Using Mutation Coverage for Advanced Bug Hunting and Verification Signoff

Nicolae Tusinschi

onespin
making electronics reliable
Agenda

Formal Coverage
Example: FIFO Verification
Case Study: I²C Verification
PortableCoverage
Summary
Q&A
The Verification Loop

- Requirements
- Verification Plan
- Build Test Bench
- Run & Debug
- Sign-off
Assessing the Quality of Verification
If you don’t measure, you don’t know

When am I done?

• What part of the design has been **exercised** by my assertions/covers?
• Have I written **good quality** assertions?
• **Which parts** of the design have been checked by my assertions?

• Are all specified functions **implemented**?
• Are all specified functions **verified**?

---

Quantify™

GapFreeVerification™
Coverage & Bug Hunting
Two sides of the same coin

• Both coverage and bug hunting are important

• Where coverage is *analytical*, bugs are *anecdotal*

• 100% coverage with bugs in the design is unacceptable

• Extracting coverage should be quick and easy

• Report data must be meaningful
Inconclusive Formal Coverage

COI and proof core (ProofCore, FormalCore)

Cone-of-Influence
- Very over-optimistic
- Much logic not relevant for assertion proof
- Proof engines try to trim irrelevant logic

Proof Core
- Over-optimistic and engine-dependent
- Results hard to interpret*
- Need support from vendor*
- Mismatch with abstraction level of other forms of coverage*
- Makes review process much harder*

* Tutorial – Formal Verification in the Real World, DVCon US 2018, Verilab
Mutation Coverage
Addressing over optimism of proof core

Mutation Analysis Tools
- **Insert** mutations into DUT
- **Control**: can the stimulus generator activate the mutation?
- **Observe**: does the mutation propagate to a check that detects it and fails?

Mutation detected == DUT location observed by the testbench

Mutation analysis detects verification errors and gaps
Multi-Dimensional Coverage View

• Has the statement been **activated/controlled**?
  • Idea:
    – If a statement has not been activated during verification, it can’t break a check.
    – If a statement has been reached, would a check fail?
  • Can measure quality of stimuli

• Has the statement been **detected/observed**?
  • Idea:
    – If a statement is modified and activated, some checks should fail
    – Would any check fail if the statement cannot be reached?
  • Can measure quality of checkers
Mutation Coverage
Multi-dimensional view – quantity and quality

Assessing the *quality* of verification by providing a *quantitative* metric

Structural Coverage (Quantity)
- Activation & Detection Coverage—provides quantitative assessment

Functional Coverage (Quality)
- Assertion Coverage—provides qualitative assessment
Mutation Coverage Use Model for Design Verification

Iterative signoff flow for bug hunting and 100% coverage

Regular Flow

Used by verification engineers for signoff

Iterative Flow

Can be used both by designers and verification engineers for iterative signoff
Coverage Solution: Provide Meaningful Metrics

Continuous feedback for design and verification

**Designer Bring Up: Get feedback on the quality of design**

- Dead code; reachability
- Redundant code

**Verification: When quality and quantity both matter**

- Metrics should indicate gaps in verification and show you where these are
  - Missing assertions
  - Over-constraints
  - Find bugs
### Mutation Coverage Results

#### Activation coverage

<table>
<thead>
<tr>
<th>Activation / Controllability</th>
<th>Result</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dead</td>
<td>Mutation cannot be activated</td>
</tr>
<tr>
<td></td>
<td>Reached</td>
<td>Mutation activated by at least one assertion (witness)</td>
</tr>
<tr>
<td></td>
<td>Constrained</td>
<td>Mutation cannot be activated because of constraints</td>
</tr>
</tbody>
</table>

Important to identify which parts of the design are dead and which parts are over-constrained.

As the code is dead or over-constrained, one cannot control it.
## Mutation Coverage Results

Detection coverage

<table>
<thead>
<tr>
<th>Detection / Observability</th>
<th>Result</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Uncovered</td>
<td>Mutation not detected by any assertion</td>
</tr>
<tr>
<td></td>
<td>Covered</td>
<td>Mutation detected by at least one assertion</td>
</tr>
<tr>
<td></td>
<td>Unobserved</td>
<td>Mutation activated but not detected</td>
</tr>
</tbody>
</table>

**Important to assess the quality of the assertions**

**Have we observed all the design signals?**

**Do we have quality assertions?**
### Additional Coverage Results

Identify redundant code, report code excluded from analysis

<table>
<thead>
<tr>
<th>Exclusion</th>
<th>Result</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Redundant</td>
<td>No contribution to design I/O behavior</td>
</tr>
<tr>
<td></td>
<td>Verification</td>
<td>Only used for verification</td>
</tr>
<tr>
<td></td>
<td>Excluded</td>
<td>Excluded by user</td>
</tr>
</tbody>
</table>

**Important to identify redundant code**

**Assess if the code is redundant in design, or in verification**

**User can exclude code from coverage analysis**
## Overview of Mutation Coverage Results

<table>
<thead>
<tr>
<th></th>
<th>Result</th>
<th>Verification Hole</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Activation /</strong></td>
<td><strong>Reached</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Controllability</strong></td>
<td><strong>Constrained</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Dead</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Detection /</strong></td>
<td><strong>Uncovered</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Observability</strong></td>
<td><strong>Unobserved</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Covered</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Exclusion</strong></td>
<td><strong>Redundant</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Verification</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Excluded</strong></td>
<td></td>
</tr>
</tbody>
</table>
Quantify Model-Based Mutation Coverage

User Experience
• Accurate, familiar metrics
• Detects verification gaps and errors
• Intuitive interface
• Integrates with simulation metrics
• Supports bounded proofs

Under The Hood
• Includes formal-optimised mutation analysis
• Mutations in the model, not RTL
• Parallel mutations and assertions analysis
• Dedicated algorithms
• Patented technology

Model-Based Mutations
• Mutations inserted in the model (post-compile)
• No RTL instrumentation or recompilation required
# Quantify Dashboard - Key Components

## Structural Coverage Overview

<table>
<thead>
<tr>
<th>Status</th>
<th>Statements</th>
<th>Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>… covered</td>
<td>12</td>
<td>80.00%</td>
</tr>
<tr>
<td>… reached</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td>… unknown</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td>… unobserved</td>
<td>3</td>
<td>20.00%</td>
</tr>
<tr>
<td>… uncovered</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td>… constrained</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td>… dead</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td>Sum quantify targets</td>
<td>15</td>
<td>…</td>
</tr>
</tbody>
</table>

## Excluded Code Overview

<table>
<thead>
<tr>
<th>Code Status</th>
<th>Statements</th>
<th>Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xu excluded by user</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td>Xr excluded redundant code</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td>Xv excluded verification code</td>
<td>15</td>
<td>50.00%</td>
</tr>
<tr>
<td>0/1/U quantify targets</td>
<td>15</td>
<td>50.00%</td>
</tr>
<tr>
<td>Sum total code</td>
<td>30</td>
<td>12</td>
</tr>
</tbody>
</table>

## Assertion Coverage

<table>
<thead>
<tr>
<th>Id</th>
<th>Property</th>
<th>Kind</th>
<th>Proof Result</th>
<th>Proof Radius</th>
<th>Cover Result</th>
<th>Cover Radius</th>
<th>Quantified</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>sva/as_empty_from_full</td>
<td>assert</td>
<td>FORMAL_PROOF</td>
<td>infinite</td>
<td>COVER_PASS</td>
<td>9</td>
<td>yes</td>
</tr>
<tr>
<td>1</td>
<td>sva/as_full_from_empty</td>
<td>assert</td>
<td>FORMAL_PROOF</td>
<td>infinite</td>
<td>COVER_PASS</td>
<td>1</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>sva/u_fifo/as_ordering_check</td>
<td>assert</td>
<td>FORMAL_PROOF</td>
<td>infinite</td>
<td>COVER_PASS</td>
<td>2</td>
<td>yes</td>
</tr>
</tbody>
</table>
Quantify Dashboard
Directly linked to design browser

case (fsm_state_s)
  idle:
    if (start_i)
      begin
        fsm_state_next <= locking;
        load_counter <= 1'b1;
      end
    else if (write_req_i)
      cfg_reg_write <= 1'b1;
    else if (error_i)
      fsm_state_next <= error;
  locking:
    if (counter==0'h00)
      fsm_state_next <= idle;
  error:
    begin
      //error cond code/
      cfg_reg <= 4'd10;
      counter <= 4'd00;
      fsm_state_next <= idle;
    end
  else
    fsm_state_next <= idle;
endcase
Mutation Coverage for Bug Hunting

Example: FIFO
FIFO Interface

Input
ABCDEFGH..

Output
ABCDEFGH.. ✓
ABCFE GH.. ❌
ABC EFGH.. ❌
ABC DDEFGH.. ❌

resetn
clk
w_valid_i
w_ack_o
r_valid_i
r_ack_o
data_in
rptr
wptr
data
empty
full
data_out
Requirements for Verification

- Ordering is correct
- No duplication
- No data loss
- No data corruption
- Empty and full flags activation
  - Must be empty at the right time
  - Must be full at the right time
  - If empty, then eventually full
  - If full, then eventually empty
Quantify on FIFO Example—I

With no assertions at all

<table>
<thead>
<tr>
<th>Structural Coverage Overview</th>
<th>Statements</th>
<th>Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Statements</td>
<td>Branches</td>
</tr>
<tr>
<td>1 covered</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P reached</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>U unknown</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0R unobserved</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 uncovered</td>
<td>22</td>
<td>7</td>
</tr>
<tr>
<td>0C constrained</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0D dead</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sum quantify targets</td>
<td>22</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Structural Coverage by File</th>
<th>Statements</th>
<th>Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Statements</td>
<td>Branches</td>
</tr>
<tr>
<td>fifo.v</td>
<td>22</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assertion Coverage</th>
<th>Id</th>
<th>Property</th>
<th>Kind</th>
<th>Proof Result</th>
<th>Proof Radius</th>
<th>Cover Result</th>
<th>Cover Radius</th>
<th>Quantified</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Status</td>
<td>Id</td>
<td>File</td>
<td>Language</td>
<td>Kind</td>
<td>Full Name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td>----</td>
<td>----------</td>
<td>----------</td>
<td>------</td>
<td>-----------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>fifo.v</td>
<td>verilog</td>
<td>design</td>
<td>/home/onespin/my_labs/fifo_quantify_demo_v2/no_checks/rtl/fifo.v</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Quantify on FIFO Example—II

Design view

```verilog
class always_flip_edge(clk, reset)
begin
    if (reset)
        w_wr := 1'b0;
    else if ('dValid)
        w_wr := 1'b1;
    else if ('dHbd)
        w_wr := 1'b0;
    assign w_wr_i = w_wr;

    assign w_wr_i = w_wr;
    assign w_wr_i = w_wr;

    if (reset)
        wptr := 'dValid ? 1'bx : (Full ? 'b1 : 'b0);
    else if ('dValid)
        wptr := 'dValid ? 'b1 : (wptr + 'b1);
    assign wpkt_addr := wpkt_addr + wpkt_size;
    assign wpkt_size := wpkt_size + 'd1;

    if (reset)
        w_rd := 1'b0;
    else if ('dValid)
        w_rd := 1'b1;
    else if ('dHbd)
        w_rd := 1'b0;
    assign w_rd_i = w_rd;

    assign w_rd_i = w_rd;
    assign w_rd_i = w_rd;

    if (reset)
        rptr := 'dValid ? 1'bx : (Full ? 'b1 : 'b0);
    else if ('dValid)
        rptr := 'dValid ? 'b1 : (rptr + 'b1);
    assign rpkt_addr := rpkt_addr + rpkt_size;
    assign rpkt_size := rpkt_size + 'd1;

    if (reset)
        w_hbd := 1'b0;
    else if ('dValid)
        w_hbd := 1'b1;
    else if ('dHbd)
        w_hbd := 1'b0;
    assign w_hbd_i = w_hbd;

    assign w_hbd_i = w_hbd;
    assign w_hbd_i = w_hbd;

    if (reset)
        r_hbd := 1'b0;
    else if ('dValid)
        r_hbd := 1'b1;
    else if ('dHbd)
        r_hbd := 1'b0;
    assign r_hbd_i = r_hbd;

    assign r_hbd_i = r_hbd;
    assign r_hbd_i = r_hbd;
endclass
```
FIFO Verification Strategy

Uses symbolic and data abstraction

• Use two symbolic transactions for tracking all possible data values
• Send these symbolic values in a pre-determined order in the FIFO
• Ensure that they come out of the FIFO in the same order
• Use four sampling registers
  – sampled_in_d1
  – sampled_in_d2
  – sampled_out_d1
  – sampled_out_d2
• One side constraint
• One main ordering assertion
FIFO Ordering Properties

Glue logic

//-- Force d1 inside before d2
am_d1_before_d2:
assume property ( @(posedge clk)
    !sampled_in_d1 |-> !sampled_in_d2);

//-- End-to-end ordering assertion
as_ordering_check:
assert property ( @(posedge clk) disable iff (!resetn)
    sampled_in_d1 && sampled_in_d2 && !sampled_out_d1
    |-> !sampled_out_d2);
Quantify on FIFO Example—III
With just ordering assertion

**Structural Coverage Overview**

<table>
<thead>
<tr>
<th>Status</th>
<th>Statements</th>
<th>Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 covered</td>
<td>14</td>
<td>63.64%</td>
</tr>
<tr>
<td>R reached</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td>U unknown</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td>OR unobserved</td>
<td>7</td>
<td>31.82%</td>
</tr>
<tr>
<td>DC uncovered</td>
<td>1</td>
<td>4.55%</td>
</tr>
<tr>
<td>OC constrained</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td>OD dead</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td>Sum quantity targets</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>63.64%</td>
</tr>
</tbody>
</table>

**Excluded Code Overview**

<table>
<thead>
<tr>
<th>Code Status</th>
<th>Statements</th>
<th>Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>excluded by user</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td>excluded redundant code</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td>excluded by verification</td>
<td>14</td>
<td>38.69%</td>
</tr>
<tr>
<td>0/1/U quantified targets</td>
<td>22</td>
<td>61.11%</td>
</tr>
<tr>
<td>Sum total code</td>
<td>36</td>
<td>63.64%</td>
</tr>
</tbody>
</table>

**Structural Coverage by File**

<table>
<thead>
<tr>
<th>File</th>
<th>Statements</th>
<th>Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>fifo.v</td>
<td>22</td>
<td>7</td>
</tr>
<tr>
<td>fifo.sv.sv</td>
<td>14</td>
<td>4</td>
</tr>
</tbody>
</table>

**Assertion Coverage**

<table>
<thead>
<tr>
<th>Id</th>
<th>Property</th>
<th>Kind</th>
<th>Proof Result</th>
<th>Proof Radius</th>
<th>Cover Result</th>
<th>Cover Radius</th>
<th>Quantified</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>sv2u_fifo/as ordering check</td>
<td>assert</td>
<td>FORMAL_PROOF</td>
<td>infinite</td>
<td>COVER_PASS</td>
<td>63.64%</td>
<td>yes</td>
</tr>
<tr>
<td>1</td>
<td>sv2u_fifo/am_d1_before_d2</td>
<td>assume</td>
<td>FORMAL_ASSUMPTION</td>
<td>infinite</td>
<td>N/A</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>sv2u_fifo/am_if_full</td>
<td>assume</td>
<td>FORMAL_ASSUMPTION</td>
<td>infinite</td>
<td>N/A</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>sv2u_fifo/am_stable_d1</td>
<td>assume</td>
<td>FORMAL_ASSUMPTION</td>
<td>infinite</td>
<td>N/A</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>4</td>
<td>sv2u_fifo/am_stable_d2</td>
<td>assume</td>
<td>FORMAL_ASSUMPTION</td>
<td>infinite</td>
<td>N/A</td>
<td>0</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**File Status**

<table>
<thead>
<tr>
<th>Id</th>
<th>File</th>
<th>Language</th>
<th>Kind</th>
<th>Full Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>fifo</td>
<td>verilog</td>
<td>design</td>
<td>/home/onasspin/my Labs/fifo/quantify_demo_v2/Step2_ordering_check_only/fifo.v</td>
</tr>
<tr>
<td>1</td>
<td>fifo.sv.sv</td>
<td>verilog</td>
<td>design</td>
<td>/home/onasspin/my Labs/fifo/quantify_demo_v2/Step2_ordering_check_only/sv/fifo.sv.sv</td>
</tr>
</tbody>
</table>
Quantify on FIFO Example—IV

What is still missing?

Missing Coverage

- Unobserved
- Uncovered
Quantify on FIFO Example—V
Let’s add assertions on full and empty

as_empty_to_full:
  assert property (@(posedge clk) disable iff (!resetn)
      empty_o ##1 (push_i && !pop_i)[*FIFO_DEPTH] |=> full_o);

as_full_to_empty:
  assert property (@(posedge clk) disable iff (!resetn)
      full_o ##1 (pop_i && !push_i)[*FIFO_DEPTH] |=> empty_o);

as_empty_after_reset:
  assert property (@(posedge clk) !resetn |=> empty);
Quantify on FIFO Example—VI

Now, how are we doing?

72.73% design covered

Vacuous Failure
Quantify on FIFO Example—VII

What are the missing coverage targets?

Missing Coverage
- Unobserved code
- Cannot observe “empty”!
## Quantify on FIFO Example—VIII

A closer look

```plaintext
if (!resetn)
    w_ack <= 1'b1;
else if (!full)
    w_ack <= 1'b1;
else if (full)
    w_ack <= 1'b0;
assign w_ack_o = w_ack;
assign r_ack_o = empty ? 1'b0 : (full ? 1'b0 : 1'b1);
assign w_hsk = w_valid_i && w_ack_o;
```

This looks buggy …
Let’s go and fix it!
Quantify on FIFO Example—IX

After the fix on r_ack_o, coverage has increased

Coverage has increased to 77.27%

But still missing 22.7%! 
Quantify on FIFO Example—X
Let’s dig deeper to find out why

Missing coverage on w_ack and w_hsk
Unobserved code
Quantify on FIFO Example—XI
Let’s add the remainder properties

//-- Fairness constraints
assume property (@(posedge clk) disable iff (!resetn)
   !r_valid_i |-> ##[0:$] r_valid_i);

assume property (@(posedge clk) disable iff (!resetn)
   !w_valid_i |-> ##[0:$] w_valid_i);

//-- Liveness assertions
assert property (@(posedge clk) disable iff (!resetn)
   !r_hsk |-> ##[0:$] r_hsk);

assert property (@(posedge clk) disable iff (!resetn)
   !w_hsk |-> ##[0:$] w_hsk);
Quantify on FIFO Example—XII

How are we doing now?

- Still 9.09% design unobserved
- Coverage has increased to 90.91%

At this stage, if we didn’t have Quantify we would most certainly signoff the verification as we have:

- Exhaustive Proofs
- No conflicting constraints
- No vacuous proofs
- A very high metric in 90.91%

But last 10% unobserved makes us think!

- Cannot signoff yet!
Quantify on FIFO Example—XIII

So, what’s going on?

In the cycle, if the FIFO is full, then we should not accept another write.

However, we only delay the write in the following cycle.

So it looks like we are allowing the write to a full FIFO!

But … my proofs should have failed …. Why didn’t the ordering proof fail?
Quantify on FIFO Example—XIV

Let’s look at the constraints

```c
33
34
35
36

//--- Interface contraints
am_intf_full: assume property (full_o |-> !w_hsk || r_hsk);
```

When the FIFO is full, this constraint forces a read in the same cycle when there is a write.

Let’s take this constraint away ... and rerun the proofs.
Quantify on FIFO Example—XV

What happens to the proofs? Two assertions fail!
Quantify on FIFO Example—XVI

Let’s look at the failing ordering property

D2 exits FIFO before D1
Quantify on FIFO Example—XVII
What does our coverage look like?

Coverage reduced...... from 90.91% to 63.64%

Just as we were about to signoff at 90.91% we see coverage drop to 63.64% and failing properties and more design bugs!
Quantify on FIFO Example—XVIII
Fix the bug, prove, then Quantify

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td><code>assign w_ack_o = full ? 1'b0 : 1'b1;</code></td>
</tr>
<tr>
<td>41</td>
<td><code>assign r_ack_o = empty ? 1'b0 : 1'b1;</code></td>
</tr>
</tbody>
</table>
Quantify on FIFO Example—XVIII

100% Covered!
No over-constraints
No design bugs
All design statements observed
Quality of assertions is good
Ready for signoff
Quantify on FIFO Example—XIX

What happened to our constraint?

Constraints are no longer required

The design is guaranteed not to accept new data when full, and cannot be read out when empty

Let’s check that this is indeed the case

Let’s add additional assertions
Quantify on FIFO Example—XX
What happened to our constraint? It has become an assertion!

### Assertion Coverage

<table>
<thead>
<tr>
<th>Id</th>
<th>Property</th>
<th>Kind</th>
<th>Proof Result</th>
<th>Proof Radius</th>
<th>Cover Result</th>
<th>Cover Radius</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>sva/u_fifo /as_empty_after_reset</td>
<td>assert</td>
<td>FORMAL_PROOF</td>
<td>infinite</td>
<td>COVER_PASS</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>sva/u_fifo /as_empty_to_full</td>
<td>assert</td>
<td>FORMAL_PROOF</td>
<td>infinite</td>
<td>COVER_PASS</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>sva/u_fifo /as_full_to_empty</td>
<td>assert</td>
<td>FORMAL_PROOF</td>
<td>infinite</td>
<td>COVER_PASS</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>sva/u_fifo /as_intf_empty</td>
<td>assert</td>
<td>FORMAL_PROOF</td>
<td>infinite</td>
<td>COVER_PASS</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>sva/u_fifo /as_intf_full</td>
<td>assert</td>
<td>FORMAL_PROOF</td>
<td>infinite</td>
<td>COVER_PASS</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>sva/u_fifo /as_ordering_check</td>
<td>assert</td>
<td>FORMAL_PROOF</td>
<td>infinite</td>
<td>COVER_PASS</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>sva/u_fifo /as_rhsk_ininitely_often</td>
<td>assert</td>
<td>FORMAL_PROOF</td>
<td>infinite</td>
<td>COVER_PASS</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>sva/u_fifo /as_whsk_ininitely_often</td>
<td>assert</td>
<td>FORMAL_PROOF</td>
<td>infinite</td>
<td>COVER_PASS</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>sva/u_fifo /am_d1_before_d2</td>
<td>assume</td>
<td>FORMAL_ASSUME</td>
<td>infinite</td>
<td>N/A</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>sva/u_fifo /am_fair_rvalid</td>
<td>assume</td>
<td>FORMAL_ASSUME</td>
<td>infinite</td>
<td>N/A</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>sva/u_fifo /am_fair_wvalid</td>
<td>assume</td>
<td>FORMAL_ASSUME</td>
<td>infinite</td>
<td>N/A</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>sva/u_fifo /am_stable_d1</td>
<td>assume</td>
<td>FORMAL_ASSUME</td>
<td>infinite</td>
<td>N/A</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>sva/u_fifo /am_stable_d2</td>
<td>assume</td>
<td>FORMAL_ASSUME</td>
<td>infinite</td>
<td>N/A</td>
<td>0</td>
</tr>
</tbody>
</table>
Quantify on FIFO Example—XXI

We discover additional requirements on this design

### Assertion Coverage

<table>
<thead>
<tr>
<th>Id</th>
<th>Property</th>
<th>Kind</th>
<th>Proof Result</th>
<th>Proof Radius</th>
<th>Cover Result</th>
<th>Cover Radius</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>sva/u_fifo /as_empty_after_reset</td>
<td>assert</td>
<td>FORMAL_PROOF</td>
<td>infinite</td>
<td>COVER_PASS</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>sva/u_fifo /as_empty_to_full</td>
<td>assert</td>
<td>FORMAL_PROOF</td>
<td>infinite</td>
<td>COVER_PASS</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>sva/u_fifo /as_full_to_empty</td>
<td>assert</td>
<td>FORMAL_PROOF</td>
<td>infinite</td>
<td>COVER_PASS</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>sva/u_fifo /as_intf_empty</td>
<td>assert</td>
<td>FORMAL_PROOF</td>
<td>infinite</td>
<td>COVER_PASS</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>sva/u_fifo /as_intf_full</td>
<td>assert</td>
<td>FORMAL_PROOF</td>
<td>infinite</td>
<td>COVER_PASS</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>sva/u_fifo /as_ordering_check</td>
<td>assert</td>
<td>FORMAL_PROOF</td>
<td>infinite</td>
<td>COVER_PASS</td>
<td>2</td>
</tr>
</tbody>
</table>

34  // Interface Assertions
35    as_intf_empty:  assert property (empty_o |-!r_hsk);
36    as_intf_full:   assert property (full_o |-!w_hsk);
Summary of FIFO Example

Using coverage for bug hunting

• Without any test bench: everything uncovered
• Single ordering assertion: Quantify reports 63.64% coverage
• We spotted missing assertions on empty and full
• We add these assertions, prove -> RTL bug found!
• Fix, prove, then Quantify
• Still unobserved design -> need to write more assertions
• Wrote more assertions, re-ran proofs -> expected to see 100% coverage but had 90.91%
• An over-constraint in the test bench was masking another RTL bug!
Summary of FIFO Example

Bugs in your design indicate you do not have 100% coverage

• All proofs marked as proven, **AND** no property was marked unreachable, **AND** we had assertions on all design statements, **AND** yet the coverage was not 100%

• Missing coverage forced us to think

• Tool gave hints on where the gaps were

• This allowed us to unearth bugs in design and over-constraints in TB

• We fixed the RTL bug

• Constraints are not required, as design is guaranteed to have the behavior

• In fact, we prove this on the design by proving these two additional assertions

• Overall, we find bugs, remove bad constraints, find more bugs, and enrich our test bench with more good quality assertions
Tracking Coverage and Achieving Formal Verification Signoff

Case Study: Verification of I²C Serial Protocol Interface
Systematic Verification Flow

Requirement tracing and coverage are of paramount importance.
Tracking Progress in the Verification Plan

Integrating formal and simulation verification

- Requirements
- Specification
- Test Benches
- Assertions
- Simulation
- Formal ABV
- Simulation Results
- Mixed Results
- Formal Results
- Annotate Verification Plan
- Verification Plan Annotated
**Motivation**

How do we verify IP blocks implementing off-chip serial protocols?

Typically used to connect a number of ICs at relatively low data rates

I²C, SPI, UART, CAN, etc.

What would be an ideal approach?

Verify protocol compliance at the interfaces binding a VIP
Make use of a scoreboard to check data integrity

What is the challenge?

Even slow SoCs are running at frequencies starting in the range of 10MHz, while I²C standard-mode speed is up to 100kHz

- Do the math: *The formal tool needs to examine many cycles in order to prove that a single byte is transferred correctly.*
I²C Bus Protocol

- MCU-A
- LCD DRIVER
- EEPROM
- GATE ARRAY
- ACD
- MCU-B

SCL
SDA
The Verification Process

Verification plan: what needs to be verified?

DUT Spec

I²C – Spec (UM10204)

V-Plan

- 1. SW programmable register
  - 1.a Read read-only registers
  - 1.b Read after write registers
  - 1.c Clear command register at transfer complete, or arbitration lost
  - 1.d Reset registers
- 2. Reset functionality
- 3. Arbitration lost Interrupt, with automatic transfer cancelation
  - 3.a Core drives SDA high, but other master keeps SDA line high
  - 3.b Incoming stop detected, but not requested
- 4. Condition generation
  - 4.a Start condition generation
  - 4.b Repeated Start condition generation
  - 4.c Stop condition generation
- 5. Bus busy detection
  - 5.a Incoming start detection
  - 5.b Incoming stop detection
- 6. Data validity
  - 6.a SDA line must be stable when SCL line high
  - 7. Clock synchronization, between two masters engaging the bus at the same time
    - 7.a SCL line held LOW by the device with longest LOW period
    - 7.b SCL line held HIGH by the device with shortest HIGH period
- 8. Clock stretching, slave introduces wait states
  - 8.a During transfer master drives SCL high, but slave keeps SCL low
- 9. Slave address transfer
  - 9.a 7bit addressing mode
  - 9.b 10bit addressing mode
- 10. Data transfer
  - 10.a Write operation
  - 10.b Read operation
- 11. Acknowledge detection from slave - write operation
- 12. Acknowledge generation to slave - read operation
- 13. Interrupt handling
- 14. Range of input frequencies
The Verification Process
What is the very first verification step?

Let’s analyze the design.

Let’s do an automatic inspection. Why?

- Signal domain violation
- Dead code
- Unreachable FSM states
- Signal toggling

Validate results: are failing checks expected?

<table>
<thead>
<tr>
<th>Type of Checks</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>full case checks</td>
<td>2</td>
</tr>
<tr>
<td>parallel case checks</td>
<td>2</td>
</tr>
<tr>
<td>resolution_x checks</td>
<td>2</td>
</tr>
<tr>
<td>signal_domain checks</td>
<td>2</td>
</tr>
<tr>
<td>init checks</td>
<td>128</td>
</tr>
<tr>
<td>fsm checks</td>
<td>2</td>
</tr>
<tr>
<td>dead_code checks</td>
<td>134</td>
</tr>
<tr>
<td>stick checks</td>
<td>108</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type of Checks</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 hold</td>
<td>0</td>
</tr>
<tr>
<td>0 fail</td>
<td>0</td>
</tr>
<tr>
<td>0 open</td>
<td>0</td>
</tr>
<tr>
<td>3 hold</td>
<td>0</td>
</tr>
<tr>
<td>0 fail</td>
<td>0</td>
</tr>
<tr>
<td>0 open</td>
<td>0</td>
</tr>
<tr>
<td>0 hold</td>
<td>2</td>
</tr>
<tr>
<td>2 fail</td>
<td>0</td>
</tr>
<tr>
<td>0 open</td>
<td>0</td>
</tr>
<tr>
<td>118 hold</td>
<td>10</td>
</tr>
<tr>
<td>10 fail</td>
<td>0</td>
</tr>
<tr>
<td>0 open</td>
<td>0</td>
</tr>
<tr>
<td>134 hold</td>
<td>0</td>
</tr>
<tr>
<td>0 fail</td>
<td>0</td>
</tr>
<tr>
<td>0 open</td>
<td>0</td>
</tr>
<tr>
<td>106 hold</td>
<td>2</td>
</tr>
<tr>
<td>2 fail</td>
<td>0</td>
</tr>
<tr>
<td>0 open</td>
<td>0</td>
</tr>
</tbody>
</table>
The Verification Process

What is the verification approach?

Important to have a well-defined flow!
# Quantify Coverage Results

## Quantify MDV Overview

### Structural Coverage Overview

<table>
<thead>
<tr>
<th>Status</th>
<th>Statements</th>
<th>Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>covered</td>
<td>246</td>
<td>111</td>
</tr>
<tr>
<td>reached</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>unknown</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>unobserved</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>uncovered</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>constrained</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>dead</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sum</td>
<td>254</td>
<td>112</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>Statements</th>
<th>Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>covered</td>
<td>246</td>
<td>111</td>
</tr>
<tr>
<td>reached</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>unknown</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>unobserved</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>uncovered</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>constrained</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>dead</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sum</td>
<td>254</td>
<td>112</td>
</tr>
</tbody>
</table>

## Structural Coverage by File

<table>
<thead>
<tr>
<th>File</th>
<th>Statements</th>
<th>Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_master_bit_ctrl.v</td>
<td>133</td>
<td>48</td>
</tr>
<tr>
<td>i2c_master_byte_ctrl.v</td>
<td>55</td>
<td>30</td>
</tr>
<tr>
<td>i2c_master_top.v</td>
<td>55</td>
<td>28</td>
</tr>
</tbody>
</table>
Tracking Progress Over Time

Verification Process Overview

-Bug Hunting
-Complete Verification Plan
-Fix Coverage Holes

- ST-Covered
- ST-Constrained
- Vplan-Progress
- BR-Covered
- Bugs

Graph shows the tracking progress over time with various metrics such as Bug Hunting, Complete Verification Plan, and Fix Coverage Holes. The graph includes metrics like ST-Covered, ST-Constrained, Vplan-Progress, BR-Covered, and Bugs.
Quantify Coverage Results
Detection of over-constrained code

/**************************** ******************************************************/
/* 28 SEP */
/**************************** ******************************************************/

// RD is mutual exclusive to WR
am_read_exclusive_to_write:
assume property(disable iff(!rstn || wb_rst_i)
    write_active |-> RD != WR );
/**************************** ******************************************************/
Quantify Coverage Results
Detection of over-constrained code

```c
if (start)
begin
    c_state <= ST_START;
core_cmd <= 'I2C_CMD_START;
end
else if (read)
begin
    c_state <= ST_READ;
core_cmd <= 'I2C_CMD_READ;
end
else if (write)
begin
    c_state <= ST_WRITE;
core_cmd <= 'I2C_CMD_WRITE;
end
else // stop
begin
    c_state <= ST_STOP;
core_cmd <= 'I2C_CMD_STOP;
end
```

// RD is mutual exclusive to WR
am_read-exclusive-to-write:
assume property( disable iff (!rstn || wb_rst_i)
    write_active |-> !(RD && WR));

START immediately followed by a STOP is an illegal format
Tracking Progress Over Time

Coverage vs effort

Coverage vs. Effort

Bug Hunting

Complete Verification Plan

Fix Coverage Holes

- ST-Covered
- ST-Reached
- ST-Unknown
- ST-Unobserved
- ST-Uncovered
- ST-Constrained

Assertions

Constraints
Summary of I²C Case Study

What is the motivation?
Off-chip serial protocols are everywhere, therefore we need to verify protocol compliance and data integrity.
Verifying serial protocols with formal is challenging.

Why does the approach matter?
Having a well-defined verification approach helps in achieving great results.
Coverage increases confidence and helps us to easily identify over-constrained, not exercised code.
Collecting regression data over time gives a clear view on where effort is being expended and how things are progressing.
Quantify Formal Coverage: Scalable and Automated

<table>
<thead>
<tr>
<th>Design</th>
<th>#Code Lines</th>
<th>#Assertions</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIFO</td>
<td>321</td>
<td>30</td>
<td>100s</td>
</tr>
<tr>
<td>FSM-DDR2-Read</td>
<td>839</td>
<td>6</td>
<td>106s</td>
</tr>
<tr>
<td>vCore-Processor</td>
<td>295</td>
<td>8</td>
<td>204s</td>
</tr>
<tr>
<td>Arithmetic Block</td>
<td>383</td>
<td>2</td>
<td>257s</td>
</tr>
</tbody>
</table>

Real example at Infineon
Quantify identified verification holes and guided assertion development.
New assertions detected critical bugs.

Quantify now used to provide management metrics on all designs!

Formal Safety Verification with Qualified Property Sets
Holger Busch at DAC’14 in Accelerating Productivity Through Formal and Static Methods (Session 38.3)
Interoperable Coverage Solution

PortableCoverage

- Integrate formal and simulation coverage
- Accelerate coverage closure
- Track formal results in verification plan

PortableCoverage™

- Use any simulator and planning tool

Simulation Verification (Optional)

Formal Verification

- Verification Coverage Integration App
- Coverage Closure Accelerator App
- Verification Planning Integration App
- Quantify™ Model-Based Mutation Coverage

Verification Plan

Coverage Database

Coverage Closure

Verification Signoff

acellera SYSTEMS INITIATIVE

DVCON 2018 CONFERENCE AND EXHIBITION EUROPE
Formal-Simulation Seamless Integration

1. Design Specification
   - Verification Plan
     - DUT
     - Testbench
     - Simulator
     - Test Results
     - Simulation Coverage
     - CCA App
     - Property Checker
     - Quantify
   - Assertions
     - Formal Coverage
     - Proof Results
     - VCI App
     - VPI App
   - Coverage Viewer
Side-by-Side Analysis of Coverage Contributions

Verification Coverage Integration (VCI) App

Simulation Coverage

Quantify Formal Coverage

VCI App
Summary

**Design Bring Up**
- Automated checks
- Reachability analysis – find design bugs as you bring up design
- Redundant code – find wasted area in your design
- Designer asserts – get coverage when you have designer asserts

**Verification Quality and Metrics**
- Metrics indicate gaps in verification and show you ‘where’ these gaps are
- Quantify identifies missing or low quality assertions
- Identify accidental over-constraints, focus on verification
- Pushbutton solution: run frequently and track progress

**PortableCoverage**
- Integrate formal and simulation coverage
- Accelerate coverage closure
- Track formal coverage results in the verification plan
- Use any simulator, coverage database, verification planning tool
References and Further Reading

Formal Safety Verification with Qualified Property Sets
Holger Busch at DAC’14 in Accelerating Productivity Through Formal and Static Methods (Session 38.3)

Design Verification Is All About Good Hygiene
https://www.onespin.com/resources/white-papers/

Planning Out Verification
https://www.onespin.com/resources/videos/

Compatible Qualification Metrics for Formal Property Checking
Thank you!

Questions?