanup methodology
- **Checker cleanup methodology**
- Results
- Summary and next steps

Missing Checker Cleanup - COI

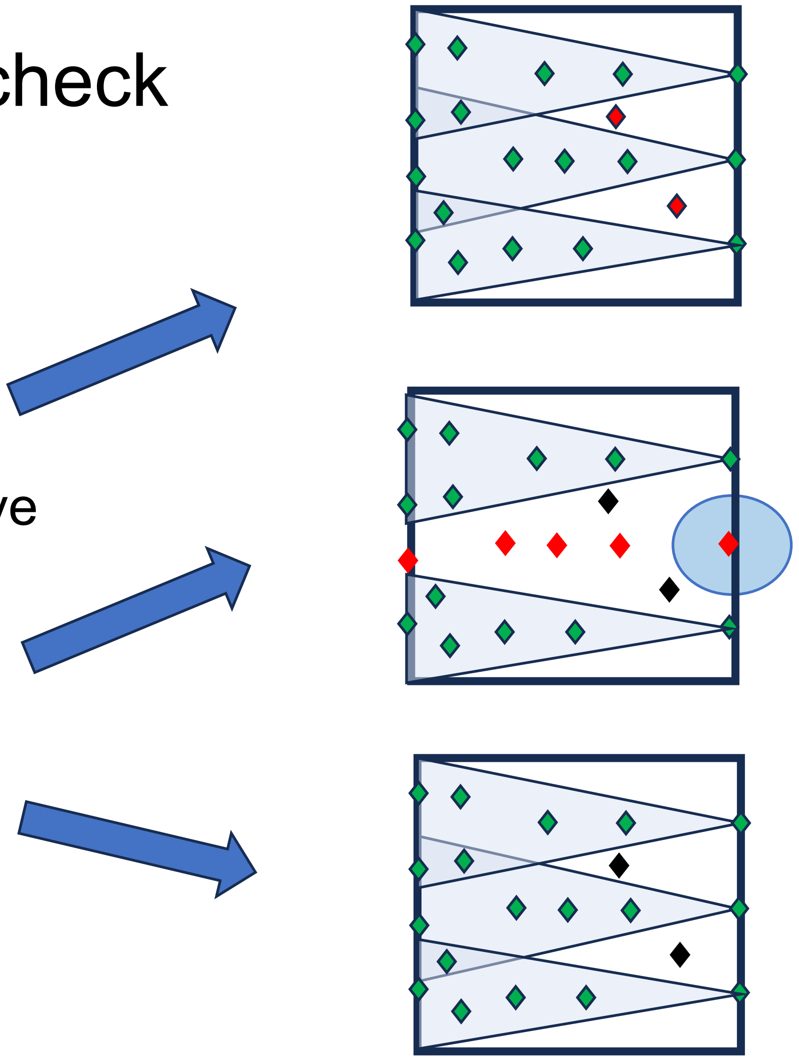
Cone of Influence (COI) is a structural check

- Simple, quick, coarse results

Flow for cleaning COI:

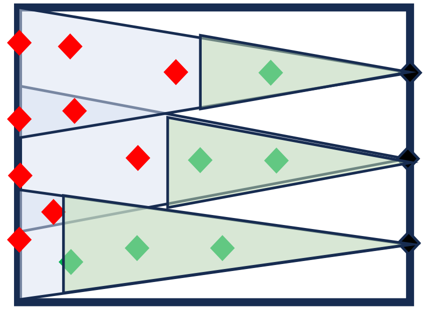
1. Remove dangling logic
   - Add a dummy assertion on each output
   - Run COI check. Out-of-COI → dangling, waive
2. Check out-of-COI outputs
   - Remove dummy assertions, and rerun COI
   - Only outputs are relevant
     - Add assertions for out-of-COI outputs

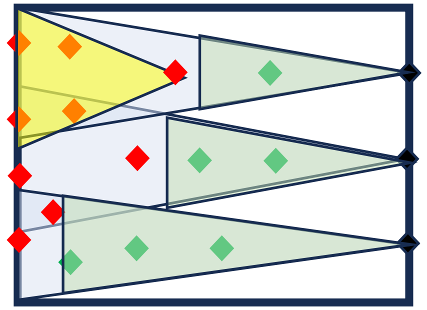
Result: Quick and full cleanup



Missing Checker Cleanup – Proof Core

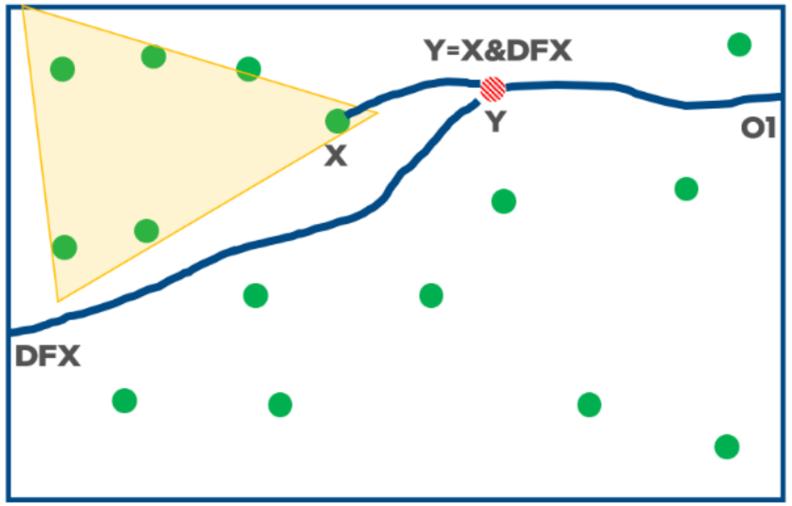
- *Proof core – actual part of an assertion's COI needed for proving it*
  - Runs after FV engine is complete
  - Cleanup involves writing more assertions
  - *How to clean it efficiently?*
- Our methodology:
  - **Prioritize** the cover point with the **largest fanin cone**
    - Adding an assertion here may cover other out-of-proof points
  - Extra care needed for **undetermined assertions**
    - Some cover points may change status when assertion is resolved
  - Address logic unreachable due to **gating**
    - More details in the next slide

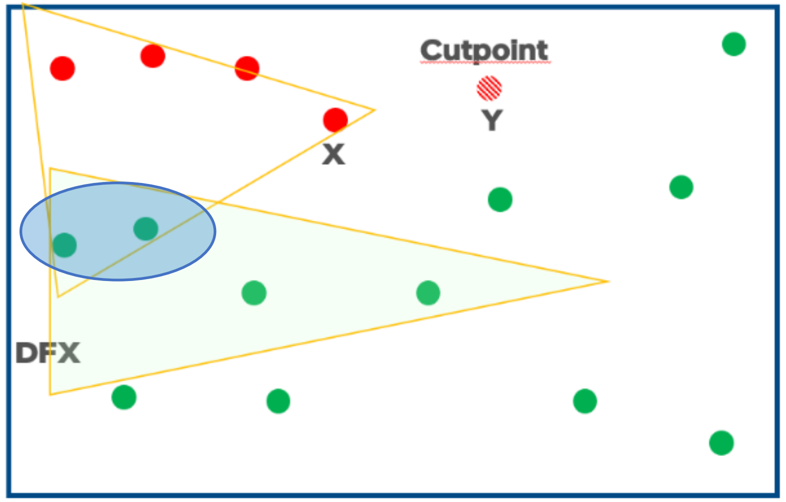




Proof-Core – Gated Logic

- Reachable logic may be gated
  - ‘X’ is reachable. ‘Y’ not, because ‘DFX==0’
  - ‘X’ is part of structural COI
  - ‘X’ is flagged as out of proof-core
    - *How can we know it, and can we waive it?*
- Our methodology:
  - Add cutpoints on unreachable signals (‘Y’)
  - Run COI check again
    - New out-of-COI are those driving only gated logic
      - Waive them
    - Some covers in the cone of ‘X’ are part of other cones
      - Therefore, not waived





Waiving such covers saves a lot of debug time

Agenda

- Motivation and problem statement
- Stimuli cleanup methodology
- Checker cleanup methodology
- **Results**
- Summary and next steps

Results

- Flow applied in several designs and helped finding bugs like:
  - Wrong assumptions
  - Missing or partial assertions
- **Stimuli violations** reduced to **nearly zero** in a **short time**

Design	#cov items	UNR after name-based cleanup	UNR After cleanup	Effort (days)	#bugs found
D1	3k	4%	0.80%	3	1
D2	45k	6%	3%	7	2
D3	3.5k	31%	0.50%	2	0
D4	7.6k	12%	0%	1	1
D5	5.8k	2.30%	0.10%	2	0
D6	2.8k	13.50%	0%	3	1

Results – cont.

- **Checker violations reduced sharply**, but **require more work**
  - Understanding the intent behind internal signals violations

Design	#cov items	Checker violations	Violations after cleanup	Effort (days)	#bugs found
D1	3k	43%	0.90%	7	0
D2	45k	55%	6%	10	0
D3	3.5k	16%	4%	7	1
D4	7.6k	46%	0%	2	2
D5	5.7k	61%	9%	18	1
D6	2.8k	42%	14%	25	1

Agenda

- Motivation and problem statement
- Stimuli cleanup methodology
- Checker cleanup methodology
- Results
- **Summary and next steps**

Summary

- Coverage checks are crucial for verifying FV work is completed
- Lots of data → flows + automation needed
- We were able to achieve clean stimuli and checker using this flow
- Bugs were uncovered and addressed:
  - Through additional assertions
  - By resolving unreachable covers
  - If not detected, **could become escapees**, since FV is the sign-off tool

Coverage has been integrated into the FV signoff process, achieving high-quality cleanup efficiently and within a practical timeframe

Future Work

- Ongoing enhancements to our flows, targeting special cases:
  - Scalability for large design projects
  - Integration with black-box components
  - Efficient merging of various coverage types in proof-core analysis
    - e.g., branch, statement
- Exploring strategies for efficient deployment of Mutation coverage

ありがとうございます

Questions?

Backup