Expediting the Code Coverage Closure Using Static Formal Techniques – A Proven Approach at Block and SoC Levels

Questa Formal team and friends

2015 DVCon India
D1A2.1-DV
Agenda

■ Introduction
■ Coverage Backgrounder
■ Targeting Unreachable Coverage with Formal
■ Reaching Coverage Closure Faster
■ Conclusion
Functional Verification Challenges

Coverage Ranks at the Top of Project Management’s Concerns

**Most Important Concerns**

- Creating Sufficient Tests to Verify the Design (Coverage Closure): 32%
- Defining Appropriate Coverage Metrics: 14%
- Knowing Current Verification Coverage: 12%
- Managing the Verification Process: 15%
- Time to Isolate and Resolve a Bug: 15%
- Time to Discover the Next Bug: 9%
- Other: 3%

Industry Trends: ASIC D&V Use of Code Coverage

FPGA Verification Technique Trends

Code Coverage Challenge

Are we there yet?

Data & graph from 2011 customer paper

- 270 man weeks to do coverage waiver analysis for one design
- 180 man weeks to write missing tests
- That’s almost 9 man-years!
Verification Management Challenge

What’s been covered?

What needs to be covered?

How long before we are done?

How can I improve on my processes?
What You Will Learn Today

- Primer on coverage types and how coverage is recorded
- How to rapidly identify “unreachable” coverage areas
- How to reach your coverage goals faster
Agenda

- Introduction
- Coverage Backgrounder
- Targeting Unreachable Coverage with Formal
- Reaching Coverage Closure Faster
- Conclusion
Types of Coverage

- **Origin of Source**
  - Specification
  - Implementation

- **Method of Creation**
  - Explicit
  - Implicit

- **Both are Required**
  - Functional
  - Structural
Coverage: Implementation

What areas of the design have been touched by verification

- **Code Coverage**
  - Did these lines/branches of code get exercised?
  - Automated in the simulation environment
  - One of the basic design verification signoff metrics
  - A basic measure with little correlation to functionality

- **FSM Coverage**
  - Did all the states and transitions get exercised?
  - Automated in the simulation environment
  - One of the basic design verification signoff metrics
  - Typically included with code coverage
Code Coverage: Statement (s)

- Counts the execution of each statement on a line — Even if multiple statements

- Example:

```verilog
always @(posedge clk or negedge rstn)
...
reg <= dat;
...
C <= A && B;
```

- Report style based on number of Statements

<table>
<thead>
<tr>
<th>Enabled Coverage</th>
<th>Active</th>
<th>Hits</th>
<th>Misses</th>
<th>% Covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stmts</td>
<td>415</td>
<td>387</td>
<td>28</td>
<td>93.2</td>
</tr>
</tbody>
</table>
Code Coverage: Branch (b)

- Counts the execution of each conditional “if/then/else” and case statement
  - All true and false branches are considered
  - Each (if/else if/else | case) element counts as a branch

- Example (if statement):
  ```
  if (!rstn)
      q <= 1'b0;
  else
      q <= d;
  ```

- Report style based on number of Branches

<table>
<thead>
<tr>
<th>Enabled Coverage</th>
<th>Active</th>
<th>Hits</th>
<th>Misses</th>
<th>% Covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branches</td>
<td>47</td>
<td>45</td>
<td>2</td>
<td>95.7</td>
</tr>
</tbody>
</table>
Code Coverage: Condition (c)

- Analyzes the decision made in “if” and ternary statements — Considered extension of branch coverage

- Example:

```
if (ce && we)
```

| 1 | 0/1 |

- Report style based on Focused Expression Coverage

<table>
<thead>
<tr>
<th>Enabled Coverage</th>
<th>Active</th>
<th>Hits</th>
<th>Misses</th>
<th>% Covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEC Condition Terms</td>
<td>16</td>
<td>13</td>
<td>3</td>
<td>81.2</td>
</tr>
</tbody>
</table>

All FEC conditions must be hit: 
ce = 0,1; we = 0,1

ce is uncovered: 
Never hit 0
**Code Coverage: Expression (e)**

- Analyzes expressions on the right hand side of an assignment
- Example:
  
  ```
  wire C = A && B
  
  1 0/1
  ```

- Report style based on Focused Expression Coverage

<table>
<thead>
<tr>
<th>Enabled Coverage</th>
<th>Active</th>
<th>Hits</th>
<th>Misses</th>
<th>% Covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEC Condition Terms</td>
<td>25</td>
<td>14</td>
<td>11</td>
<td>56.0</td>
</tr>
</tbody>
</table>

All FEC conditions must be hit:

A = 0,1; B = 0,1

A is uncovered:

Never hit 0
Code Coverage: Toggle (t)

- Counts each time a logic node transitions one state to another

- Example:

```vhdl
reg FF_A;
always @(posedge clk)
    FF_A <= din;
```

- Report style based on Toggle Bins

<table>
<thead>
<tr>
<th>Enabled Coverage</th>
<th>Active</th>
<th>Hits</th>
<th>Misses</th>
<th>% Covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toggle Bins</td>
<td>356</td>
<td>351</td>
<td>5</td>
<td>98.5</td>
</tr>
</tbody>
</table>

To be covered:

`FF_A` must toggle:

0 to 1 and 1 to 0
Code Coverage: FSM (f)

- Counts the states and transitions of a FSM

- Example:
  
  FSM States: **s1; s2; s3**
  FSM Transitions: **s1 -> s1; s1 -> s2; s2 -> s3; s2 -> s1; s3 -> s1**

- Report style based on FSM States and Transitions

<table>
<thead>
<tr>
<th>Enabled Coverage</th>
<th>Active</th>
<th>Hits</th>
<th>Misses</th>
<th>% Covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>States</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>100.0</td>
</tr>
<tr>
<td>Transitions</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>80.0</td>
</tr>
</tbody>
</table>

*All States and Transitions must be hit*

*This Transition not exercised (uncovered)*
Coverage: Structural

How much verification has stressed the design

- **Assertion Coverage**
  - How many times did the assertion get evaluated, pass, fail
  - Automated in the simulation environment
  - **Doesn’t answer the questions:**
    - Is the assertion implemented correctly? (check anything of value)
    - Are there enough assertions?

- **Structural Coverage**
  - Measures corner case type activity
    - How many times was my FIFO empty, full, hit high water mark
  - Implementation specific, can be automated with assertions
  - How well is the TB environment stressing the design?
Cover Statements/Properties

- Properties and Sequences can be “covered”
- Useful for checking temporal behavior of your design
  - SVA/PSL designed for describing temporal behavior
  - Cover statements typically target a sequence of events
    - Can also target single cycle events
    - Simulation will count the number of occurrences
    - Formal will tell you if it’s reachable or unreachable

Examples:

```verilog
cov_sm_trans: cover property (@(posedge clk) cstate == TRANS);
cov_ddr_wr: cover property (@(posedge clk) ddr_act ##[1:20] ddr_wr);
sequence apb_wr;
    pselx && pwrite && !penable ##1 pselx && pwrite && penable;
endsequence
cov_b2b_wr: cover property (@(posedge clk) apb_wr ##1 apb_wr);
cov_seq: cover property (@(posedge clk) a ##2 b ##[1:3] c[*4] ##1 a );
```
Coverage: Functional
What features of the design have been tested

- Transactional Coverage
  - Measures interface type transactions
    - Have I covered all my AHB/AXI transactions?
  - Typically implemented with cover groups/points
  - Often used with complex TB environments (TLA, CR, iTBA)

- Functional Coverage
  - Measures occurrence of functional events
    - Did my design do back-to-back writes?
  - Typically implemented with cover groups and cover directives
  - Used in complex TB environments, correlate function to spec
Functional Coverage

- Must be specified by the user and cannot be automatically inferred from design

- Validates actual functionality

- Formal specification of verification plan
  - Direct correlation between requirements and verification

- Measures verification completeness against specification
  - Have I verified all functional requirements?
  - Have I covered the entire verification plan?
  - Are my tests adding values to my verification goal?
  - Have I exercised all corner cases in my design?
  - Am I done?

- Counts how many times “interesting” things occur
CoverGroups

- System Verilog CoverGroups
  - coverpoints and coverbins used to categorize/display data
  - Must be instantiated

- Example: CG in module

```verilog
module cover_pci_master32_sm (input clk_in, input [3:0] cur_state);
covergroup cg_cur_state @ (posedge clk_in);
  cp: coverpoint cur_state {
    bins s_idle = {1};
    bins s_addr = {2};
    bins s_tran = {4};
    bins s_end = {8};
  }
endgroup : cg_cur_state;

cg_cur_state cg_cur_state_inst = new;
endmodule
```

Are coverbins reachable?
CoverGroups Example: FSM, Arbiters

covergroup cg_cstate @ (posedge clk);
    cp: coverpoint cstate {
        bins s_valid [5] = {1,2,4,8,16};
        bins s_illegal = {0,3,5,6,7,[9:15]};
    }
endgroup : cg_cstate;

wire [1:0] enables = {wr_en,rd_en};
wire en = $changed(enables);
reg len;
always @*
if (!clk) len <= en;
wire gclk = clk & len;
covergroup cg_enables @ (posedge gclk);
    cp: coverpoint enables {
        bins reads = {1};
        bins writes = {2};
        illegal_bins bad = {3};
        bins idle = default; }
endgroup : cg_enables;
cg_enables cg_enables_inst = new;
Coverage: Metrics

- **Basic: Code/FSM/Assertion Coverage**
  - Checks that all RTL has been exercised
  - All assertions have been exercised

- **Semi-Automated: Transaction/Structural Coverage**
  - Checks that all types of transactions have occurred
  - Ensures that the tests have sufficiently stressed the design

- **Advanced: Functional Coverage**
  - Checks that all the requirements for the design have been tested
  - Does the design work in all scenarios?

- All these coverage types are measured and tracked to determine when verification is complete and the chip can tape out
Coverage: Metric Holes

- Code/FSM/Assertion Coverage
  - Functional dead code and unreachable FSM states/transitions
  - Modes of the design that create dead code
  - Time can be wasted trying to hit these holes!

- Transaction/Structural Coverage
  - TB doesn’t stress the design enough
  - Incomplete models don’t exercise all transactions

- Functional Coverage
  - Incomplete spec or planning, lack of knowledge/time

- Proper test planning can mitigate some of these challenges

- Making use of automated formal techniques such as Questa CoverCheck can minimize time to closure
Common Methods to Achieve Coverage

- Directed Tests
  - Can target specific areas
  - Less setup typically

- Constrained Random Tests
  - More sophisticated setup
  - More automated to coverage

- Intelligent Testbench Automation
  - >10X Faster Coverage Than CRT
  - >100X More Tests Than DT

- Goals
  - Achieve total coverage faster
  - With fewer resources
  - In less time
Typical Coverage Closure Methods

- Fix design issues that prevent code coverage from being achieved

- Run more vectors to hit missing code coverage
  - Directed tests
  - Constrained random
  - Intelligent test bench generation
  - Spend a lot of time analyzing and applying new vectors

- Apply formal methods to determine coverage reachability

- Add exclusions by hand
  - Sometimes the simulator can add automated exclusions

- Use an automated flow to generate exclusions for unreachable coverage elements
Coverage Backgrounder Summary

- There is no “silver bullet” structural and functional coverage methodology or metric

- Multifaceted simulation and formal-based automation, guided by the D&V engineer’s judgment, is required
Agenda

- Introduction
- Coverage Backgrounder
- Targeting Unreachable Coverage with Formal
- Reaching Coverage Closure Faster
- Conclusion
Coverage Closure Challenges

Today coverage-driven verification is a well established methodology

Question: What if certain parts of the design simply cannot be reached?

Answer: You will run extra constrained-random tests to try to cover these parts

This can lead to a lot of wasted effort!
Example: Branch/Statement Coverage

- Dead code easily slips into the design
  — Especially after changes are made

- Dead code often identifies incorrect assumptions
  — Leading to critical bugs due to differing interpretation of design requirements

- Possibly synthesizes into logic that is not needed

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

reg T;
always @* begin
  T = 1'bX;
  case (R)
    2'b00:      T = 1'b0;
    2'b01:      T = 1'b1;
    2'b10:      T = 1'b1;  // **R can never be 2'b10**
    2'b11:      T = 1'b0;
  endcase
end
```

Hence this statement and branch can never be reached.
Example: Condition/Expression Coverage

Design configuration can have a large impact here
- Has every combination of signals been exercised?
- Is every combination of signals possible?

If “use_conf” is tied to 0 in the design, the condition “1-” of the two signals isn’t reachable

```
always @*
  case (cstate)
  IDLE: if (!use_conf && !data_empty)
    nstate <= RR_RESP;
  else
    nstate <= RR_IDLE;
  ...
endcase
```
Example: Toggle Coverage

- Typically registers and signals can’t toggle due to configuration or some other constraint/bug in the design

```verilog
always @(posedge pclk or negedge prstn)
if (!prstn)
    b_active <= 1'b0;
else
    if (apb_wr && bready)
        b_active <= 1'b0;
    else if (bready && b_active)
        b_active <= 1'b0;
    else if (apb_wr && !bready)
        b_active <= 1'b1;
    else
        b_active <= b_active;
```

Some signals which are stuck due to either configuration or a bug in the design prevent the `b_active` signal from toggling 0 => 1
Example: FSM Coverage

- Indicates an over specified state machine
  - May lead to unused logic

- Easily overlooked in simulation
  - Info is passed to simulation for exclusion in the set of coverage goals

This state is deadlocked

This transition can never happen
Traditional Coverage Closure

- **Questa UVM Simulation**
- **UCDB**
- **Review Coverage Holes**
- **Write Tests**
- **Write Waivers**
- **Coverable**
- **Uncoverable**
Questa CoverCheck

Automatic code coverage enhancement solution

Goal: 100% Code Coverage

Difficult to achieve:
1. Some coverage items cannot be reached
2. Other coverage items are difficult to hit

Problem: Wasted Time

• Engineers waste time manually identifying unreachable coverage & justifying waivers

Problem: High Effort

• Requires significant manual effort to create complex test scenarios

Questa CoverCheck

• Identifies and prunes unreachable goals
• Guide test generation for reachable goals
The Questa CoverCheck Methodology

- The unreachable code information is passed on to Questa Simulation
- Measuring coverage now will automatically excludes code that cannot be reached so you know when you are done!

No more is time wasted to try to cover unreachable code
Checks for Coverage Exclusions

- **Branch**
  - Unreachable if/else and case branches

- **Condition/Expression**
  - Unreachable FEC conditions

- **Statement**
  - Unreachable lines of code

- **Toggle**
  - Unreachable register transitions

- **FSM**
  - Unreachable FSM states and transitions

- **Covergroups**
  - Unreachable covergroup bins

Unreachable items are automatically excluded from your coverage model
The Coverage Improvement Process

1. Generate Final UCDB file from simulations

2. Run Questa CoverCheck reading final UCDB
   — Target uncovered code coverage elements
   — Run major blocks

3. Generate the exclude file

4. Apply exclusions to your simulation results
   — Update existing .ucdb file with exclude file

5. Report coverage
   — Track and manage coverage data
Questa CoverCheck Verification Flow

Use static analysis to improve simulation results!
Scaling Unreachable Analysis to the SoC Level

Top

A

B

Questa CoverCheck

Coverage Exclusions A

Coverage Exclusions Top

Coverage Exclusions B

UCDB

Questa CoverCheck

Coverage Exclusions A

Coverage Exclusions Top

Coverage Exclusions B
### Improved Code Coverage Scores

#### Code Coverage Closure Tutorial, DVCon 2015

![Code Coverage Table](image)

<table>
<thead>
<tr>
<th>Instance</th>
<th>Stmt count</th>
<th>Stmts hit</th>
<th>Stmt %</th>
<th>Stmnt graph</th>
<th>Toggle nodes</th>
<th>Toggles hit</th>
<th>Toggle %</th>
<th>Toggle graph</th>
<th>States</th>
<th>States hit</th>
<th>State %</th>
<th>State graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>130</td>
<td>98</td>
<td>75.4%</td>
<td>Red</td>
<td>1116</td>
<td>646</td>
<td>57.3%</td>
<td>Red</td>
<td>9</td>
<td>3</td>
<td>33.3%</td>
<td>Red</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>397</td>
<td>111</td>
<td>27.5%</td>
<td>Red</td>
<td>786</td>
<td>9</td>
<td>1.15%</td>
<td>Red</td>
<td>7</td>
<td>1</td>
<td>14.3%</td>
<td>Red</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>55</td>
<td>56</td>
<td>96.6%</td>
<td>Green</td>
<td>826</td>
<td>500</td>
<td>61.2%</td>
<td>Red</td>
<td>4</td>
<td>4</td>
<td>100%</td>
<td>Red</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>72</td>
<td>67</td>
<td>95.1%</td>
<td>Green</td>
<td>570</td>
<td>437</td>
<td>78.2%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
<td>8</td>
<td>3</td>
<td>37.5%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Red</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>2</td>
<td>2</td>
<td>100%</td>
<td>Green</td>
<td>144</td>
<td>46</td>
<td>31.5%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>2</td>
<td>2</td>
<td>100%</td>
<td>Green</td>
<td>144</td>
<td>49</td>
<td>34.4%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
<td>392</td>
<td>117</td>
<td>29.8%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>1</td>
<td>1</td>
<td>100%</td>
<td>Green</td>
<td>10</td>
<td>8</td>
<td>80%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>1</td>
<td>1</td>
<td>100%</td>
<td>Green</td>
<td>6</td>
<td>6</td>
<td>100%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>1</td>
<td>1</td>
<td>100%</td>
<td>Green</td>
<td>10</td>
<td>6</td>
<td>60%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>1</td>
<td>1</td>
<td>100%</td>
<td>Green</td>
<td>8</td>
<td>6</td>
<td>75%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>1</td>
<td>1</td>
<td>100%</td>
<td>Green</td>
<td>10</td>
<td>8</td>
<td>80%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>1</td>
<td>1</td>
<td>100%</td>
<td>Green</td>
<td>10</td>
<td>0</td>
<td>0%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>1</td>
<td>1</td>
<td>100%</td>
<td>Green</td>
<td>10</td>
<td>8</td>
<td>80%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>1</td>
<td>1</td>
<td>100%</td>
<td>Green</td>
<td>8</td>
<td>8</td>
<td>100%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>107</td>
<td>102</td>
<td>93.3%</td>
<td>Green</td>
<td>1126</td>
<td>717</td>
<td>63.7%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>4</td>
<td>4</td>
<td>100%</td>
<td>Green</td>
<td>436</td>
<td>246</td>
<td>56.4%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
<td>20</td>
<td>19</td>
<td>95%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
<td>20</td>
<td>19</td>
<td>95%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>32</td>
<td>32</td>
<td>100%</td>
<td>Green</td>
<td>140</td>
<td>140</td>
<td>100%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>4</td>
<td>4</td>
<td>100%</td>
<td>Green</td>
<td>436</td>
<td>222</td>
<td>50.5%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
<td>20</td>
<td>19</td>
<td>95%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>24</td>
<td>21</td>
<td>87.5%</td>
<td>Red</td>
<td>110</td>
<td>106</td>
<td>96.4%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Red</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
<td>16</td>
<td>15</td>
<td>93.8%</td>
<td>Green</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>18</td>
<td>18</td>
<td>100%</td>
<td>Green</td>
<td>806</td>
<td>717</td>
<td>88%</td>
<td>Green</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>2</td>
<td>2</td>
<td>100%</td>
<td>Green</td>
<td>70</td>
<td>3</td>
<td>4.22%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
<tr>
<td>SYSTEM/bridge32_top/bridge Avishone...</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
<td>6</td>
<td>3</td>
<td>50%</td>
<td>Red</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td>Green</td>
</tr>
</tbody>
</table>
# Simulation Coverage Before/After Exclusions

## Coverage Report Summary Data by file

### File: ../../pci/rtl/verilog/pci_target32_sm.v

<table>
<thead>
<tr>
<th>Enabled Coverage</th>
<th>Active</th>
<th>Hits</th>
<th>Misses</th>
<th>% Covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stmts</td>
<td>98</td>
<td>93</td>
<td>5</td>
<td>94.8</td>
</tr>
<tr>
<td>Branches</td>
<td>22</td>
<td>21</td>
<td>1</td>
<td>95.4</td>
</tr>
<tr>
<td>FEC Condition Terms</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100.0</td>
</tr>
<tr>
<td>FEC Expression Terms</td>
<td>186</td>
<td>127</td>
<td>59</td>
<td>68.2</td>
</tr>
<tr>
<td>FSMs</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>100.0</td>
</tr>
<tr>
<td>States</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>80.0</td>
</tr>
<tr>
<td>Transitions</td>
<td>106</td>
<td>100</td>
<td>6</td>
<td>94.3</td>
</tr>
</tbody>
</table>

### File: ../../pci/rtl/verilog/pci_target32_sm.v

<table>
<thead>
<tr>
<th>Enabled Coverage</th>
<th>Active</th>
<th>Hits</th>
<th>Misses</th>
<th>% Covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stmts</td>
<td>93</td>
<td>93</td>
<td>0</td>
<td>100.0</td>
</tr>
<tr>
<td>Branches</td>
<td>22</td>
<td>21</td>
<td>1</td>
<td>95.4</td>
</tr>
<tr>
<td>FEC Condition Terms</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100.0</td>
</tr>
<tr>
<td>FEC Expression Terms</td>
<td>186</td>
<td>128</td>
<td>58</td>
<td>68.8</td>
</tr>
<tr>
<td>FSMs</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>100.0</td>
</tr>
<tr>
<td>States</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>100.0</td>
</tr>
<tr>
<td>Transitions</td>
<td>106</td>
<td>100</td>
<td>6</td>
<td>94.3</td>
</tr>
</tbody>
</table>
CoverGroup Coverage Before/After Exclusion

Coverage Report Summary

TOTAL COVERGROUP COVERAGE: 28.1%  COVERGROUP TYPES: 4

TOTAL ASSERTION COVERAGE: 80.0%  ASSERTIONS: 5

Total Coverage By File (code coverage only, filtered view): 39.1%

TOTAL COVERGROUP COVERAGE: 48.2%  COVERGROUP TYPES: 4

TOTAL ASSERTION COVERAGE: 80.0%  ASSERTIONS: 5

Total Coverage By File (code coverage only, filtered view): 42.3%
Calculating Your ROI from Using CoverCheck

Calculating the amount of time saved in your coverage closure flow by using CoverCheck is fairly easy:
- \( N \) = the number of unreachable coverage elements
- \( T \) = the time it would have taken you to manually analyze it
- ROI = total amount of time saved automating your exclusion flow
- \( \text{ROI} = N \times T \)

Example: In one of the above examples there were over 3000 unreachable coverage elements in the design
- Let’s be generous and estimate it would have taken 15 minutes on average to analyze each unreachable item and exclude it
- \( \text{ROI} = 3000 \times 15 \text{ min} \)
- \( \text{ROI} = 45000 \text{ min} \) (750 hr)
- \( \text{ROI} = \sim 4.5 \text{ man months of effort saved} \)
Summary: CoverCheck Benefits

Schedule predictability
- Save project time that would have been spent manually reviewing the coverage holes

Improved metrics
- Automatically eliminate code that’s never meant to be exercised
- Tune measurement to the relevant modes of operation

Elimination of waiver rot
- Manually generated waivers have to be maintained as the code changes

Improved design quality
- Witness waves eliminate danger of ignoring coverage holes that are reachable
- Guides design for verification
Using Questa CoverCheck To Speed Up RTL Freeze of PCIe IP

Sundararajan Haran
Engineering Manager
Microsemi
Section Agenda

- Questa Covercheck
- PCIe evaluation bench approach
- Results
- Benefits
- References
Questa CoverCheck Details

- CoverCheck analyzes coverage items that are found to be:
  - Unreachable through simulation using a QuestaSim Universal Coverage Database (UCDB)
  - Or through a formal analysis

- CoverCheck can run without a simulation UCDB
  - App automatically runs formal analysis on the entire design to determine and analyze the unreachable items
  - Downside: this takes a long time (several hours).

```
Compile the Design

covercheck compile

Run the Analysis

covercheck load ucdb (optional)
covercheck verify
covercheck generate exclude (optional)

Verify/Debug the Results

qverify
```
Out-of-the-Box CoverCheck Flow

**Questa CoverCheck**
Use formal to improve simulation results.

- **RTL**
- **CoverCheck**
- **Coverage Exclusions**
- **TB**
- **Simulation**
- **Coverage Results**

**Step 1:** Find unreachable
codes

**Step 2:** Improve simulation coverage metrics

**Step 3:** Feedback simulation coverage data

**Step 4:** Generate waveforms to guide test creation

**Questa CoverCheck methodology.** The tool applies formal methods to target code that’s unreachable by the simulator.
Section Agenda

- Questa Covercheck
- PCIe evaluation bench approach
- Results
- Benefits
- References
PCiE Evaluation Bench Approach

- We used PCiE block level environments for this exercise

- CoverCheck was chosen and used at the block level
Section Agenda

- Questa Covercheck
- PCIe evaluation bench approach
- Results
- Benefits
- References
CoverCheck was run on the final merged coverage database of PCIe block

```
gcover check load ucdb ../../../reports/final.ucdb
gcover check compile -d pcie_system_top
gcover check generate exclude covercheck_verify.db covercheck_exclude.do
gcover check verify -effort low -witness_waveforms
gcover check verify -effort high -witness_waveforms
```

8,1 All
CoverCheck generated the exclusion list after running formal analysis with UCDB and RTL.

- Generated exclusion list was then reviewed by Design Team for Sign-off.
- Saved approx. 3 weeks which involves (reviewing the coverage database for each uncovered item.)
Section Agenda

- Questa Covercheck
- PCIe evaluation bench approach
- Results
- Benefits
- References
Benefits

The CoverCheck tool saved time in coverage exclusion analysis

- It only took 3 hours to run
- But it saved ~3 weeks of analysis/debug and design team interaction effort!
Section Agenda

- Questa Covercheck
- PCIe evaluation bench approach
- Results
- Benefits
- References
References

- PCIe block internal specification
- Questa CoverCheck User Guide, v10.3a
THANK YOU
Agenda

- Introduction
- Coverage Backgrounder
- Targeting Unreachable Coverage with Formal
- Reaching Coverage Closure Faster
- Conclusion
Coverage Closure Process

- Verification Planning
  - Requirements Mapping
  - Coverage Planning

- Testbench Creation
  - Coverage Modeling
  - Stimulus Modeling
  - Verification IP

- Achieving Coverage
  - Regression Management
  - Simulation-Based Techniques
  - **Formal-Based Techniques**

- Analysis & Reporting
  - Analyzing
  - Ranking & Merging
  - Reporting
Coverage Data Management is the Key to Reaching Overall Coverage Closure Faster

Optimized Database
- Unified
- Tracking
- Visibility
- Analysis
- Open standard

UCDB READ/WRITE API

- Merge
- Analyze
- Testplan Tracking
- Coverage
- Ranking
- Report
- Tests
Questa Verification Management

The intersection of Process, Tools and Data

- Built around a high performance Unified Coverage Database
- Electronic Coverage Closure with Testplan Tracking
- Improve Regression time-to-debug with Results Analysis
- "Are we getting closer to done?" Trend Analysis
- Improve Regression Productivity with Run Management
- Improve Code Coverage Closure with Questa CoverCheck
Create a “test plan” from your spec

1. An XML file generated from your CSV/IP-XACT/XML containing test plan entries for all checks & coverage
2. Can be converted to a UCDB and viewed/merged into the Questa Verification Management environment
Formal Integration with Project Testplan

- Testplan flow provides [multi-tech] management, tracking, & analysis

- Formal data includes coverage, proofs, and property checks
Manage and Comprehend Volume of Results with Powerful Analysis and Reporting Capabilities

Analyze

Report

68 Code Coverage Closure Tutorial, DVCon 2015
Putting it All Together: Tracking Process and Coverage Metrics

Bugs found by TYPE of VERIFICATION

- Gate-Level
- Power-Aware
- Mixed-Signal
- Digital

WorkWeeks

Cumulative Opened

Cumulative Closed

New

Opened
## Users’ Productivity Gains from Focusing on Coverage Closure & Verification Management

<table>
<thead>
<tr>
<th>Industry</th>
<th>Sub process</th>
<th>Productivity</th>
<th>Before Questa VM</th>
<th>With Questa VM</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP Developer</td>
<td>Nightly regression test time</td>
<td>Throughput</td>
<td>28 hours</td>
<td>2.5 hours</td>
<td>9 X faster</td>
</tr>
<tr>
<td></td>
<td>Results and Coverage Analysis</td>
<td>Turn-around</td>
<td>2 hours</td>
<td>20 minutes</td>
<td>6 X faster</td>
</tr>
<tr>
<td></td>
<td>Regression file cleanup</td>
<td>Capacity</td>
<td>15 minutes</td>
<td>30 seconds</td>
<td>30 X faster</td>
</tr>
<tr>
<td>Automotive</td>
<td>Nightly regression test maximum</td>
<td>Throughput</td>
<td>40 tests</td>
<td>320 tests</td>
<td>8 X more tests</td>
</tr>
<tr>
<td></td>
<td>Nightly regression test setup time</td>
<td>Turn-around</td>
<td>30 minutes</td>
<td>2 minutes</td>
<td>15 X less time</td>
</tr>
<tr>
<td></td>
<td>Nightly regression addition time</td>
<td>Turn-around</td>
<td>60 minutes</td>
<td>5 minutes</td>
<td>12 X less time</td>
</tr>
<tr>
<td></td>
<td>Nightly regression Script Files</td>
<td>Turn-around</td>
<td>10 files</td>
<td>1 file</td>
<td>10 X easier</td>
</tr>
<tr>
<td></td>
<td>Nightly regression Results Analysis</td>
<td>Turn-around</td>
<td>&gt;1 hour</td>
<td>&lt;1 minute</td>
<td>60 X faster</td>
</tr>
<tr>
<td>Microprocessor</td>
<td>Test Merge Time</td>
<td>Turn-around</td>
<td>7 days</td>
<td>7 hours</td>
<td>24 X faster</td>
</tr>
<tr>
<td></td>
<td>Data Storage</td>
<td>Capacity</td>
<td>1 GB</td>
<td>10 MB</td>
<td>100X reduction</td>
</tr>
<tr>
<td>Wireless</td>
<td>Results Analysis Queries</td>
<td>Turn-around</td>
<td>1 hour</td>
<td>15 minutes</td>
<td>4 X faster</td>
</tr>
<tr>
<td>Semiconductor</td>
<td>Merge of all coverage from all tests</td>
<td>Turn-around</td>
<td>55 hours</td>
<td>70 minutes</td>
<td>47 X faster</td>
</tr>
</tbody>
</table>
Agenda

- Introduction
- Coverage Backgrounder
- Targeting Unreachable Coverage with Formal
- Reaching Coverage Closure Faster
- Conclusion
Conclusion

- “Coverage” in all its forms is an effective way to measure progress, allocate resources, and reach sign-off faster

- The volume of coverage data is exceeding what manual inspection or basic scripting methodologies can handle

- Automated, exhaustive coverage analysis solutions have enabled engineers at companies like MicroSemi (and Microsoft, Micron, Rockwell Automation, Thales, and more) to save 1,000s of hours of compute and R&D time
Resources

- **Appendix**
  - Coverage closure case studies shared at DVCon USA last March
  - Rockwell Automation, Micron, Microsoft, Thales

- **Conference Papers**
  - DVCon 2015: “Coverage Data Exchange Is No Robbery, or Is It?”, MGC

- **Whitepapers**

- **On Demand Webinars & Courses**
  - New School Coverage Closure: [http://goo.gl/2JTy7o](http://goo.gl/2JTy7o)
  - Verification Academy: CoverCheck – Accelerating Coverage Closure [https://verificationacademy.com/sessions/CoverCheck-Accelerating-Coverage-Closure](https://verificationacademy.com/sessions/CoverCheck-Accelerating-Coverage-Closure)

- **Speaker contact info**
  - Joe Hupcey III: [Joe_Hupcey@mentor.com](mailto:Joe_Hupcey@mentor.com)
  - Nuni Srikanth (a/k/a Shree): [Nuni_Srikanth@mentor.com](mailto:Nuni_Srikanth@mentor.com)
  - Bhushan Safi: [Bhushan_Safi@mentor.com](mailto:Bhushan_Safi@mentor.com)
Code Coverage case study at Rockwell Automation
Case Study: Rockwell Automation
Analysis of Missed Coverage

The uncovered condition:

```verbatim
always @ (*) begin
  case (state_ff)
    STATE_x: begin
      if(A) begin
        if (((B) || (C && !D)) begin
          ...
        end else begin
          if (((!E && !D) || (!F && D)) begin
            ...
          end
        end
      end
    endcase
end
```

Never hit the following conditions:
- D=1 and F=1
- D=0 and E=1
Case Study: Rockwell Automation Analysis of Missed Coverage

The properties:

- The properties:

  ```
  assert property @(posedge clk) disable iff (!resetn)
  !((state_ff == STATE_X) && (A) && !(B || (C && !D)) && (D==1) && (F==1));
  ```

  ```
  assert property @(posedge clk) disable iff (!resetn)
  !((state_ff == STATE_X) && (A) && !(B || (C && !D)) && (D==0) && (E==1));
  ```

- Formal Results:
  - Assertions fired
  - The given counterexample was illegal protocol
Rockwell Automation Case Study Conclusion

- Use Formal early and often
  - Top level and block level connectivity verification
  - Top level address map verification
  - Complex control logic

- Add Formal analysis for missing coverage
  - Holes always show up late in design cycle
Appendix

Questa CoverCheck
Success at Micron
Verification Closure

Bala Chandrasekaran
ASIC Verification Engineer

DVCon 2015
About Micron SoC Design and Verification

- Micron SoC designs
  - Multi million gate NAND-controller IP blocks designed and verified

- Verification flow
  - Constrained-random, coverage driven approach using UVM
  - Testing at IP block and SoC level
  - Vplan - Requirements tracking
  - Coverage metrics
    - Functional coverage with SV cover groups
    - Assertion coverage with SVA covers
    - Code coverage

- Statement, Branch, Expression, Condition, FSM

- Sign-off requirements
  - All test requirements tracked through to completion
  - 100% functional and code coverage
## Micron Case Study: Questa AutoCheck Results

<table>
<thead>
<tr>
<th>Check</th>
<th>Evaluations</th>
<th>Found</th>
<th>Waived</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOCK_UNREACHABLE</td>
<td>1353</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>FSM_STUCK_BIT</td>
<td>101</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>FSM_UNREACHABLE_TRANS</td>
<td>220</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>INDEX_ILLEGAL</td>
<td>150</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>LOGIC_UNUSED</td>
<td>3038</td>
<td>118</td>
<td>0</td>
</tr>
<tr>
<td>X.Assign.Reachable</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>X.Unresolved</td>
<td>54</td>
<td>54</td>
<td>0</td>
</tr>
<tr>
<td><strong>AC Total</strong></td>
<td><strong>4918</strong></td>
<td><strong>178</strong></td>
<td>0</td>
</tr>
</tbody>
</table>

© Mentor Graphics Corp.  Company Confidential  www.mentor.com
Micron Case Study: Violations

- **INDEX_ILLEGAL** (17 Found)

  **Illegal Array Index**: Array index value is illegal.

  ```vhdl
  reg [1:8] v;
  always @(posedge clk)
    if (rst) v[1] <= b;
    else v[a] <= b;
  ```

- **LOGIC_UNDRIVEN** (863 Found)

  **Logic not Driven from Inputs**: Design contains logic that has no driver.

  ![Diagram of logic not driven from inputs]
Micron Case Study: Cautions

- **ASSIGN_IMPLICIT_CONSTANT** (65 Found)
  - RHS of an assignment statement includes a non-constant expression, but the statement only assigns a constant value when sensitized.

- **BLOCK_UNREACHABLE** (4 Found)
  - Block of code cannot be reached.

- **FSM_UNREACHABLE_TRANS** (1 Found)

```plaintext
int a, b, var;
if (a == 0)
    var <= a;
else
    var <= b;

reg[2:0] a, b, var;
if (a == 9)
    var <= 0;
else
    var <= b;
```

**FSM State Transition is Unreachable:** FSM has a state transition that cannot be sensitized.

- `S0`
- `S1`
- `S2`

```
if (rst | c) state <= S0;
else case (state)
    S0: state <= S1;
    S1: state <= S2;
    S2: state <= c ? S1 : S0;
endcase
```
## Micron Case Study: Questa CoverCheck Results

<table>
<thead>
<tr>
<th>Coverage Type</th>
<th>Active</th>
<th>Unreachable</th>
<th>Reachable</th>
<th>Inconclusives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch</td>
<td>731</td>
<td>78</td>
<td>636</td>
<td>17</td>
</tr>
<tr>
<td>Condition</td>
<td>135</td>
<td>15</td>
<td>113</td>
<td>7</td>
</tr>
<tr>
<td>Expression</td>
<td>483</td>
<td>153</td>
<td>315</td>
<td>15</td>
</tr>
<tr>
<td>FSM States</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Transitions</td>
<td>31</td>
<td>5</td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>Statement</td>
<td>875</td>
<td>95</td>
<td>768</td>
<td>12</td>
</tr>
<tr>
<td>Toggle</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Coverbin</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>2258</strong></td>
<td><strong>347</strong></td>
<td><strong>1860</strong></td>
<td><strong>51</strong></td>
</tr>
</tbody>
</table>
Micron Case Study: What we found

- We got AutoCheck and CoverCheck up & running in 30 min
- We found 347 unreachable items (our prior analysis missed)!
- These were found without constraints
  - If we add a reset/initialization state and constraints we could potentially find even more
- How does this impact schedule?
  - Assuming it takes 15 min to review each item
    - 347 exclusions * 15 minutes = 5,205 minutes (86.75 h)

> 2 Man Weeks Saved!

AutoCheck and CoverCheck analyses are now the required Plan of Record.
Appendix

Questa CoverCheck Success at Microsoft
Author Information

- Nguyen Le
  - Principal Design Verification Engineer
  - Microsoft Corp.
  - Email: ngle@microsoft.com
About Microsoft SoC Design and Verification

- 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
CoverCheck Case Study Results

- Exclusions improved code coverage by 10 – 15% in most blocks
  - Coverage number improved from 87% to 97%

- In auto-generated code for register blocks the improvement was 20%
  - There are simulation hooks that are unreachable
Benefits of Formal Code Exclusion

- **Improved code coverage metrics**
  - Metrics are automatically tuned to the relevant modes of operation for reused IP blocks

- **Improved design quality**
  - Exclusions are formally proven reducing the risk of ignoring important goals

- **Case study ROI**
  - Time to manually write exclusions vs. auto-generate
    (1 Design Engineer + 1 Verification Engineer) x 10 min/exclusion
    = 4 man months saved
Example: Two Days to Manually Exclude

module sample_code_cov(
  input logic reset, clk,
  input logic [3:0] code_val,
  input logic [11:0] pkt_len,
  output logic error_case
);

#define SIZE_1 4'b0001
#define SIZE_2 4'b0010
#define SIZE_3 4'b0011
#define SIZE_4 4'b0100
#define SIZE_5 4'b0101

always @(posedge reset or posedge clk)
begin
  if(reset) begin
    error_case <= 1'b0;
  end
  else begin
    if((code_val < `SIZE_1) ||
      (code_val > `SIZE_5)) begin
      error_case <= 1;
    end
    else if (((pkt_len[3:0] != 0) && (code_val == `SIZE_1)) ||
      ((pkt_len[8:0] != 0) && (code_val == `SIZE_2)) ||
      ((pkt_len[9:0] != 0) && (code_val == `SIZE_3)) ||
      ((pkt_len[10:0] != 0) && (code_val == `SIZE_4)) ||
      ((pkt_len[11:0] != 0) && (code_val == `SIZE_5))) begin
      error_case <= 1'b1;
    end
    else begin
      error_case <= 1'b0;
    end
  end
endmodule
Example: Detail Coverage

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

<table>
<thead>
<tr>
<th>Input Terminal</th>
<th>Covered</th>
<th>Reason</th>
<th>Hint</th>
</tr>
</thead>
<tbody>
<tr>
<td>(pkt_len[3:0] != 0)</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(code_val == 1)</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(pkt_len[8:0] != 0)</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(code_val == 2)</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(pkt_len[9:0] != 0)</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(code_val == 3)</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(pkt_len[10:0] != 0)</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(code_val == 4)</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(pkt_len != 0)</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(code_val == 5)</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Unreachable code_val == 1 never false
Microsoft Case Study Conclusions

- Verification of complex SoC projects is always more difficult to manage than expected

- Time saved by automatic code coverage closure is easily an order of magnitude

- Wish list
  - There are still complex FECs that the tool would give up
    - We are hoping for more complex expression to be handled
  - Or possible RTL recoding suggestion that can help the tool
Appendix

Questa CoverCheck
Success at Thales
The coverage challenge

— Coverage driven verification well adopted by industry now

- To measure that every lines of design code have been exercised
- But difficult to reach 100 % coverage
  - Insufficient or incorrect input stimulus during simulation
  - Unreachable coverage items
    - because of bugs, particular statements or configurations
- Need to identify manually unreachable parts
  - Could be a painful task
- Add extra tests to cover not reached items

This leads to a lot of effort to reach the coverage closure
Mentor solution: Questa Covercheck

— A formal tool
  — Automating the debug process of coverage closure
    — Inputs
      rtl design & ucdb simulation results (not mandatory) to focus analysis only on code items not reached by simulation
    — Outputs
      reports of proved unreachable code
      bugs or conditions making the code unreachable
      exclusions files
      for truly unreachable code
      guidance waveforms
      for code that could be reached

Points on causes of unreachable and help cover not yet reached code
Overview of Covercheck flow

— Use formal static analysis to improve code coverage results

Can be used without simulation results
Covercheck implementation flow

Phase 1: compile the design

vcom +cover=bcsf +acc

Phase 2: simulate the design

vsim –coverage -do “...;coverage save before.ucdb; exit”

Phase 3: run covercheck verification

qautocheck –c –do “do directives.tcl\ covercheck load ucdb ...
covercheck compile ...
covercheck verify .....\ covercheck generate exclude ..... 
exit“

Phase 4: analyse & debug the results

qautocheck covercheck_verify.db

Phase 5: generate new coverage results

vsim -c -viewcov before.ucdb -do “\ do exclude.do;coverage save after.ucdb;exit”

Every steps is tcl scriptable
Find root cause of unreachable code

From the window reports

Show source

Understand Why

Show root cause

Easy to pinpoint the root cause of an unreachable code and analyse if it's a bug that need to be fixed or not
The generated exclude file

```bash
vsim -c viewcov before.ucdb do "do exclude.do coverage save after.do"
```

Coverage exclude:
- `work.controller` - `controller.vhd` - `linerange 79` - `item s 1` - `comment "CoverCheck:Statement"`
- `work.datapath` - `datapath.vhd` - `linerange 52` - `item b 1` - `comment "CoverCheck:Branch"
- `work.datapath` - `datapath.vhd` - `linerange 53` - `item s 1` - `comment "CoverCheck:Statement"
- `work.controller` - `controller.vhd` - `linerange 77` - `item b 1` - `comment "CoverCheck:Branch"

---

![Diagram](image_url)

85% **Questa/ModelSim**
- RTL & tests
- UCDB

Run code coverage

Automatically create "waivers"
Create guidance waveforms

100% **Questa/ModelSim**
- RTL & tests
- UCDB

Run code coverage

---

Less effort to improve code coverage results to target 100%
Show guidance waveforms: waivers

From the window reports

Show how to reach statement

Create a testbench

Help to write a directed test or adjust constraints for a constrained random testbench

Statement_SC_51498.vhd

library modelsim_lib;
use modelsim_lib.util.all;
entity z1_replay_vhdl is
end entity;
architecture z1_replay_vhdl_rtl of z1_replay_vhdl is
begin
z1_replay_vhdl_proc : process begin
  wait for 1 ns;
signal_force (/alu/a, "0000", open, freeze, open, 0);
signal_force (/alu/b, "0000", open, freeze, open, 0);
signal_force (/alu/control, "00", open, freeze, open, 0);
signal_force (/alu/reset", "0", open, freeze, open, 0);
  wait for 40 ns;
end process z1_replay_vhdl_proc;
end architecture z1_replay_vhdl;
The evaluation (1)

— Machine
  — 32 cores @ 2.7 GHZ, 128 GB RAM, Linux RedHat 5U7 64 bits
— Questa CoverCheck 10.2b
— Design characteristics
  — 60 klines of vhdl code
  — implemented within a XILINX KINTEX7 device (xc7k325)
    — Slices 30k, DSP 178, Bram 374
The Results

— Without UCDB file (branch condition statement verification)

<table>
<thead>
<tr>
<th></th>
<th>verify effort low 30 min</th>
<th>verify effort high 7 hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actives</td>
<td>52918</td>
<td>52918</td>
</tr>
<tr>
<td>Unreachables</td>
<td>3310</td>
<td>3323</td>
</tr>
<tr>
<td>Reachables</td>
<td>23798</td>
<td>27809</td>
</tr>
<tr>
<td>Inconclusives</td>
<td>25810</td>
<td>21786</td>
</tr>
</tbody>
</table>

— Using an UCDB file

— Runtime 20 min with a verification effort to low

Passing low to high effort increases drastically the runtime verification (x 14) for a small decrease in inconclusives (15%)
Conclusion

— Easy to use tool
  – User guide, Tutorial, good support
  – Intuitive debug user interface

— Questa Covercheck brings benefits
  – Reducing time trying to hit truly unreachable code
  – Helping find stimulus to improve code coverage
  – Facilitating process review for justifying unreachables code items
    – Particularly for code that does not matter
  – Eliminating manual errors for creating exclusions files
  – Easing maintenance of exclusions files as the design evolves

— Next Steps
  – Run with higher effort levels to reduce inconclusives
  – Run on more designs