
2014
ARM®
TechCon™
UBM
Tech

Advanced Verification Management and Coverage Closure Techniques

Nguyen Le, Microsoft

Harsh Patel, Mentor Graphics

Roger Sabbagh, Mentor Graphics

Darron May, Mentor Graphics

Josef Derner, Mentor Graphics

Presenter Information

- Nguyen Le
  - Principal Design Verification Engineer
  - 20+ years experience
  - Email: ngle@microsoft.com
  - Interactive Entertainment Business Unit
  - Microsoft Corp.

About Microsoft SoC Development

- IEB SoC designs
  - Multi-million gate internal IP blocks designed and verified
- Verification flow
  - Constrained-random, coverage driven approach using UVM
  - Testing at IP block and SoC level
  - Testplan requirements tracking
  - Coverage metrics
    - Functional coverage with SV covergroups
    - Assertion coverage with SVA covers
    - Code coverage

      Statement, Branch, Expression, Condition, FSM
- Sign-off requirements
  - All test requirements tracked through to completion
  - 100% functional, assertion and code coverage

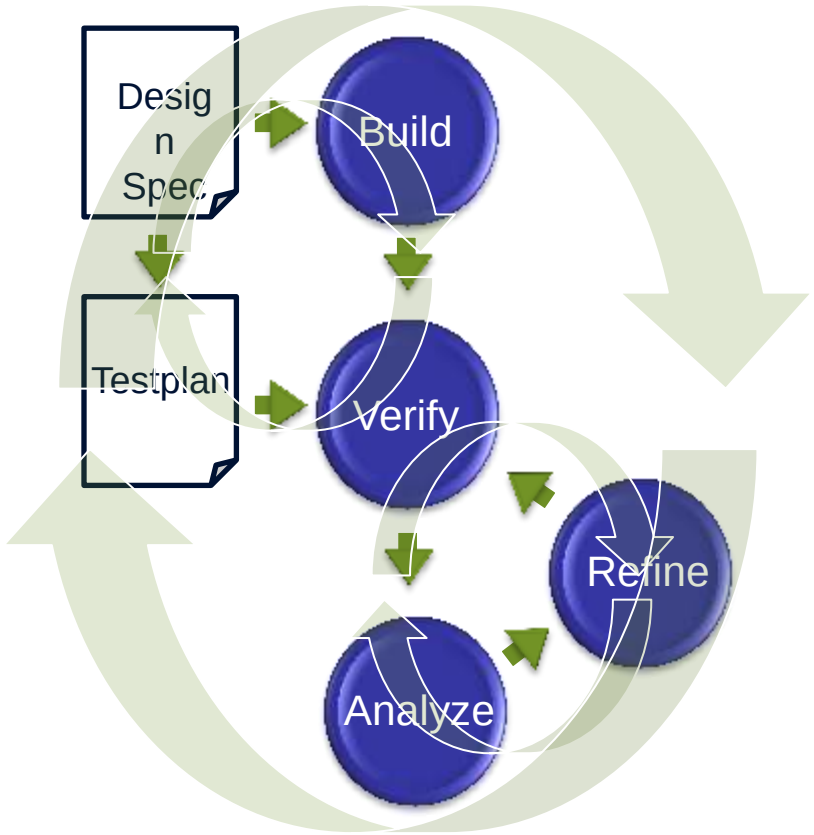
Prior Work at Microsoft

- Adopted UVM
  - Entire DV team trained and deploying UVM
- Used a different regression management tool
  - FCOV is ok
  - Code coverage is very poor
- Property synthesis through simulation trace
  - Prompt designer and DV to review RTL code
- Manual exclusions for unreachable code
  - Add pragmas to exclude unreachable lines
  - Time consuming

Verification Management

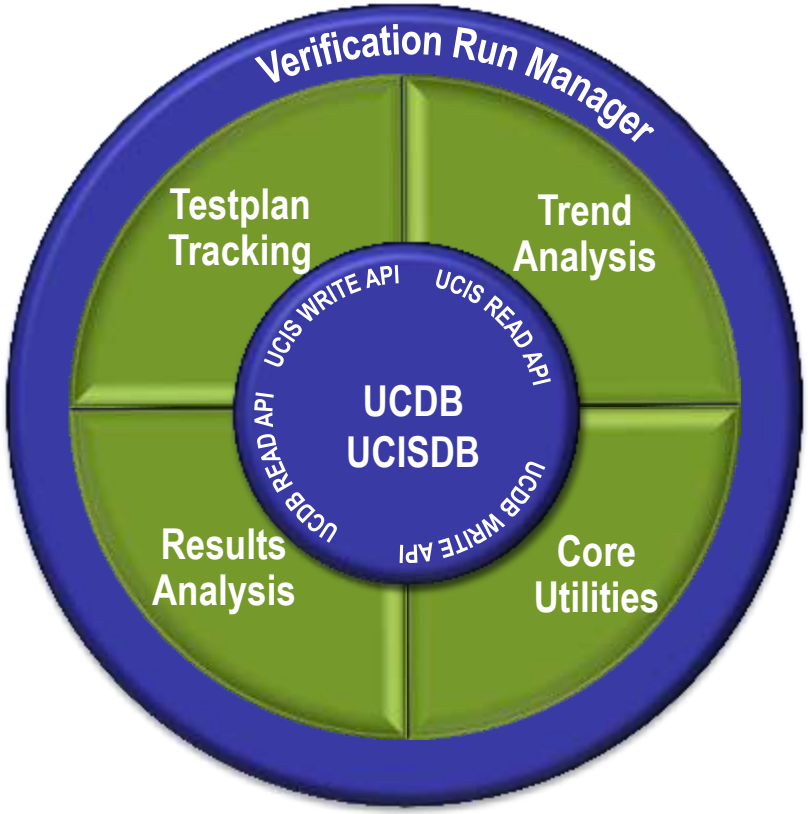
Verification Management Challenges

- Process Management
  - Automated closed loop
  - Full Visibility
  - Turn-around time
- Data Management
  - Data overload, storage capacity
  - Handling complex relationships
  - Immediate & Historical Analysis
- Tool Management
  - Multiple Verification methods
  - Optimize throughput
  - Repeatability, Control & Automation



Questa Verification Management

- Built around the UCDB
  - Store all coverage data and verification process data
- Import verification plan
  - Compare results vs. plan to guide closure
- Results analysis
  - Merge run outputs
  - Triage failures
- Consistent infrastructure
  - Dispersed project teams share data and methods

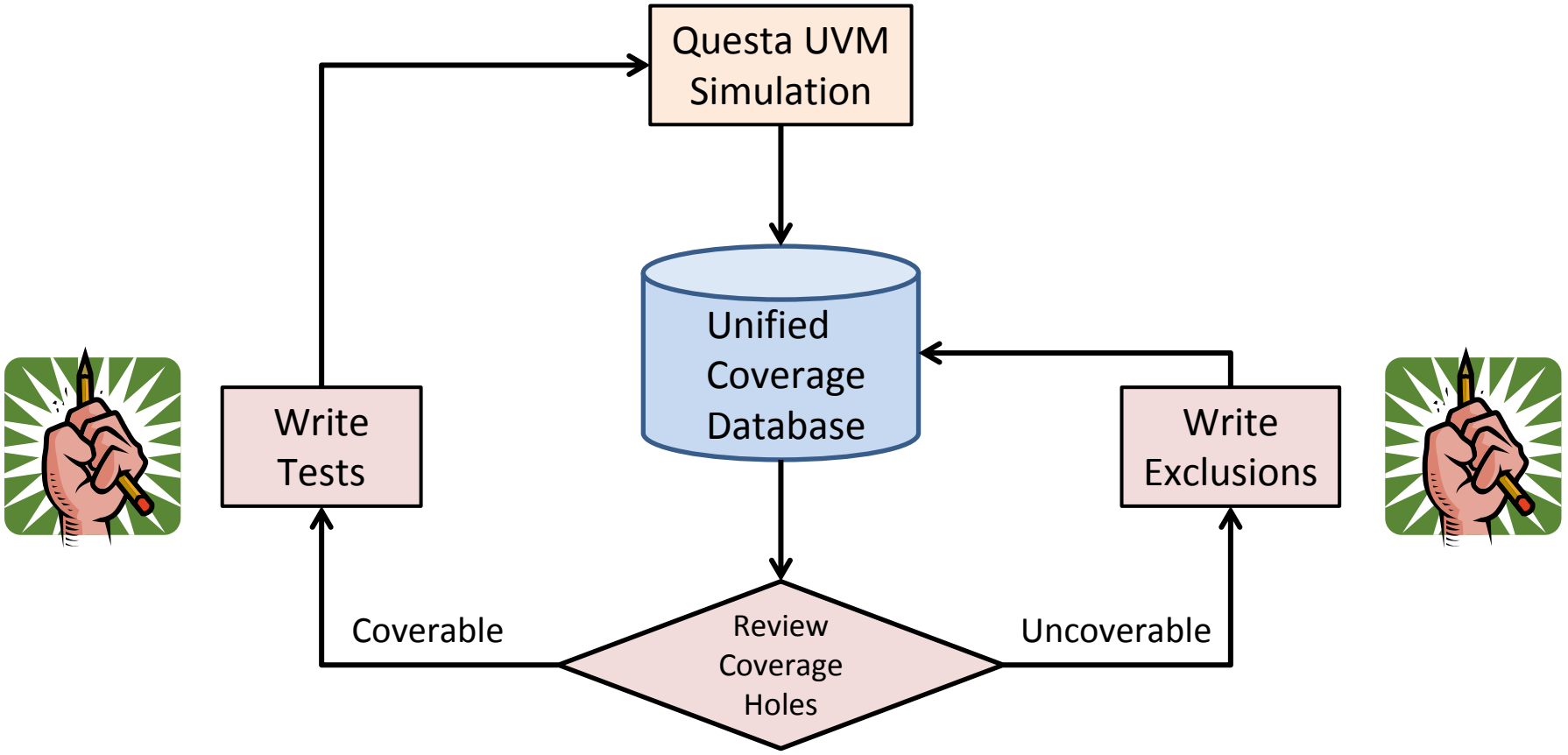


Verification Management Results

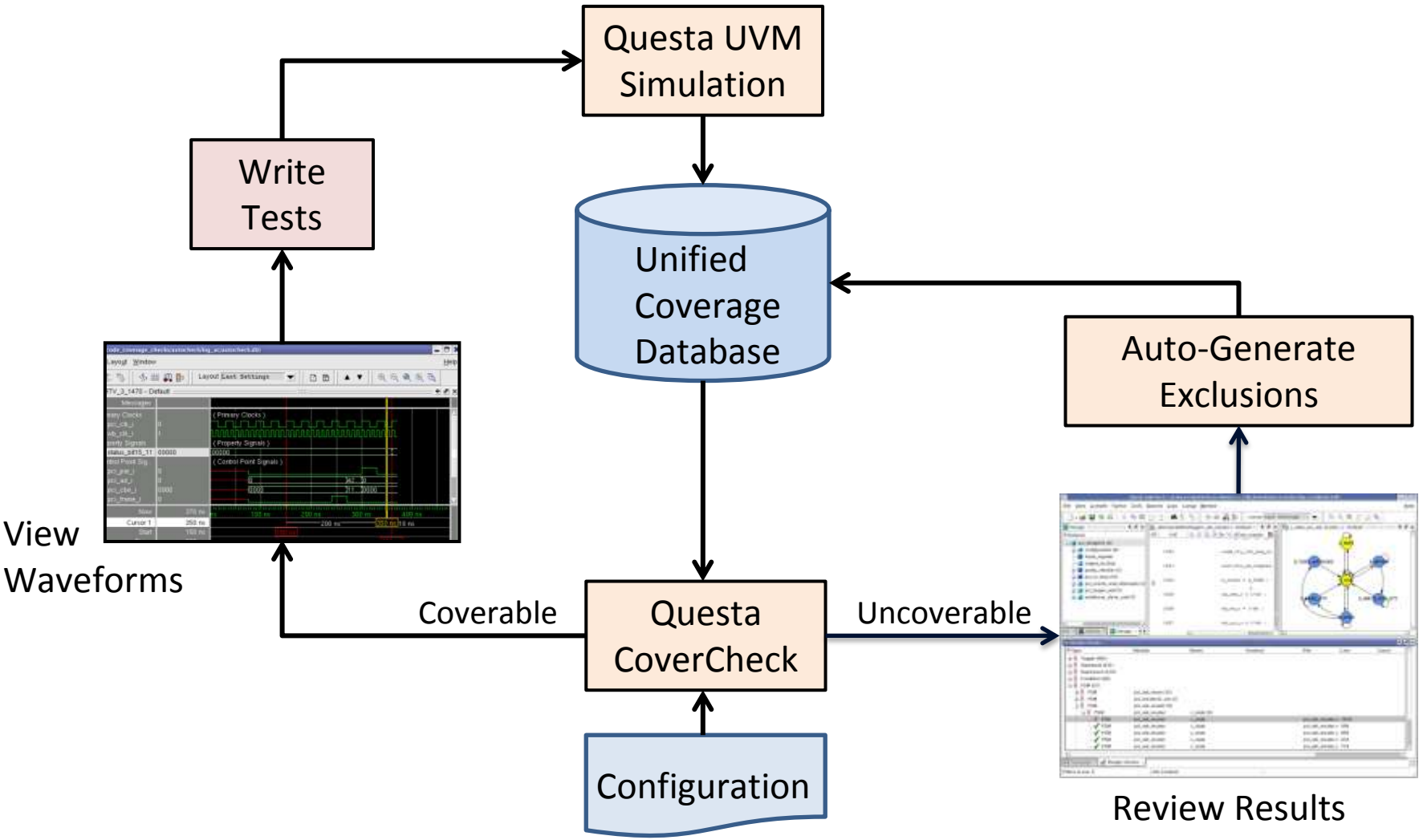
- Measurable benefits
- Reduced runtime variations
  - Random sequences used in coverage driven tests
  - Original random tests varied up to 10X in runtime
  - Better visibility enabled optimization
  - Now they vary by less than **2X**
- Improved regression throughput
  - Ranked and optimized random tests
  - Overall speed up in regression runtime **3X**

Coverage Closure

Coverage Closure Challenges



Questa CoverCheck Automation



Coverage Types

- Code Coverage
  - Automated in simulation
  - Statement, FSM, Branch, Condition, Expression, Toggle
- Functional Coverage
  - Manually implemented coverage model
  - May be automated with assertion generation
  - SV cover directives and covergroups

Where do Coverage Holes Come From?

- Code Coverage
  - Dead code due to RTL coding style
  - Auto-generated code
  - Unused functions in reused IP blocks
- Functional Coverage
  - Over-specified coverage model
  - Misreading spec
  - Incomplete test stimulus

Dead Code Example

- Dead code easily slips into designs
  - Especially with RTL code changes
- Dead code may identify design bugs
  - Highlights different interpretations of design requirements

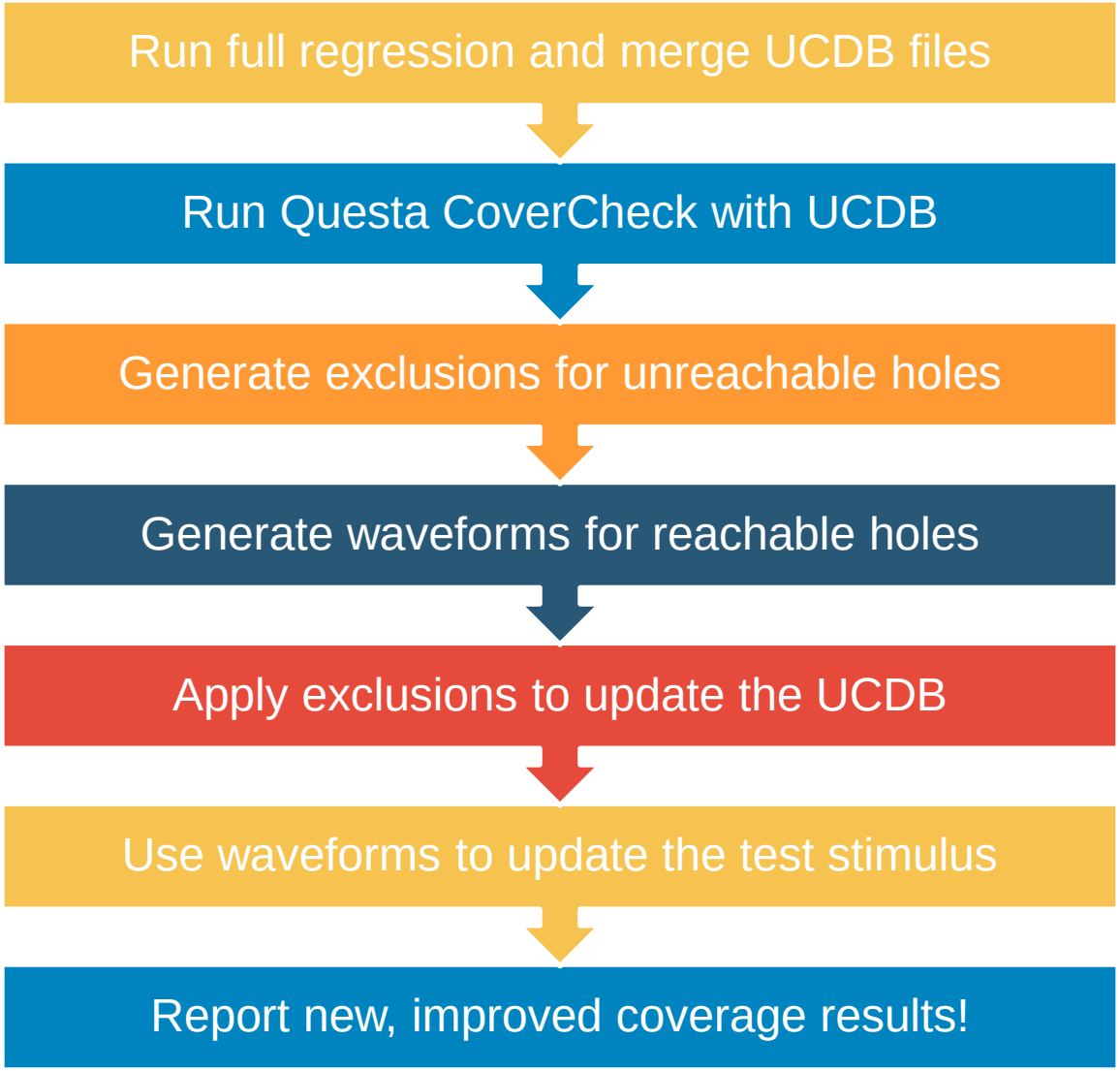
```
reg [1:0] R;
always @* begin
  if (a)       R = 2'b00;
  else if (b)  R = 2'b01;
  else         R = 2'b11;
end
```

R can never be 2'b10

```
reg T;
always @* begin
  T = 1'bX;
  case (R)
  2'b00:       T = 1'b0;
  2'b01:       T = 1'b1;
  2'b10:       T = 1'b1;
  2'b11:       T = 1'b0;
  endcase
end
```

Hence this statement can never be reached

Coverage Improvement Process



CoverCheck Results

- Improved coverage scores
  - Unreachable coverage bins formally proven
  - Auto-generated coverage database exclusions
- Improved code coverage by **10 – 15%** in most hand-coded RTL blocks
- Improvement of up to **20%** for auto-generated RTL code
  - Register blocks contain unreachable simulation hooks
- Overall coverage number improved from **87%** to **97%**

CoverCheck ROI

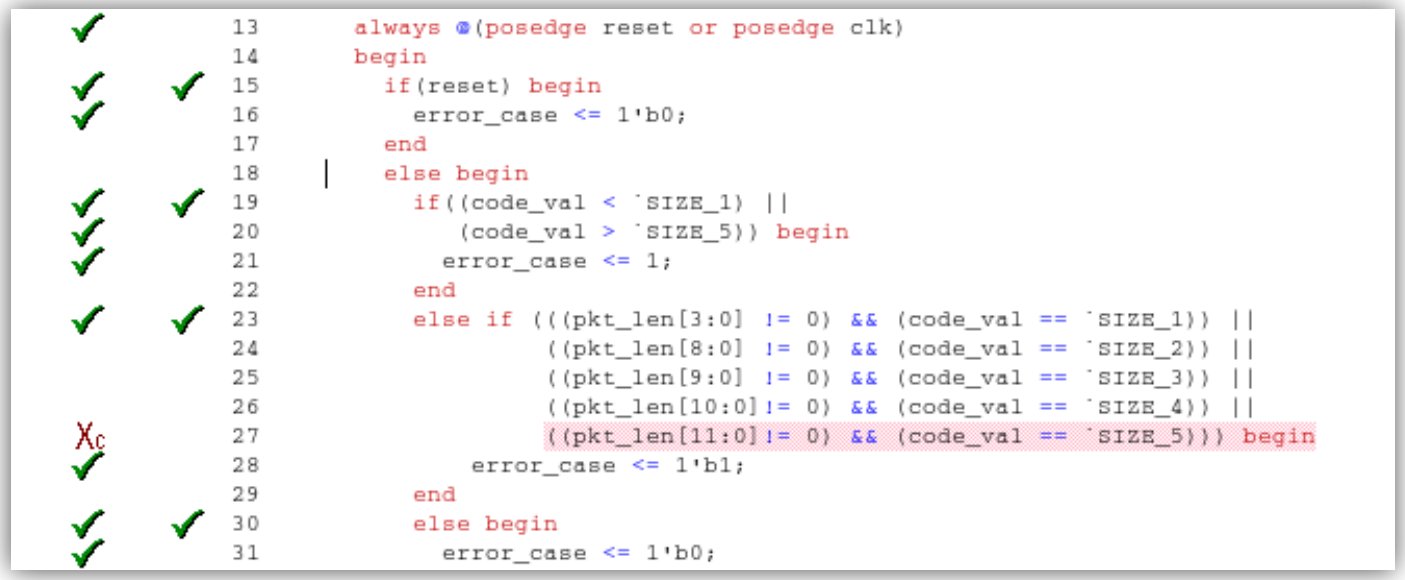
- Estimate of time savings
  - Auto-generated exclusions vs. manual process

Time saved
= (1 design engineer
+ 1 verification engineer)
x 10 min/exclusion
= **4 man months**

- Time savings could be much greater
- Some exclusions could take much longer to generate manually…

Example: Difficult Exclusion

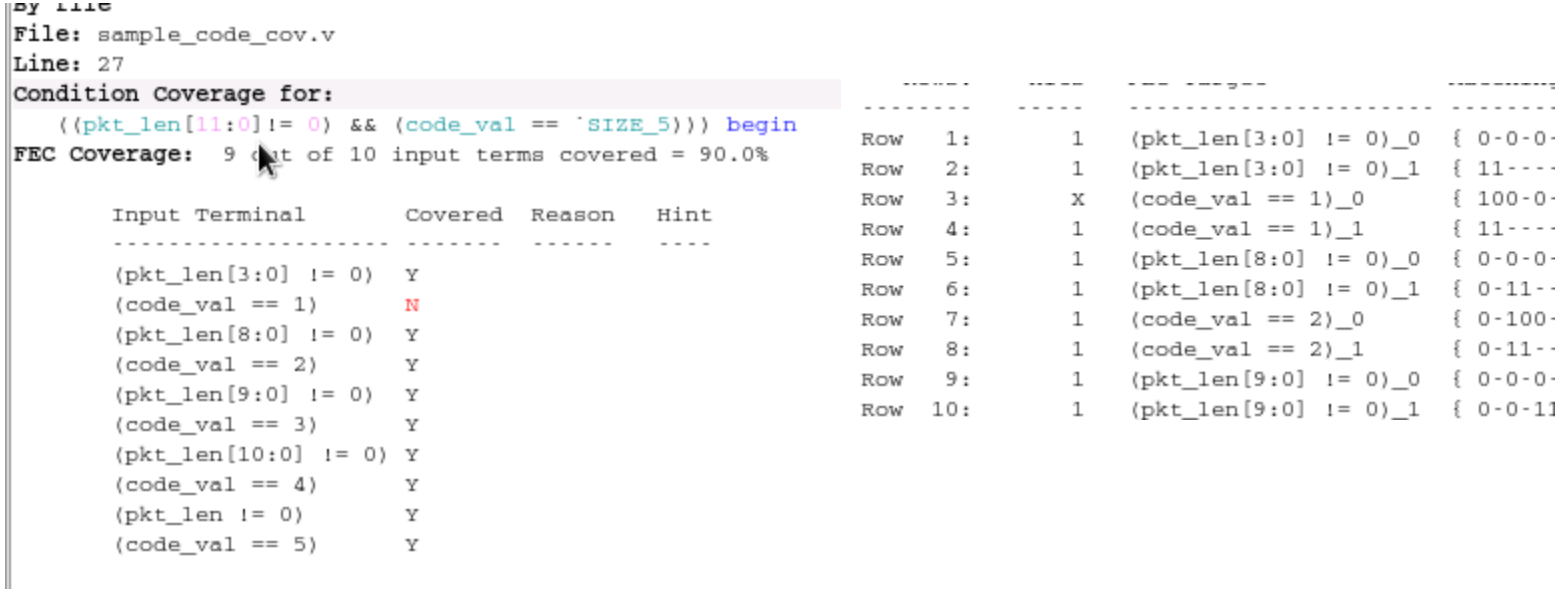
- Took 2 days to prove to ourselves through manual analysis and experimentation
- Last condition in the OR function is impossible to hit when all other terms are false



```
13      always @(posedge reset or posedge clk)
14      begin
15        if(reset) begin
16          error_case <= 1'b0;
17        end
18        else begin
19          if((code_val < `SIZE_1) ||
20             (code_val > `SIZE_5)) begin
21            error_case <= 1;
22          end
23          else if (((pkt_len[3:0] != 0) && (code_val == `SIZE_1)) ||
24                   ((pkt_len[8:0] != 0) && (code_val == `SIZE_2)) ||
25                   ((pkt_len[9:0] != 0) && (code_val == `SIZE_3)) ||
26                   ((pkt_len[10:0]!= 0) && (code_val == `SIZE_4)) ||
27                   ((pkt_len[11:0]!= 0) && (code_val == `SIZE_5))) begin
28              error_case <= 1'b1;
29          end
30          else begin
31            error_case <= 1'b0;
```

Example: Detailed Coverage

- After extracting this snippet of code and run 64k cases (exhaustive), we are convinced of the exclusion from CoverCheck



```
By Line
File: sample_code_cov.v
Line: 27
Condition Coverage for:
   ((pkt_len[11:0]!= 0) && (code_val == `SIZE_5))) begin
FEC Coverage:  9 out of 10 input terms covered = 90.0%

       Input Terminal        Covered  Reason   Hint
       --------------------- -------  ------   ----
       (pkt_len[3:0] != 0)   Y
       (code_val == 1)       N
       (pkt_len[8:0] != 0)   Y
       (code_val == 2)       Y
       (pkt_len[9:0] != 0)   Y
       (code_val == 3)       Y
       (pkt_len[10:0] != 0)  Y
       (code_val == 4)       Y
       (pkt_len != 0)        Y
       (code_val == 5)       Y
```

```
---------  -----  ----------------------  ---------
Row   1:       1  (pkt_len[3:0] != 0)_0   { 0-0-0-
Row   2:       1  (pkt_len[3:0] != 0)_1   { 11----
Row   3:       X  (code_val == 1)_0       { 100-0-
Row   4:       1  (code_val == 1)_1       { 11----
Row   5:       1  (pkt_len[8:0] != 0)_0   { 0-0-0-
Row   6:       1  (pkt_len[8:0] != 0)_1   { 0-11--
Row   7:       1  (code_val == 2)_0       { 0-100-
Row   8:       1  (code_val == 2)_1       { 0-11--
Row   9:       1  (pkt_len[9:0] != 0)_0   { 0-0-0-
Row  10:       1  (pkt_len[9:0] != 0)_1   { 0-0-11
```

- Unreachable code_val == 1 never false

Summary

Conclusions

- Verification of complex SoC projects is a complex process to manage
- Automation of verification management improves visibility into the regression process to allow for throughput optimization
- Time saved by automatic code coverage closure is easily an order of magnitude


2014
ARM®
TechCon™
UBM
Tech