Strategies to Reach the 100% Coverage Goal

Raghavan Ramadoss
Engineering Manager
Cisco

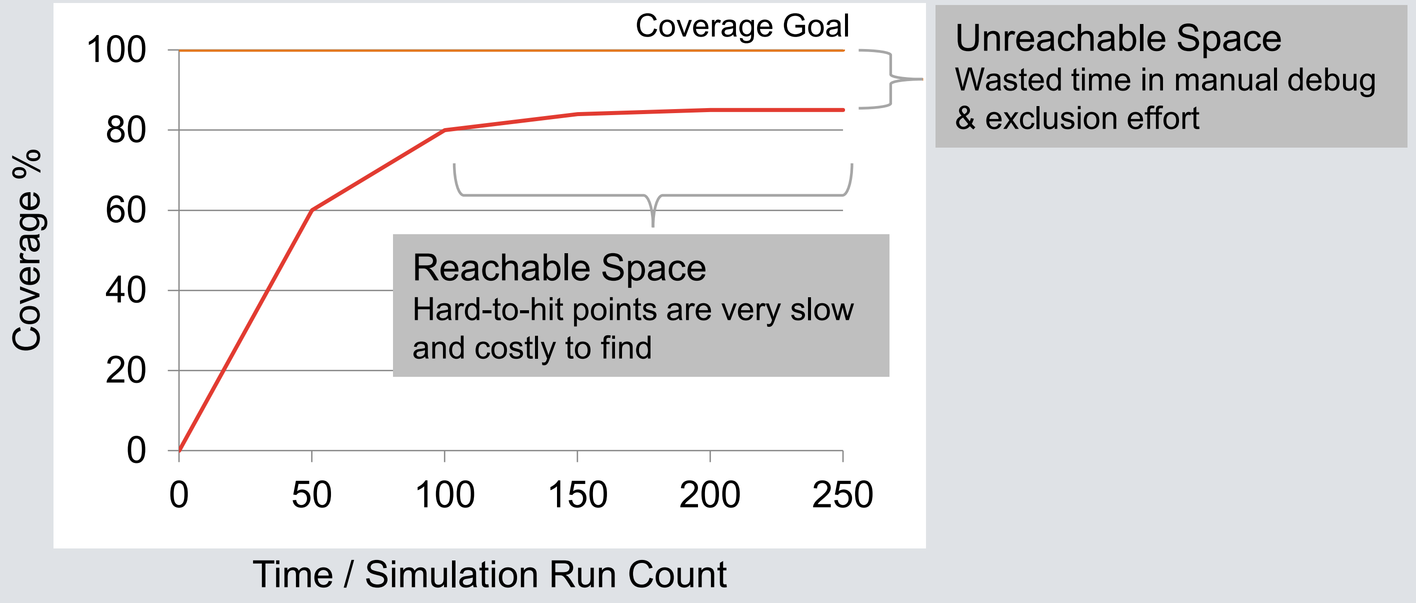
Shravani Balaraju
Applications Engineer
Synopsys

Agenda

- Birds eye view of features used
- Cisco coverage flow
  - Step #1: Selecting/Excluding hierarchies
  - Step #2: Constant identification
  - Step #3: Coverage/Exclusion management
  - Step #4: Unreachability Analysis
  - Step #5: Iterative coverage closure
- Coverage Results with the new flow

Birds eye view of features used

Coverage Closure Challenges



Hard to Hit Coverage Points

- Manually add tests or run

Unreachable Coverage Goal

- Manually debug and exclude

Separate Simulation & Formal Coverage Database

- Manually correlate coverage results

Birds eye view of features used

Feature: Selecting/excluding hierarchies

- Benefit: Improve coverage by excluding targets that never occur/not applicable for the design

Feature: Constant Analysis

- Benefit: Tool identifies and eliminates fixed value variables, configurations at compile time

Feature: Mod tree file

- Benefit: Easier to view
- Coverage is aggregated and reported over a module view rather than instance view

Feature: Unreachability analysis

- Benefit: Automatically find functionally unreachable targets and exclude them from coverage

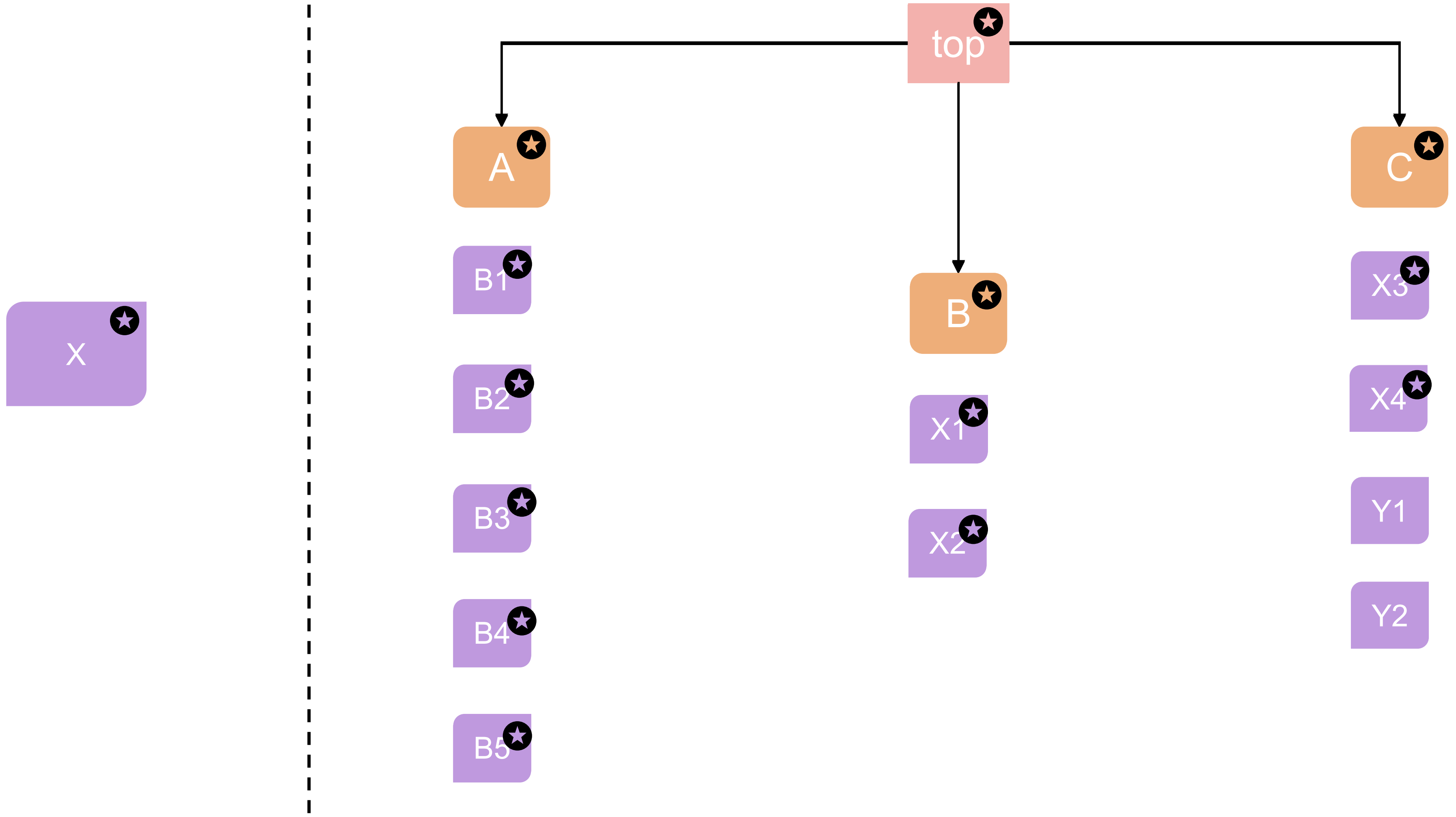
Exclusion using covermeter

- We can pick and choose which scopes to enable code coverage collection on.
- To limit the scope of coverage, '-cm_hier <hier_file.txt>' can be used
- The following can be enabled/disabled using +/- in the hier_file
  - Modules/instances
  - Source code files/file lists
  - Libraries
  - Signals
  - Signal bits/transitions

Constant Analysis

- Automatically eliminates structurally unreachable goals at compile time.
- Implements various methods to improve coverage based on constant propagation
- Let’s consider a signal s, is a constant and is set to ‘0’
  - p && s will always be zero
  - If (s) will never be true
  - This will cause a ripple effect if ‘s’ passes through multiple flops.

Constant Analysis



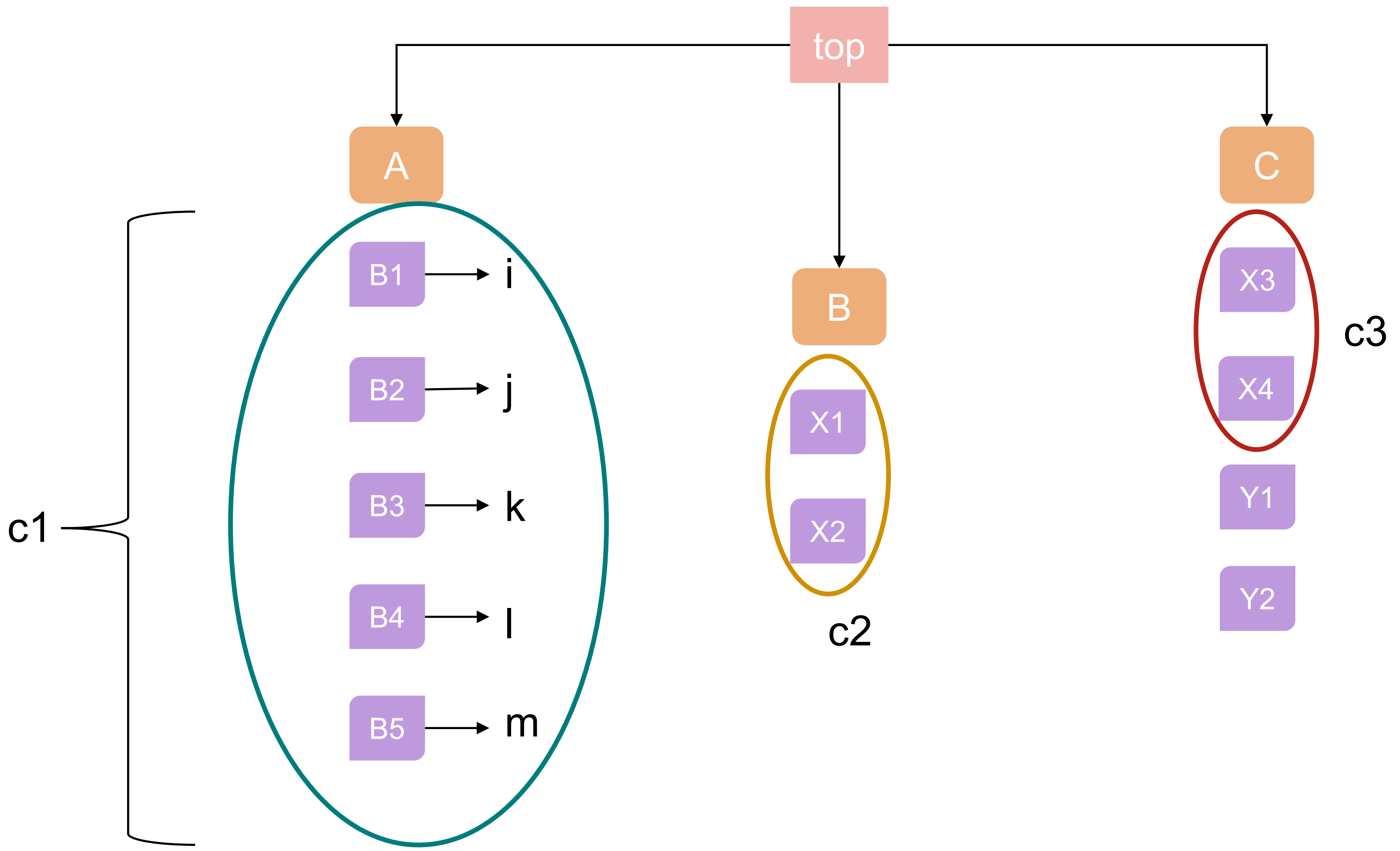
Mod tree file

- The combined score of all instances of a module can be made a part of coverage calculations

- The option ‘-modtreefile’ can be passed to either urg or Verdi during coverage analysis

- Exclusions are propagated to all sibling instances

Mod tree file



UNR

- Find functionally unreachable targets and exclude them from coverage calculations
- Analysis done by formal engines under the hood
- The coverage VDB, previous exclusion files can be parsed to narrow the scope of the analysis region
- Exposes hidden bugs
- Can be deployed at any stage of the design verification cycle

UNR

```
case (state)
          8'h00: d_out <= 8'h01;
          8'h01: d_out <= 8'h02;
          8'h02: d_out <= 8'h04;
          8'h03: d_out <= 8'h08;
          8'h04: d_out <= 8'h80;
          8'h05: d_out <= 8'h70;
```

Unreachable

fvassume –expr {state < 8'h04}

User constraint

UNR

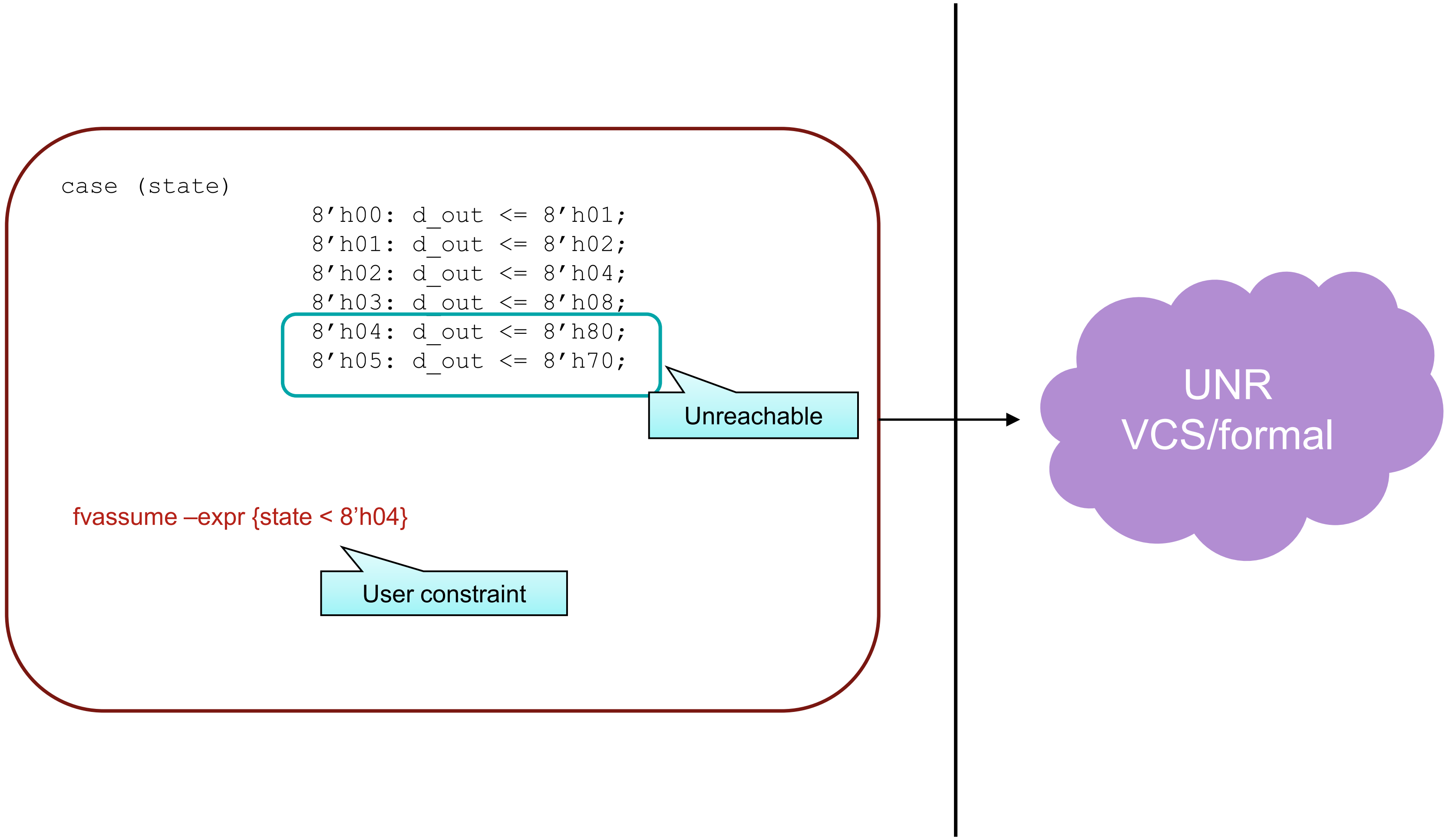
```
case (state)
            8'h00: d_out <= 8'h01;
            8'h01: d_out <= 8'h02;
            8'h02: d_out <= 8'h04;
            8'h03: d_out <= 8'h08;
            8'h04: d_out <= 8'h80;
            8'h05: d_out <= 8'h70;
```

Unreachable

```
fvassume –expr {state < 8'h04}
```

User constraint

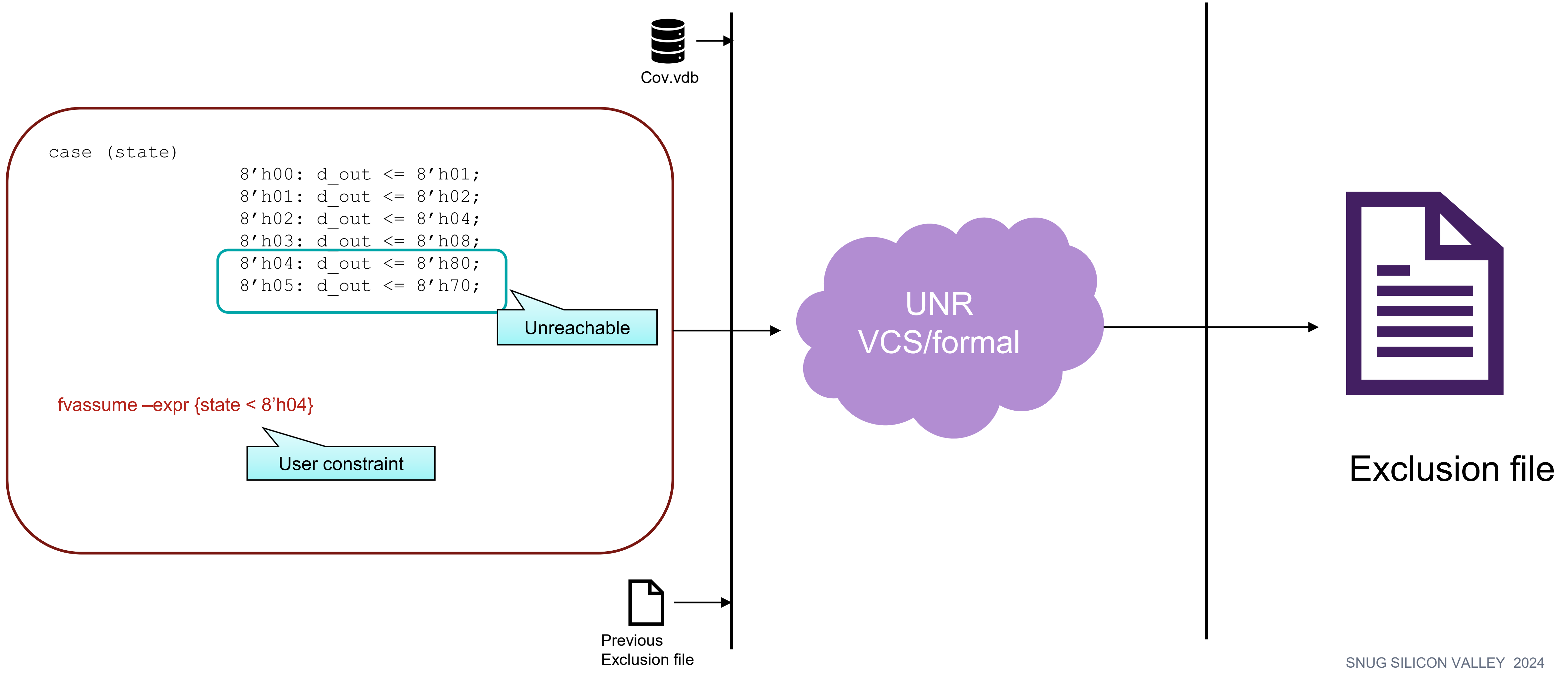
UNR



```
case (state)
                    8'h00: d_out <= 8'h01;
                    8'h01: d_out <= 8'h02;
                    8'h02: d_out <= 8'h04;
                    8'h03: d_out <= 8'h08;
                    8'h04: d_out <= 8'h80;
                    8'h05: d_out <= 8'h70;
```

fvassume –expr {state < 8'h04}

UNR



```
case (state)
            8'h00: d_out <= 8'h01;
            8'h01: d_out <= 8'h02;
            8'h02: d_out <= 8'h04;
            8'h03: d_out <= 8'h08;
            8'h04: d_out <= 8'h80;
            8'h05: d_out <= 8'h70;
```

fvassume –expr {state < 8'h04}

UNR Use Cases

Use Cases	Inputs	Project Cycle	Scope	Purpose
Find Dead Code	RTL	Early – No Testbench	Block	Find unexpected dead code, design bugs
Generate Exclusions	RTL + Coverage DB	Regressions Started	Block/Chip	Find and exclude unreachable goals to reduce manual coverage closure effort
Generate Exclusions	RTL + Cov. DB + Exclusions	Regressions Mature – Closing Coverage	Block/Chip	
Validate Exclusions	RTL + Cov. DB + Exclusions	Late - Closing Coverage	Block/Chip	Validate legacy exclusions

Cisco coverage flow

Step #1: Selecting/Excluding hierarchies
Step #2: Constant identification
Step #3: Coverage/Exclusion management
Step #4: Unreachability Analysis
Step #5: Iterative coverage closure

Step #1: Selecting/Excluding hierarchies

- Hierarchical exclude/include of tree, modules
  - Project wide file identifies reuse modules that are don't cares from coverage perspective
    - e.g. ecc_gen, synchronizers, mbist & dft logic, retimers, memories etc.
  - Block level file identifies subtrees where coverage matters
  - can pick and choose which scopes to enable code coverage collection on.

```
###Project level module exclusions###

-moduletree *synchronizer*
-moduletree *tessent*
-moduletree delay_reg*
-moduletree priority_encoder*
-moduletree rc5*
-moduletree clk_ctrl*
-moduletree *retimer*
-moduletree *mbist*
-moduletree *clock_divider*
```

```
###Block level module exclusions###

+tree mmu_tb.mmu_macro.mmu_mcore.i_mmu_top
+tree mmu_tb.mmu_macro.mmu_mcore.mctl_top
-moduletree LB_CLK_MUX2
-moduletree *cfg_configuration_ring_receiver
-moduletree *cfg_configuration_ring_transmitter
-moduletree *cif_cpu_*
-moduletree *cif_memory_access
-moduletree *cif_register_parsing
-moduletree mmu crc32 8192 xor
```

  - Compile flow merges the two to create a configuration file that's passed to VCS
    - ➢ *vcs …. –cm_hier <config_file>*

Step #2: Constant Analysis

- Identify constants in the design
  - User identified ones provided directly to VCS
    - VCS won’t monitor any code that cannot execute as a result of the constant
  - Automatically eliminates structurally unreachable goals at compile time.
  - Allow tool to parse and explore the design and identify constants
    - One time cost (compile flag); cost depends on design complexity
  - Flow at cisco identifies constants on each compile, as design can continue to evolve
    - At signoff review and checkin the file

➢ vcs …. '-cm_noconst' '-cm_seqnoconst' '-diag' 'noconst' '-cm_constfile’ ‘<constant_file>’

- Challenges: Generated file needs to be reviewed, potential for bugs
  - Solution: Run it after first complete functionality coverage analysis

Step #3: Exclusion management

- Identify instances where coverage can be signed off at module level
  - Upfront identification of identical instances helps save coverage debug time
  - Overall coverage numbers can be updated to reflect module coverage numbers
    - Specify instances to be mapped onto in <file_name>
    - Three instances of module M can be represented with the following entry in <file_name>

```
top.M1, top.M2, top.M3
```

➢ *urg … -modtreefile <file_name>*

- Active exclusions via GUI in Verdi
  - Challenges: Can be laborious and a timedrain
    - Partial Solution: “connect signal” driven exclusion can help reduce some grunt work

Step #4: Unreachability Analysis (UNR)

- Setup and run UNR
  - Uses formal engines to identify dead code
  - No TB dependency, so can be run upfront by the designer
  - Can be run via VC formal or directly with VCS
    - We ran with VCS, so negligible infra investment
- Challenges
  - Can generate huge exclusion files
    - Run after a round of coverage review, so exclusions can be trusted
    - Run by designer upfront, so we can use the DV ramp up time to review/close the list
  - Takes long time to converge
    - Use autoscale and provide more workers to help converge faster

9% Coverage Improvement with UNR

Step #5: Iterative coverage closure

- Two rounds of coverage review
  - First round:
    - We first exclude all error/debug logic, add them to separate exclusion file
    - Review coverage on just the core functionality, so we can focus on areas that really matter
    - Iterate through the process of adding more waivers, adding tests, enhancing testbenches
  - Second round:
    - Unexclude error/debug logic, related waivers
    - Run UNR to assist with exclusion list generation
    - Final signoff as we converge onto near 100%

Results with new coverage flow

Near 100% on most block level environments

- None of the blocks in the previous generation ASIC were even close
- Few exceptions to accommodate ASIC specific configs, flow
- Fast and confident elimination of noise helped team focus on holes that matter
- 9% coverage bump with UNR

Dashboard to capture status/progress

- Automated flow to publish converged results in a central area
- Easy tracking of progress as well as the effort left on each block

THANK YOU

YOUR
INNOVATION
YOUR
COMMUNITY