Using Mutation Coverage for Advanced Bug Hunting and Verification Signoff

Nicolae Tusinschi

onespin

making electronics reliable

accellera SYSTEMS INITIATIVE

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Agenda

Formal Coverage

Example: FIFO Verification

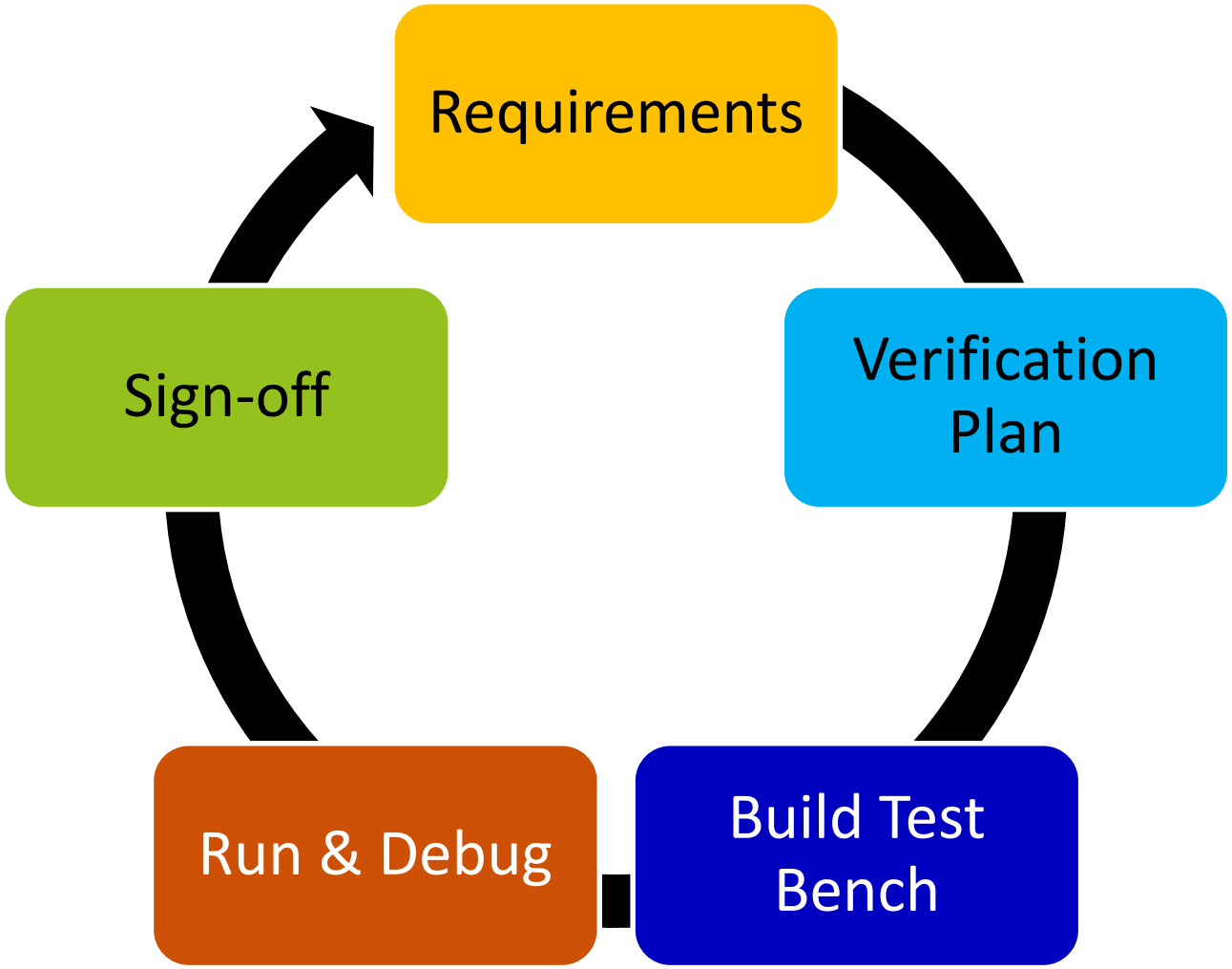
Case Study: I^2C Verification

PortableCoverage

Summary

Q&A

The Verification Loop



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?

} **Quantify™**

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

} **GapFreeVerification™**

Coverage & Bug Hunting

Two sides of the same coin

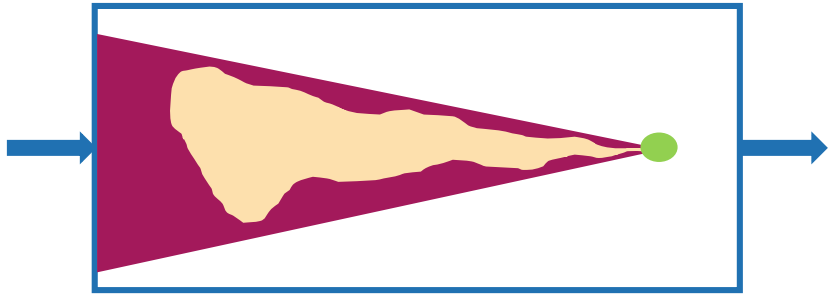
- Both coverage and bug hunting are important
- Where coverage is *analytical*, bugs are *anecdotal*
- 100% coverage with bugs in the design is unacceptable
- Extracting coverage should be quick and easy
- Report data must be meaningful

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

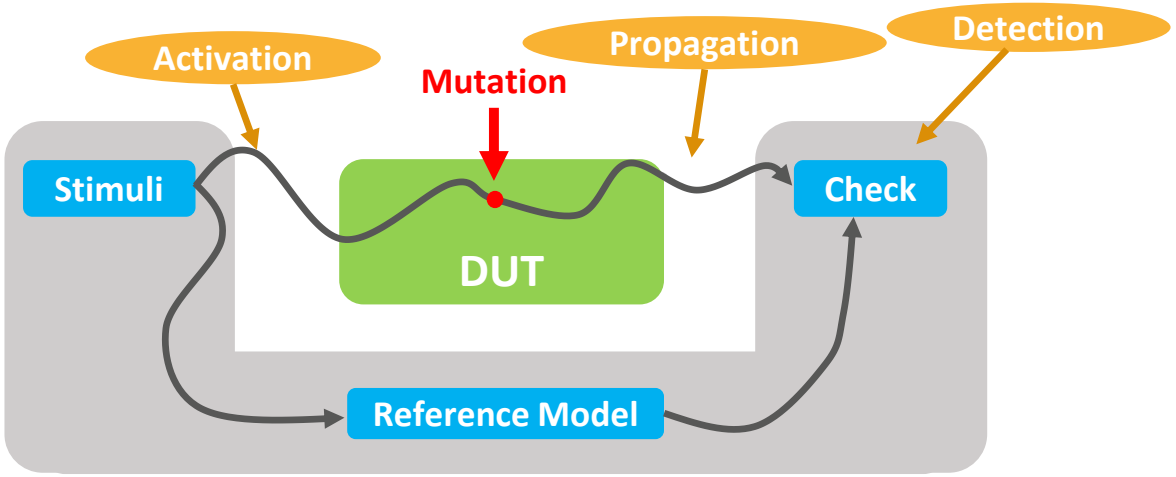
- Over-optimistic and engine-dependent
- Results hard to interpret*
- Need support from vendor*
- Mismatch with abstraction level of other forms of coverage*
- Makes review process much harder*

Design
Assertion
Cone-Of-Influence (COI)
Proof Core

* Tutorial – Formal Verification in the Real World, DVCon US 2018, Verilab

Mutation Coverage

Addressing over optimism of proof core



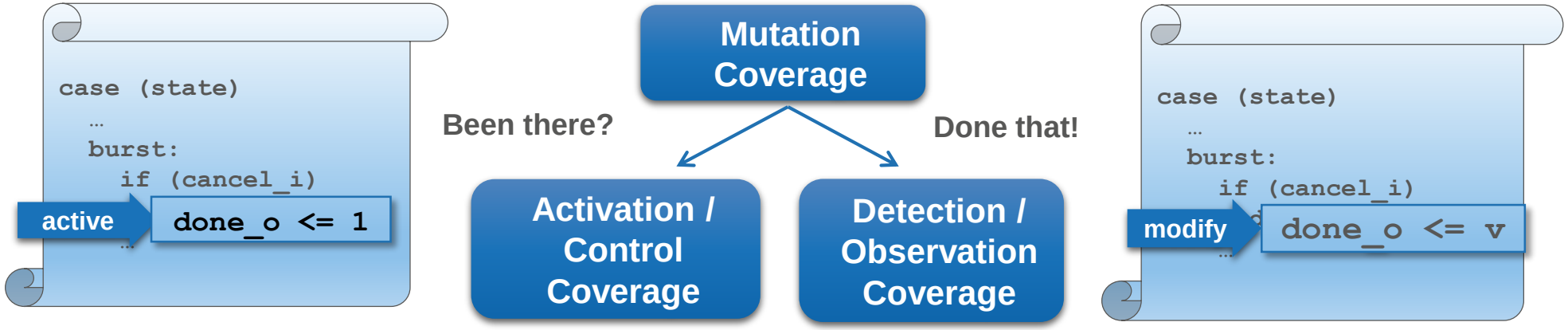
Mutation Analysis Tools

- **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)

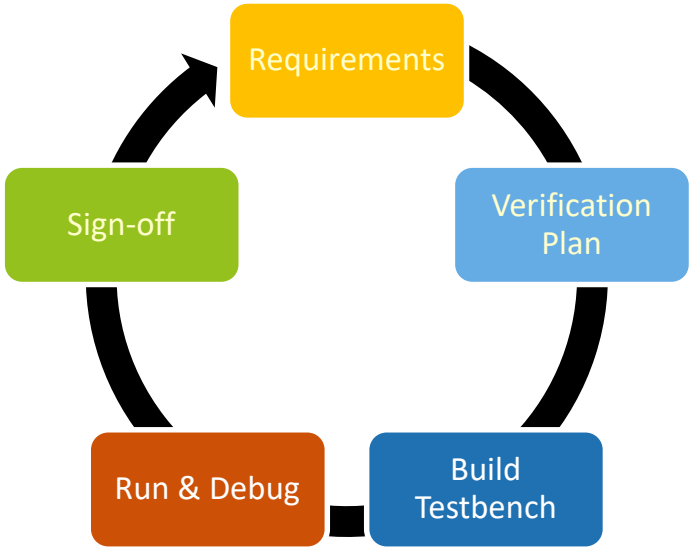
Activation & Detection Coverage—provides quantitative assessment

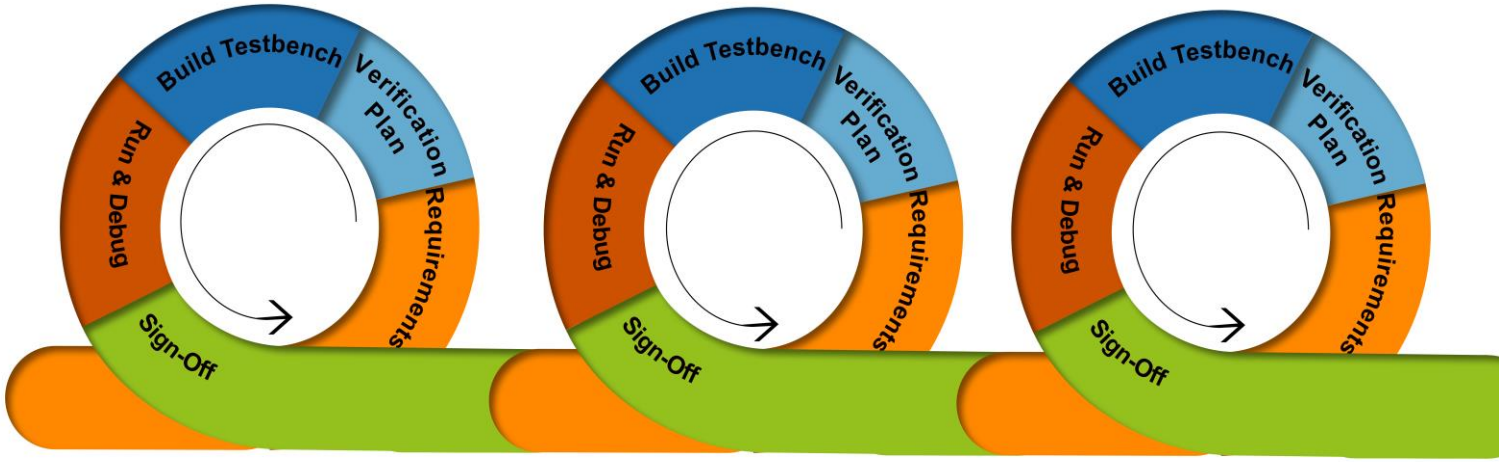
Functional Coverage (Quality)

Assertion Coverage—provides qualitative assessment

Mutation Coverage Use Model for Design Verification

Iterative signoff flow for bug hunting and 100% coverage





Regular Flow

Iterative Flow

Used by verification engineers for signoff

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

	Result	Meaning
Activation / Controllability	Dead	Mutation cannot be activated
	Reached	Mutation activated by at least one assertion (witness)
	Constrained	Mutation cannot be activated because of constraints

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

	Result	Meaning
Detection / Observability	Uncovered	Mutation not detected by any assertion
	Covered	Mutation detected by at least one assertion
	Unobserved	Mutation activated but not detected

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

	Result	Meaning
Exclusion	Redundant	No contribution to design I/O behavior
	Verification	Only used for verification
	Excluded	Excluded by user

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

	Result
Activation / Controllability	Reached
	Constrained
	Dead
Detection / Observability	Uncovered
	Unobserved
	Covered
Exclusion	Redundant
	Verification
	Excluded

Verification Hole

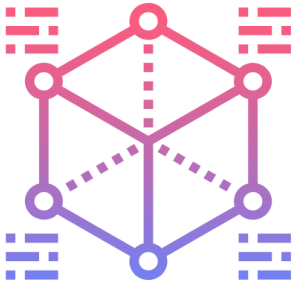
Quantify Model-Based Mutation Coverage

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

Status		Statements		Branches	
1	covered	12	80.00%	4	100.00%
R	reached	0	0.00%	0	0.00%
U	unknown	0	0.00%	0	0.00%
0R	unobserved	3	20.00%	0	0.00%
0	uncovered	0	0.00%	0	0.00%
0C	constrained	0	0.00%	0	0.00%
0D	dead	0	0.00%	0	0.00%
Sum	**quantify targets**	**15**		**4**	

Excluded Code Overview

Code Status		Statements		Branches	
Xu	excluded by user	0	0.00%	0	0.00%
Xr	excluded redundant code	0	0.00%	0	0.00%
Xv	excluded verification code	15	50.00%	8	66.67%
0/1/U	quantify targets	15	50.00%	4	33.33%
Sum	**total code**	**30**		**12**	

Assertion Coverage

Id	Property	Kind	Proof Result	Proof Radius	Cover Result	Cover Radius	Quantified
0	sva/as_empty_from_full	assert	FORMAL_PROOF	infinite	COVER_PASS	9	yes
1	sva/as_full_from_empty	assert	FORMAL_PROOF	infinite	COVER_PASS	1	yes
2	sva/u_fifo_/as_ordering_check	assert	FORMAL_PROOF	infinite	COVER_PASS	2	yes

Quantify Dashboard

Directly linked to design browser

verified code

verification hole

constrained code

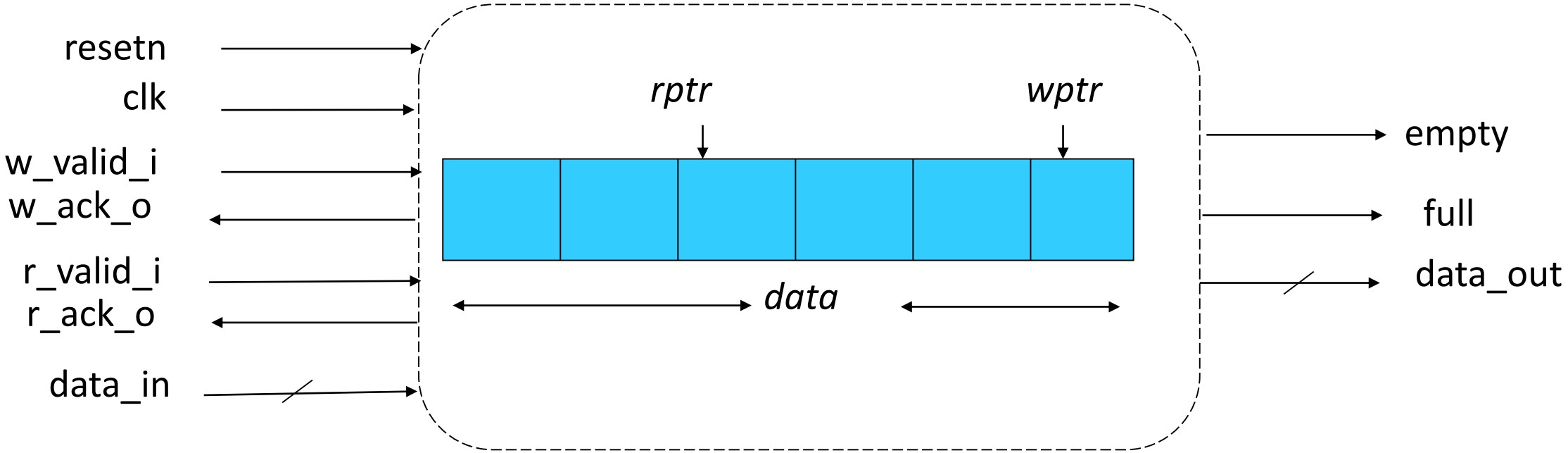
dead code

```
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==8'h00)
      fsm_state_next <= idle;
  error:
      if (error_i)
          begin
            //error cond code//
            cfg_reg     <= 4'd10;
            counter     <= 4'd00;
            fsm_state_next <= idle;
          end
        else
            fsm_state_next <= idle;
  default:
      fsm_state_next <= idle;
  endcase
```

Mutation Coverage for Bug Hunting

Example: FIFO

FIFO Interface



Input

ABCDEFGH..

Output

ABCDEFGH.. ✓

ABCDFE GH.. ✗

ABC EFGH.. ✗

ABCDDEFGH.. ✗

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

Structural Coverage Overview

Status		Statements		Branches	
1	covered	0	0.00%	0	0.00%
R	reached	0	0.00%	0	0.00%
U	unknown	0	0.00%	0	0.00%
0R	unobserved	0	0.00%	0	0.00%
0	uncovered	22	100.00%	7	100.00%
0C	constrained	0	0.00%	0	0.00%
0D	dead	0	0.00%	0	0.00%
Sum	**quantify targets**	**22**		**7**	

VERIFICATION HOLE

Structural Coverage by File

File	Statements	Branches
fifo.v	22	7

VERIFICATION HOLE

Assertion Coverage

Id	Property	Kind	Proof Result	Proof Radius	Cover Result	Cover Radius	Quantified

File Status

Id	File	Language	Kind	Full Name
0	fifo.v	verilog	design	/home/onespin/my_labs/fifo_quantify_demo_v2/no_checks/rtl/fifo.v

Quantify on FIFO Example—II

Design view

```
43    always @(posedge clk or negedge resetn)
44      if (!resetn)                                                    0
45        w_ack <= 1'b1;                                                0
46      else if (!full)                                                 0
47        w_ack <= 1'b1;                                                0
48      else if (full)                                                  0
49        w_ack <= 1'b0;                                                0
50
51    assign w_ack_o = w_ack;                                           0
52    assign r_ack_o = empty ? 1'b0 : (full ? 1'b0 : 1'b1);             0
53    assign w_hsk = w_valid_i && w_ack_o;                              0
54    assign r_hsk = r_valid_i && r_ack_o;                              0
55    assign nxt_wptr  = wptr + w_hsk;                                  0
56    assign nxt_rptr  = rptr + r_hsk;                                  0
57    assign nxt_empty = (empty || r_hsk) && !w_hsk && (nxt_rptr == nxt_wptr);  0
58
59  //--- Registered calculations for empty, wptr and rptr
60    always @(posedge clk or negedge resetn)
61     if (!resetn)                                                     0
62       begin
63         empty <= 1'b1;                                               0
64         wptr  <= {DEPTH_BITS{1'b0}};                                 0
65         rptr  <= {DEPTH_BITS{1'b0}};                                 0
66       end
67     else                                                             0
68       begin
69         empty <= nxt_empty;                                          0
70         wptr  <= nxt_wptr;                                           0
71         rptr  <= nxt_rptr;                                           0
72       end
73  //--- Write the data on a w_hsk
74    always @(posedge clk)
75      if (w_hsk)                                                     0
76        data[wptr]     <= data_i;                                    0
77  //--- Read the data on a r_hsk
78    always @(posedge clk)
79      if (r_hsk)                                                     0
80        data_int <= data[rptr];                                      0
81    assign full  = !empty && (rptr == wptr);                          0
82    assign empty_o = empty;                                           0
83    assign full_o  = full;                                            0
84    assign data_o  = data_int;                                        0
85  endmodule
```

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

Status		Statements		Branches	
1	covered	14	63.64%		
R	reached	0	0.00%	0	0.00%
U	unknown	0	0.00%	0	0.00%
0R	unobserved	7	31.82%	2	28.57%
0	uncovered	1	4.55%		
0C	constrained	0	0.00%	0	0.00%
0D	dead	0	0.00%	0	0.00%
Sum	**quantify targets**	**22**		**7**	

63.64% design covered

4.55% Design Uncovered

31.82% Design Unobserved

Excluded Code Overview

Code Status		Statements		Branches	
Xu	excluded by user	0	0.00%	0	0.00%
Xr	excluded redundant code	0	0.00%	0	0.00%
Xv	excluded verification code	14	38.89%	4	36.36%
0/1/U	quantify targets	22	61.11%	7	63.64%
Sum	**total code**	**36**		**11**	

Structural Coverage by File

File	Statements	Branches
fifo.v	22	7
fifo_sva.sv	14	4

Assertion Coverage

Id	Property	Kind	Proof Result	Proof Radius	Cover Result	Cover Radius	Quantified
0	sva/u_fifo_/as_ordering_check	assert	FORMAL_PROOF	infinite	COVER_PASS	2	yes
1	sva/u_fifo_/am_d1_before_d2	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A
2	sva/u_fifo_/am_intf_full	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A
3	sva/u_fifo_/am_stable_d1	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A
4	sva/u_fifo_/am_stable_d2	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A

Single Assertion

File Status

Id	File	Language	Kind	Full Name
0	fifo.v	verilog	design	/home/onespin/my_labs/fifo_quantify_demo_v2/Step2_ordering_check_only/rtl/fifo.v
1	fifo_sva.sv	verilog	design	/home/onespin/my_labs/fifo_quantify_demo_v2/Step2_ordering_check_only/sva/fifo_sva.sv

Quantify on FIFO Example—IV

What is still missing?

```
41      if (!resetn)                                                          0R
42        w_ack <= 1'b1;                                                      0R
43      else if (!full)                                                       0R
44        w_ack <= 1'b1;                                                      0R
45      else if (full)                                                        0
46        w_ack <= 1'b0;                                                      0
47
48    assign w_ack_o = w_ack;                                                 0R
49    //assign r_ack_o = empty ? 1'b0: 1'b1;
50    assign r_ack_o = empty ? 1'b0 : (full ? 1'b0 : 1'b1);                   1
51    assign w_hsk = w_valid_i && w_ack_o;                                    0R
52    assign r_hsk = r_valid_i && r_ack_o;                                    1
53    assign nxt_wptr  = wptr + w_hsk;                                        1
54    assign nxt_rptr  = rptr + r_hsk;                                        1
55    assign nxt_empty = (empty || r_hsk) && !w_hsk && (nxt_rptr == nxt_wptr); 1
56  //--- Registered calculations for empty, wptr and rptr
57    always @(posedge clk or negedge resetn)
58     if (!resetn)                                                           1
59       begin
60         empty <= 1'b1;                                                     0R
61         wptr  <= {DEPTH_BITS{1'b0}};                                       1
62         rptr  <= {DEPTH_BITS{1'b0}};                                       1
63       end
64     else                                                                   1
65       begin
66         empty <= nxt_empty;                                                1
67         wptr  <= nxt_wptr;                                                 1
68         rptr  <= nxt_rptr;                                                 1
69       end
70  //--- Write the data on a w_hsk
71    always @(posedge clk)
72      if (w_hsk)                                                           1
73        data[wptr]     <= data_i;                                          1
74  //--- Read the data on a r_hsk
75    always @(posedge clk)
76      if (r_hsk)                                                           1
77        data_int <= data[rptr];                                            1
78    assign full  = !empty && (rptr == wptr);                               1
79    assign empty_o = empty;                                                0R
80    assign full_o  = full;                                                 0R
81    assign data_o  = data_int;                                             1
82  endmodule
```

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?

Structural Coverage Overview

Status		Statements		Branches	
1	covered	16	72.73%		
R	reached	0	0.00%	0	0.00%
U	unknown	0	0.00%	0	0.00%
0R	unobserved	6	27.27%	3	42.86%
0	uncovered	0	0.00%	0	0.00%
0C	constrained	0	0.00%	0	0.00%
0D	dead	0	0.00%	0	0.00%
Sum	**quantify targets**	**22**		**7**	

72.73% design covered

Excluded Code Overview

Code Status		Statements		Branches	
Xu	excluded by user	0	0.00%	0	0.00%
Xr	excluded redundant code	0	0.00%	0	0.00%
Xv	excluded verification code	14	38.89%	4	36.36%
0/1/U	quantify targets	22	61.11%	7	63.64%
Sum	**total code**	**36**		**11**	

Structural Coverage by File

File	Statements	Branches
fifo.v	22	7
fifo_sva.sv	14	4

Assertion Coverage

Id	Property	Kind	Proof Result	Proof Radius	Cover Result	Cover Radius	Quantified
0	sva/u_fifo_/as_empty_after_reset	assert	FORMAL_PROOF	infinite	COVER_PASS	1	yes
1	sva/u_fifo_/as_empty_to_full	assert	FORMAL_PROOF	infinite	COVER_PASS	1	yes
2	sva/u_fifo_/as_full_to_empty	assert	FORMAL_VACUOUS	infinite	COVER_VACUOUS	infinite	no
3	sva/u_fifo_/as_ordering_check	assert	FORMAL_PROOF	infinite	COVER_PASS	2	yes
4	sva/u_fifo_/am_d1_before_d2	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A
5	sva/u_fifo_/am_intf_full	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A
6	sva/u_fifo_/am_stable_d1	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A
7	sva/u_fifo_/am_stable_d2	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A

Vacuous Failure

File Status

Id	File	Language	Kind	Full Name
0	fifo.v	verilog	design	/home/onespin/my_labs/fifo_quantify_demo_v2/Step3_with_empty_full_checks/rtl/fifo.v
1	fifo_sva.sv	verilog	design	/home/onespin/my_labs/fifo_quantify_demo_v2/Step3_with_empty_full_checks/sva/fifo_sva.sv

Quantify on FIFO Example—VII

What are the missing coverage targets?

```
42      if (!resetn)                                                                  0R
43        w_ack <= 1'b1;                                                              0R
44      else if (!full)                                                               0R
45        w_ack <= 1'b1;                                                              0R
46      else if (full)                                                                0R
47        w_ack <= 1'b0;                                                              0R
48
49    assign w_ack_o = w_ack;                                                         0R
50    assign r_ack_o = empty ? 1'b0 : (full ? 1'b0 : 1'b1);                           1
51    assign w_hsk = w_valid_i && w_ack_o;                                            0R
52    assign r_hsk = r_valid_i && r_ack_o;                                            1
53    assign nxt_wptr  = wptr + w_hsk;                                                1
54    assign nxt_rptr  = rptr + r_hsk;                                                1
55    assign nxt_empty = (empty || r_hsk) && !w_hsk && (nxt_rptr == nxt_wptr);        1
56  //--- Registered calculations for empty, wptr and rptr
57    always @(posedge clk or negedge resetn)
58     if (!resetn)                                                                   1
59       begin
60         empty <= 1'b1;                                                             1
61         wptr  <= {DEPTH_BITS{1'b0}};                                               1
62         rptr  <= {DEPTH_BITS{1'b0}};                                               1
63       end
64     else                                                                           1
65       begin
66         empty <= nxt_empty;                                                        1
67         wptr  <= nxt_wptr;                                                         1
68         rptr  <= nxt_rptr;                                                         1
69       end
70
71  //--- Write the data on a w_hsk
72    always @(posedge clk)
73      if (w_hsk)                                                                   1
74        data[wptr]     <= data_i;                                                  1
75
76  //--- Read the data on a r_hsk
77    always @(posedge clk)
78      if (r_hsk)                                                                   1
79        data_int <= data[rptr];                                                    1
80
81    assign full  = !empty && (rptr == wptr);                                        1
82    assign empty_o = empty;                                                         0R
83    assign full_o  = full;                                                          1
84    assign data_o  = data_int;                                                      1
85  endmodule
```

Missing Coverage

- Unobserved code
- Cannot observe “empty”!

Quantify on FIFO Example—VIII

A closer look

42	`if (!resetn)`	0
43	`w_ack <= 1'b1;`	0
44	`else if (!full)`	0R
45	`w_ack <= 1'b1;`	0R
46	`else if (full)`	0R
47	`w_ack <= 1'b0;`	0R
48		
49	`assign w_ack_o = w_ack;`	0R
50	`assign r_ack_o = empty ? 1'b0 : (full ? 1'b0 : 1'b1);`	1
51	`assign w_hsk = w_valid_i && w_ack_o;`	0R

This looks buggy …
Let’s go and fix it!

Quantify on FIFO Example—IX

After the fix on r_ack_o, coverage has increased

Structural Coverage Overview

Status		Statements		Branches	
1	covered	17	77.27%		
R	reached	0	0.00%	0	0.00%
U	unknown	0	0.00%	0	0.00%
0R	unobserved	5	22.73%	3	42.86%
0	uncovered	0	0.00%	0	0.00%
0C	constrained	0	0.00%	0	0.00%
0D	dead	0	0.00%	0	0.00%
Sum	quantify targets	22		7	

77.27% design covered

Still 22.7% design unobserved

Excluded Code Overview

Code Status		Statements		Branches	
Xu	excluded by user	0	0.00%	0	0.00%
Xr	excluded redundant code	0	0.00%	0	0.00%
Xv	excluded verification code	14	38.89%	4	36.36%
0/1/U	quantify targets	22	61.11%	7	63.64%
Sum	total code	36		11	

Coverage has increased to 77.27%

Structural Coverage by File

File	Statements	Branches
fifo.v	22	7
fifo_sva.sv	14	4

But still missing 22.7%!

Assertion Coverage

Id	Property	Kind	Proof Result	Proof Radius	Cover Result	Cover Radius	Quantified
0	sva/u_fifo_/as_empty_after_reset	assert	FORMAL_PROOF	infinite	COVER_PASS	1	yes
1	sva/u_fifo_/as_empty_to_full	assert	FORMAL_PROOF	infinite	COVER_PASS	1	yes
2	sva/u_fifo_/as_full_to_empty	assert	FORMAL_PROOF	infinite	COVER_PASS	5	yes
3	sva/u_fifo_/as_ordering_check	assert	FORMAL_PROOF	infinite	COVER_PASS	2	yes
4	sva/u_fifo_/am_d1_before_d2	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A
5	sva/u_fifo_/am_intf_full	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A
6	sva/u_fifo_/am_stable_d1	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A
7	sva/u_fifo_/am_stable_d2	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A

File Status

Id	File	Language	Kind	Full Name
0	fifo.v	verilog	design	/home/onespin/my_labs/fifo_quantify_demo_v2/Step4_ordering_empty_and_full_checks_but_fix_rack_o/rtl/fifo.v
1	fifo_sva.sv	verilog	design	/home/onespin/my_labs/fifo_quantify_demo_v2/Step4_ordering_empty_and_full_checks_but_fix_rack_o/sva/fifo_sva.sv

Quantify on FIFO Example—X

Let’s dig deeper to find out why

```
42      if (!resetn)                                                    0R
43        w_ack <= 1'b1;                                                0R
44      else if (!full)                                                 0R
45        w_ack <= 1'b1;                                                0R
46      else if (full)                                                  0R
47        w_ack <= 1'b0;                                                0R
48
49    assign w_ack_o = w_ack;                                           0R
50    //--- Let's fix the r_ack_o
51    assign r_ack_o = empty ? 1'b0 : 1'b1;                             1
52    assign w_hsk = w_valid_i && w_ack_o;                              0R
53    assign r_hsk = r_valid_i && r_ack_o;                              1
54    assign nxt_wptr  = wptr + w_hsk;                                  1
55    assign nxt_rptr  = rptr + r_hsk;                                  1
56    assign nxt_empty = (empty || r_hsk) && !w_hsk && (nxt_rptr == nxt_wptr);  1
57  //--- Registered calculations for empty, wptr and rptr
58    always @(posedge clk or negedge resetn)
59     if (!resetn)                                                     1
60       begin
61         empty <= 1'b1;                                               1
62         wptr  <= {DEPTH_BITS{1'b0}};                                 1
63         rptr  <= {DEPTH_BITS{1'b0}};                                 1
64       end
65     else                                                             1
66       begin
67         empty <= nxt_empty;                                          1
68         wptr  <= nxt_wptr;                                           1
69         rptr  <= nxt_rptr;                                           1
70       end
71  //--- Write the data on a w_hsk
72    always @(posedge clk)
73      if (w_hsk)                                                     1
74        data[wptr]     <= data_i;                                    1
75  //--- Read the data on a r_hsk
76    always @(posedge clk)
77      if (r_hsk)                                                     1
78        data_int <= data[rptr];                                      1
79    assign full  = !empty && (rptr == wptr);                          1
80    assign empty_o = empty;                                           1
81    assign full_o  = full;                                            1
82    assign data_o  = data_int;                                        1
83  endmodule
```

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?

Structural Coverage Overview

Status		Statements		Branches	
1	covered	20	90.91%		
R	reached	0	0.00%		0.00%
U	unknown	0	0.00%	0	0.00%
0R	unobserved	2	9.09%		
0	uncovered	0	0.00%		0.00%
0C	constrained	0	0.00%	0	0.00%
0D	dead	0	0.00%	0	0.00%
Sum	quantify targets	22		7	

90.91% design covered

Still 9.09% design unobserved

Excluded Code Overview

Code Status		Statements		Branches	
Xu	excluded by user	0	0.00%	0	0.00%
Xr	excluded redundant code	0	0.00%	0	0.00%
Xv	excluded verification code	14	38.89%	4	36.36%
0/1/U	quantify targets	22	61.11%	7	63.64%
Sum	total code	36		11	

Structural Coverage by File

File	Statements	Branches
fifo.v	22	7
fifo_sva.sv	14	4

Assertion Coverage

Id	Property	Kind	Proof Result	Proof Radius	Cover Result	Cover Radius	Quantified
0	sva/u_fifo_/as_empty_after_reset	assert	FORMAL_PROOF	infinite	COVER_PASS	1	yes
1	sva/u_fifo_/as_empty_to_full	assert	FORMAL_PROOF	infinite	COVER_PASS	1	yes
2	sva/u_fifo_/as_full_to_empty	assert	FORMAL_PROOF	infinite	COVER_PASS	5	yes
3	sva/u_fifo_/as_ordering_check	assert	FORMAL_PROOF	infinite	COVER_PASS	2	yes
4	sva/u_fifo_/as_rhsk_infinitely_often	assert	FORMAL_PROOF	infinite	COVER_PASS	2	yes
5	sva/u_fifo_/as_whsk_infinitely_often	assert	FORMAL_PROOF	infinite	COVER_PASS	2	yes
6	sva/u_fifo_/am_d1_before_d2	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A
7	sva/u_fifo_/am_fair_rvalid	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A
8	sva/u_fifo_/am_fair_wvalid	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A
9	sva/u_fifo_/am_intf_full	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A
10	sva/u_fifo_/am_stable_d1	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A
11	sva/u_fifo_/am_stable_d2	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A

File Status

Id	File	Language	Kind	Full Name
0	fifo.v	verilog	design	/home/onespin/my_labs/fifo_quantify_demo_v3/Step5/rtl/fifo.v
1	fifo_sva.sv	verilog	design	/home/onespin/my_labs/fifo_quantify_demo_v3/Step5/sva/fifo_sva.sv

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?

43	`if (!resetn)`	0R
44	`w_ack <= 1'b1;`	0R
45	`else if (!full)`	1
46	`w_ack <= 1'b1;`	1
47	`else if (full)`	0R
48	`w_ack <= 1'b0;`	0R

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

```
33
34      //--- Interface contraints
35      am_intf_full:    assume property (full_o  |-> !w_hsk || r_hsk);
36
```

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!

Session Setup File Edit CC/MV EC Tools Window Help

Design Explorer | Lint Browser | Auto Checks | Dead-Code Checks | Assertion Checks

Proof Status: mixed Validity: up to date

Instance: [top]

Name	Proof Status	Witness Status	Validity
Assertions			
sva/u_fifo_/as_empty_after_reset	hold	pass (1)	up_to_date
sva/u_fifo_/as_empty_to_full	fail (1)	pass (1)	up_to_date
sva/u_fifo_/as_full_to_empty	hold	pass (5)	up_to_date
sva/u_fifo_/as_ordering_check	fail (8)	pass (2)	up_to_date
sva/u_fifo_/as_rhsk_infinitely_often	hold	pass (2)	up_to_date
sva/u_fifo_/as_whsk_infinitely_often	hold	pass (2)	up_to_date
Constraints			

11 items total, 11 selected by filter

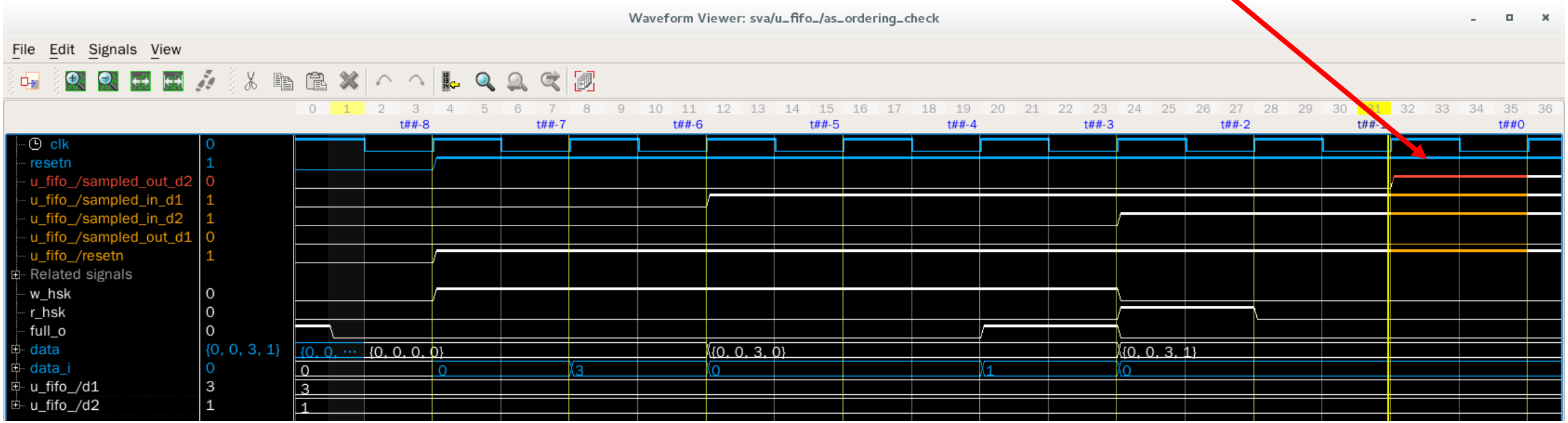
Shell | Messages | Progress

```
-I- Computing witness for 'sva/u_fifo_/as_rhsk_infinitely_often'
-R- Witness computation for 'sva/u_fifo_/as_rhsk_infinitely_often' successful (witness found within 2 cycles from reset) (0.04 sec CPU, 46
-I- Computing witness for 'sva/u_fifo_/as_whsk_infinitely_often'
-R- Witness computation for 'sva/u_fifo_/as_whsk_infinitely_often' successful (witness found within 2 cycles from reset) (0.04 sec CPU, 4
mv>
```

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?

Structural Coverage Overview

Status		Statements		Branches	
1	covered	14	63.64%	3	42.86%
R	reached	0	0.00%	0	0.00%
U	unknown	0	0.00%	0	0.00%
0R	unobserved	8	36.36%		
0	uncovered	0	0.00%	0	0.00%
0C	constrained	0	0.00%	0	0.00%
0D	dead	0	0.00%	0	0.00%
Sum	quantify targets	22		7	

36.36% unobserved

Excluded Code Overview

Code Status		Statements		Branches	
Xu	excluded by user	0	0.00%	0	0.00%
Xr	excluded redundant code	0	0.00%	0	0.00%
Xv	excluded verification code	14	38.89%	4	36.36%
0/1/U	quantify targets	22	61.11%	7	63.64%
Sum	total code	36		11	

Structural Coverage by File

File	Statements	Branches
fifo.v	22	7
fifo_sva.sv	14	4

Assertion Coverage

Id	Property	Kind	Proof Result	Proof Radius	Cover Result	Cover Radius	Quantified
0	sva/u_fifo_/as_empty_after_reset	assert	FORMAL_PROOF	infinite	COVER_PASS	1	yes
1	sva/u_fifo_/as_empty_to_full	assert	FORMAL_NONE	0	COVER_PASS	1	witness
2	sva/u_fifo_/as_full_to_empty	assert	FORMAL_PROOF	infinite	COVER_PASS	5	yes
3	sva/u_fifo_/as_ordering_check	assert	FORMAL_NONE	0			
4	sva/u_fifo_/as_rhsk_infinitely_often	assert	FORMAL_PROOF	infinite	COVER_PASS	2	yes
5	sva/u_fifo_/as_whsk_infinitely_often	assert	FORMAL_PROOF	infinite	COVER_PASS	2	yes
6	sva/u_fifo_/am_d1_before_d2	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A
7	sva/u_fifo_/am_fair_rvalid	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A
8	sva/u_fifo_/am_fair_wvalid	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A
9	sva/u_fifo_/am_stable_d1	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A
10	sva/u_fifo_/am_stable_d2	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A

NO PROOF

NO PROOF

File Status

Id	File	Language	Kind	Full Name
0	fifo.v	verilog	design	/home/onespin/my_labs/fifo_quantify_demo_v2/Step4_looknig_for_bugs_over_constr/rtl/fifo.v
1	fifo_sva.sv	verilog	design	/home/onespin/my_labs/fifo_quantify_demo_v2/Step4_looknig_for_bugs_over_constr/sva/fifo_sva.sv

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

```
40    assign w_ack_o   = full  ? 1'b0 : 1'b1;
41    assign r_ack_o   = empty ? 1'b0 : 1'b1;
```

Quantify on FIFO Example—XVIII

```
40    assign w_ack_o   = full  ? 1'b0 : 1'b1;                                  1
41    assign r_ack_o   = empty ? 1'b0 : 1'b1;                                  1
42    assign w_hsk     = w_valid_i && w_ack_o;                                 1
43    assign r_hsk     = r_valid_i && r_ack_o;                                 1
44    assign nxt_wptr  = wptr + w_hsk;                                         1
45    assign nxt_rptr  = rptr + r_hsk;                                         1
46    assign nxt_empty = (empty || r_hsk) && !w_hsk && (nxt_rptr == nxt_wptr); 1
47
48  //--- Registered calculations for empty, wptr and rptr
49    always @(posedge clk or negedge resetn)
50     if (!resetn)                                                            1
51       begin
52         empty <= 1'b1;                                                      1
53         wptr  <= {DEPTH_BITS{1'b0}};                                        1
54         rptr  <= {DEPTH_BITS{1'b0}};                                        1
55       end
56     else                                                                    1
57       begin
58         empty <= nxt_empty;                                                 1
59         wptr  <= nxt_wptr;                                                  1
60         rptr  <= nxt_rptr;                                                  1
61       end
62
63  //--- Write the data on a w_hsk
64    always @(posedge clk)
65      if (w_hsk)                                                             1
66        data[wptr]     <= data_i;                                            1
67
68  //--- Read the data on a r_hsk
69    always @(posedge clk)
70      if (r_hsk)                                                             1
71        data_int <= data[rptr];                                              1
72
73    assign full  = !empty && (rptr == wptr);                                 1
74    assign empty_o = empty;                                                  1
75    assign full_o  = full;                                                   1
76    assign data_o  = data_int;                                               1
77  endmodule
```

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?

Structural Coverage Overview

Status		Statements		Branches	
1	covered	19	100.00%	4	100.00%
R	reached	0	0.00%	0	0.00%
U	unknown	0	0.00%	0	0.00%
0R	unobserved	0	0.00%	0	0.00%
0	uncovered	0	0.00%	0	0.00%
0C	constrained	0	0.00%	0	0.00%
0D	dead	0	0.00%	0	0.00%
Sum	**quantify targets**	**19**		**4**	

Excluded Code Overview

Code Status		Statements		Branches	
Xu	excluded by user	0	0.00%	0	0.00%
Xr	excluded redundant code	0	0.00%	0	0.00%
Xv	excluded verification code	14	42.42%	4	50.00%
0/1/U	quantify targets	19	57.58%	4	50.00%
Sum	**total code**	**33**		**8**	

Structural Coverage by File

File	Statements	Branches
fifo.v	19	4
fifo_sva.sv	14	4

Assertion Coverage

Id	Property	Kind	Proof Result	Proof Radius	Cover Result	Cover Radius	Quantified
0	sva/u_fifo_/as_empty_after_reset	assert	FORMAL_PROOF	infinite	COVER_PASS	1	yes
1	sva/u_fifo_/as_empty_to_full	assert	FORMAL_PROOF	infinite	COVER_PASS	1	yes
2	sva/u_fifo_/as_full_to_empty	assert	FORMAL_PROOF	infinite	COVER_PASS	5	yes
3	sva/u_fifo_/as_intf_empty	assert	FORMAL_PROOF	infinite	COVER_PASS	1	yes
4	sva/u_fifo_/as_intf_full	assert	FORMAL_PROOF	infinite	COVER_PASS	5	yes
5	sva/u_fifo_/as_ordering_check	assert	FORMAL_PROOF	infinite	COVER_PASS	2	yes
6	sva/u_fifo_/as_rhsk_infinitely_often	assert	FORMAL_PROOF	infinite	COVER_PASS	2	yes
7	sva/u_fifo_/as_whsk_infinitely_often	assert	FORMAL_PROOF	infinite	COVER_PASS	2	yes
8	sva/u_fifo_/am_d1_before_d2	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A
9	sva/u_fifo_/am_fair_rvalid	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A
10	sva/u_fifo_/am_fair_wvalid	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A
11	sva/u_fifo_/am_stable_d1	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A
12	sva/u_fifo_/am_stable_d2	assume	FORMAL_ASSUMPTION	infinite	N/A	0	N/A

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

Id	Property	Kind	Proof Result	Proof Radius	Cover Result	Cover Radius
0	sva/u_fifo_/as_empty_after_reset	assert	FORMAL_PROOF	infinite	COVER_PASS	1
1	sva/u_fifo_/as_empty_to_full	assert	FORMAL_PROOF	infinite	COVER_PASS	1
2	sva/u_fifo_/as_full_to_empty	assert	FORMAL_PROOF	infinite	COVER_PASS	5
3	sva/u_fifo_/as_intf_empty	assert	FORMAL_PROOF	infinite	COVER_PASS	1
4	sva/u_fifo_/as_intf_full	assert	FORMAL_PROOF	infinite	COVER_PASS	5
5	sva/u_fifo_/as_ordering_check	assert	FORMAL_PROOF	infinite	COVER_PASS	2
6	sva/u_fifo_/as_rhsk_infinitely_often	assert	FORMAL_PROOF	infinite	COVER_PASS	2
7	sva/u_fifo_/as_whsk_infinitely_often	assert	FORMAL_PROOF	infinite	COVER_PASS	2
8	sva/u_fifo_/am_d1_before_d2	assume	FORMAL_ASSUMPTION	infinite	N/A	0
9	sva/u_fifo_/am_fair_rvalid	assume	FORMAL_ASSUMPTION	infinite	N/A	0
10	sva/u_fifo_/am_fair_wvalid	assume	FORMAL_ASSUMPTION	infinite	N/A	0
11	sva/u_fifo_/am_stable_d1	assume	FORMAL_ASSUMPTION	infinite	N/A	0
12	sva/u_fifo_/am_stable_d2	assume	FORMAL_ASSUMPTION	infinite	N/A	0

Quantify on FIFO Example—XXI

We discover additional requirements on this design

Assertion Coverage

Id	Property	Kind	Proof Result	Proof Radius	Cover Result	Cover Radius
0	sva/u_fifo_/as_empty_after_reset	assert	FORMAL_PROOF	infinite	COVER_PASS	1
1	sva/u_fifo_/as_empty_to_full	assert	FORMAL_PROOF	infinite	COVER_PASS	1
2	sva/u_fifo_/as_full_to_empty	assert	FORMAL_PROOF	infinite	COVER_PASS	5
3	sva/u_fifo_/as_intf_empty	assert	FORMAL_PROOF	infinite	COVER_PASS	1
4	sva/u_fifo_/as_intf_full	assert	FORMAL_PROOF	infinite	COVER_PASS	5
5	sva/u_fifo_/as_ordering_check	assert	FORMAL_PROOF	infinite	COVER_PASS	2

```
//    Interface Assertions

as_intf_empty:   assert property (empty_o |-> !r_hsk);
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

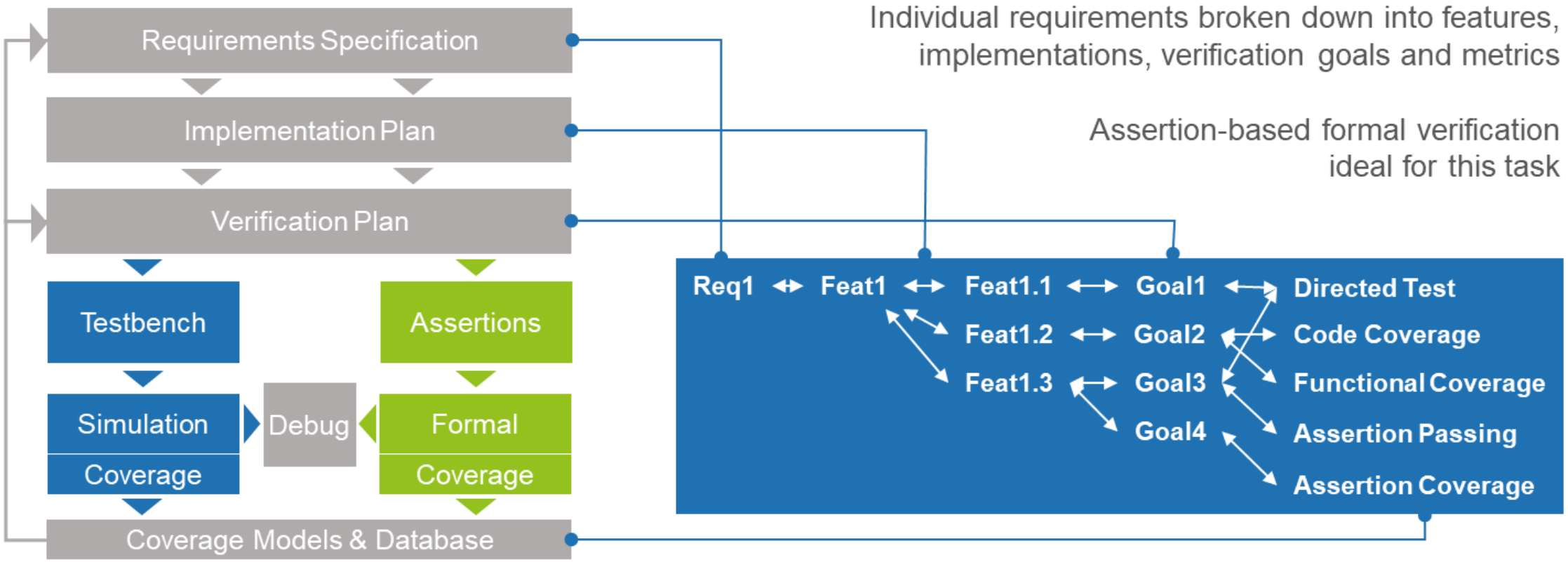
Case Study: Verification of I^2C Serial Protocol Interface

accellera SYSTEMS INITIATIVE

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

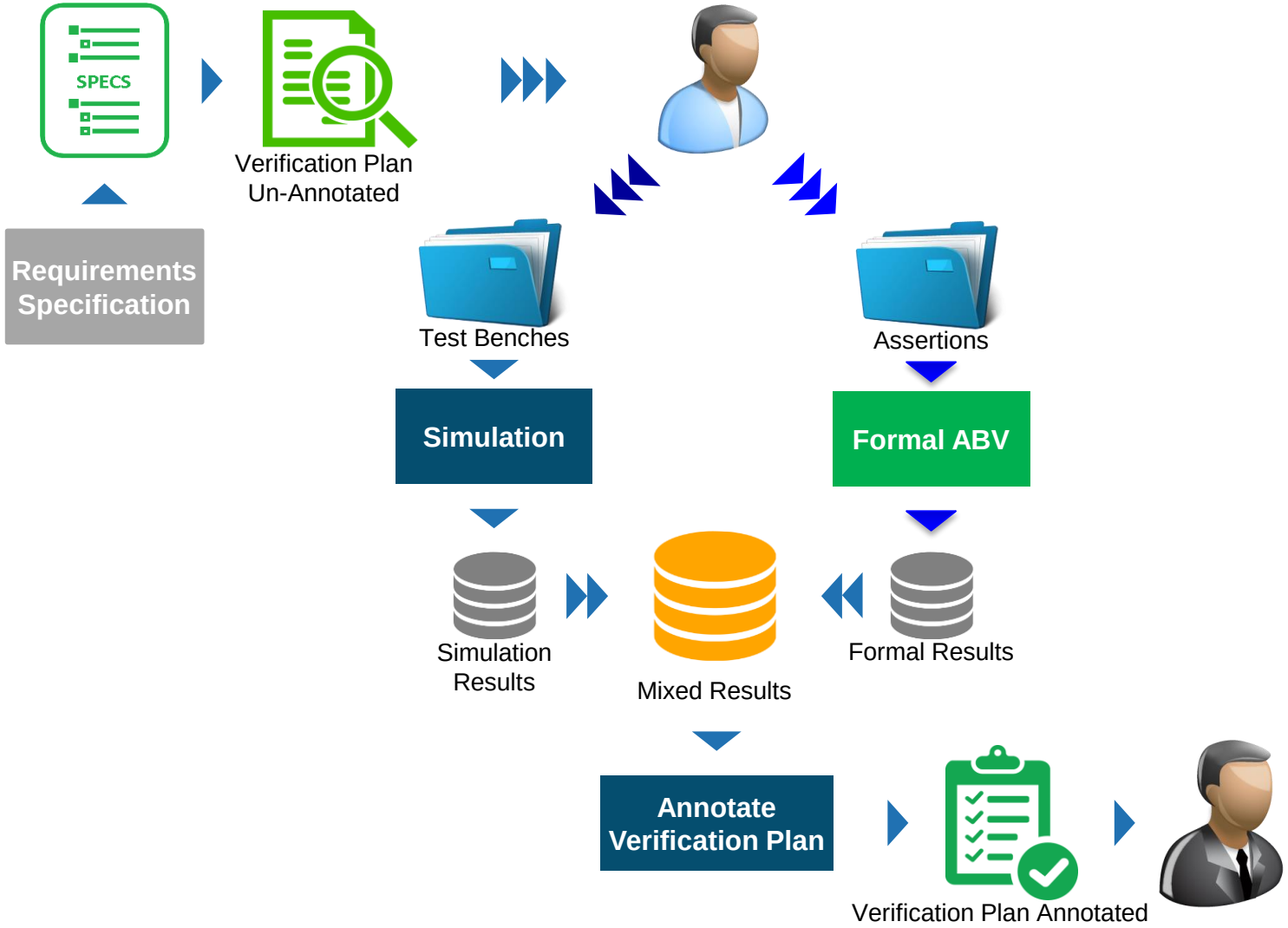
Systematic Verification Flow

Requirement tracing and coverage are of paramount importance



Tracking Progress in the Verification Plan

Integrating formal and simulation verification



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

I^2C, SPI, UART, CAN, etc.

What would be an ideal approach?

Verify protocol compliance at the interfaces binding a VIP

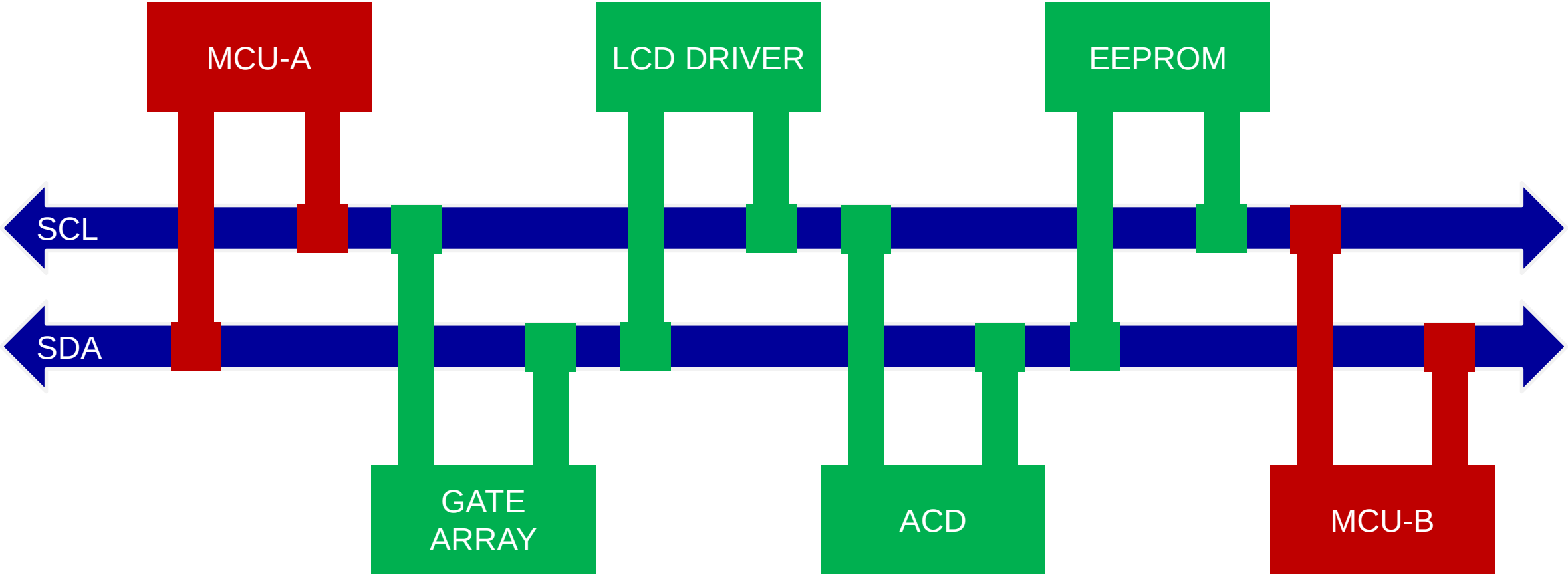
Make use of a scoreboard to check data integrity

What is the challenge?

Even slow SoCs are running at frequencies starting in the range of 10MHz, while I^2C standard-mode speed is up to 100kHz

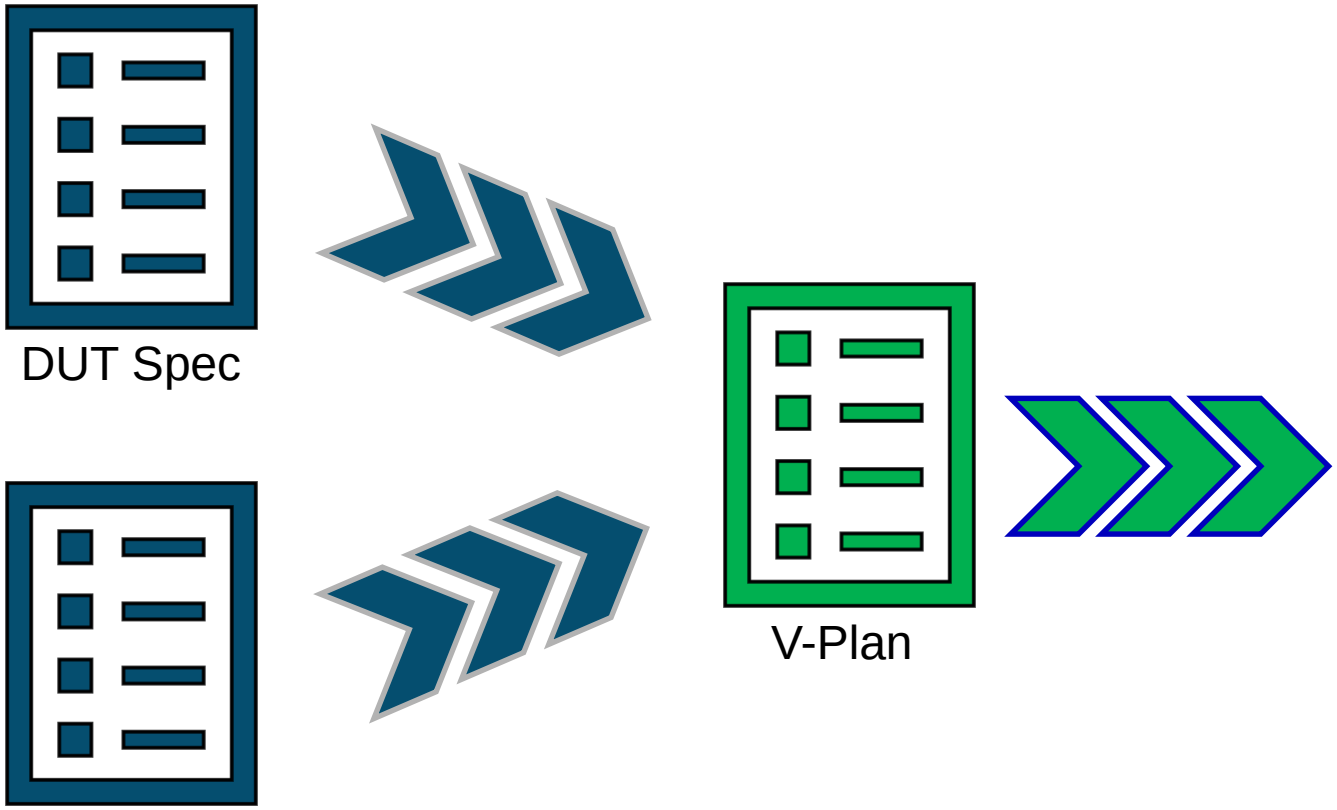
- Do the math: *The formal tool needs to examine many cycles in order to prove that a single byte is transferred correctly.*

I²C Bus Protocol



The Verification Process

Verification plan: what needs to be verified?



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 addressign 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

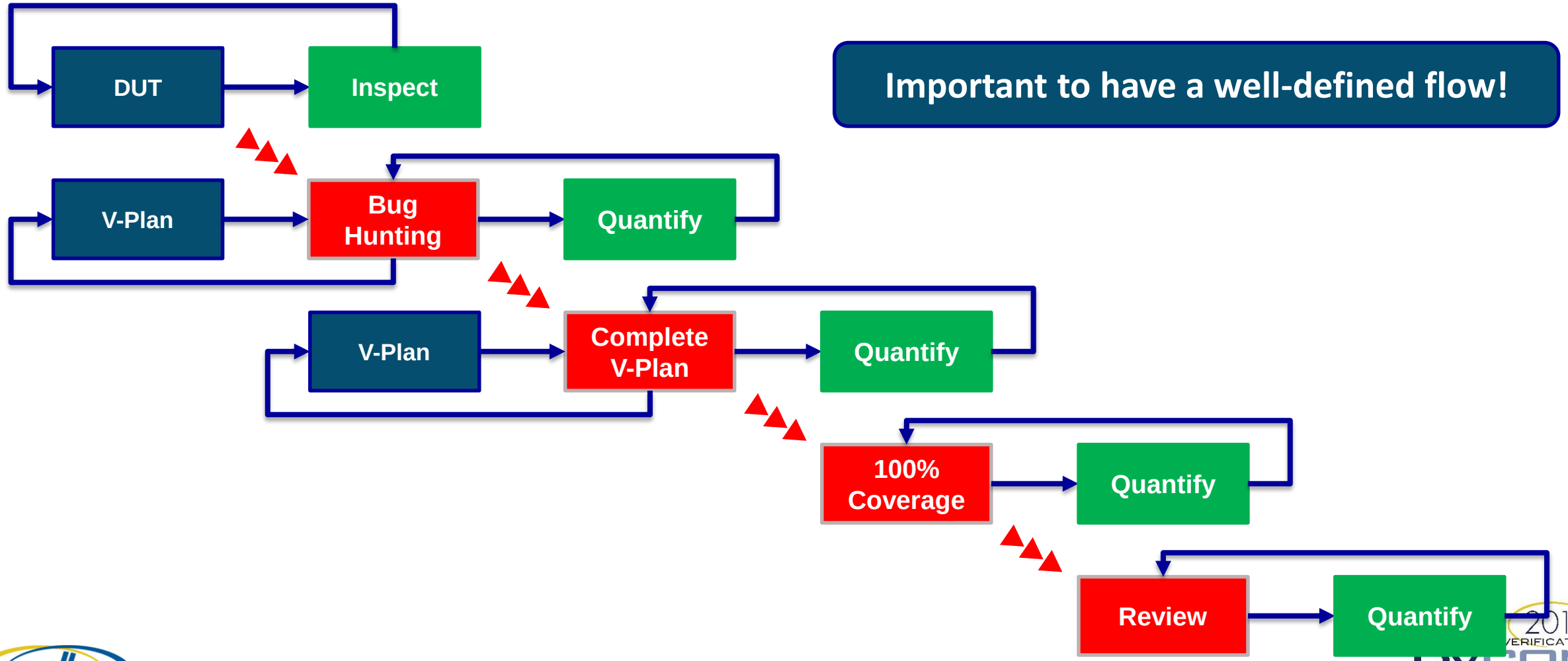
Validate results: are failing checks expected?

Language:	*Verilog*
Primary input signals:	*8 (17 bits)*
Primary output signals:	*3 (10 bits)*
Primary inout signals:	*2 (2 bits)*
State bits (flops):	*128*
Assignments:	*258 (1034 bits)*
Code branches:	*116*
FSMs:	*2*
Adders:	*0*
Multipliers:	*0*
Primary clocks:	*1*

```
full_case checks:          2 |    2 hold,    0 fail,    0 open
parallel_case checks:      3 |    3 hold,    0 fail,    0 open
resolution_x checks:       2 |    0 hold,    2 fail,    0 open
signal_domain checks:      2 |    0 hold,    2 fail,    0 open
init checks:             128 |  118 hold,   10 fail,    0 open
fsm checks:                2 |    2 hold,    0 fail,    0 open
dead_code checks:        134 |  134 hold,    0 fail,    0 open
stick checks:            108 |  106 hold,    2 fail,    0 open
```

The Verification Process

What is the verification approach?



Important to have a well-defined flow!

Quantify Coverage Results

29.10.2017

Quantify MDV Overview

Overview | Structural Coverage Overview | Structural Coverage by File | Assertion Coverage | File Status | Additional Information

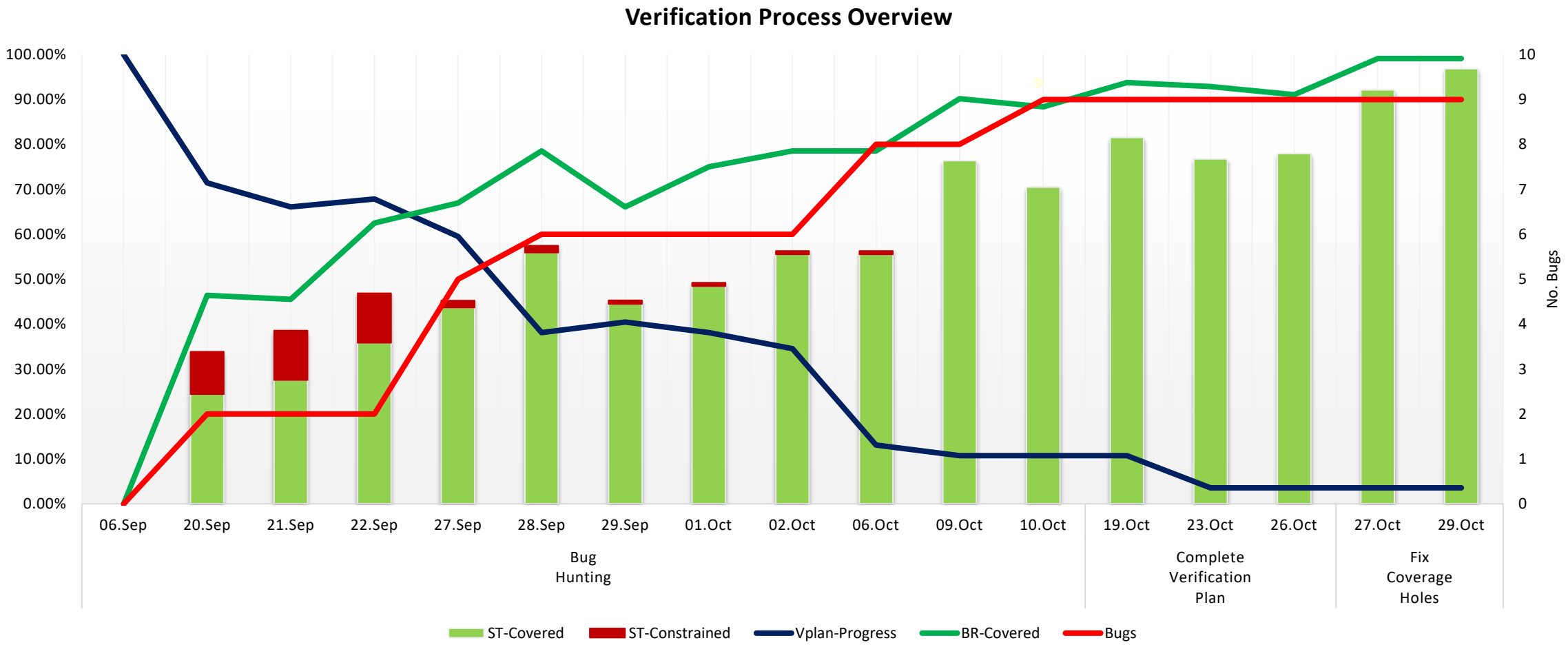
Structural Coverage Overview

Status		Statements		Branches	
1	covered	246	96.85%	111	99.11%
R	reached	8	3.15%	0	0.00%
U	unknown	0	0.00%	1	0.89%
0R	unobserved	0	0.00%	0	0.00%
0	uncovered	0	0.00%	0	0.00%
0C	constrained	0	0.00%	0	0.00%
0D	dead	0	0.00%	0	0.00%
Sum	quantify targets	254		112	

Structural Coverage by File

File	Statements		Branches	
i2c_master_bit_ctrl.v	133		48	
i2c_master_byte_ctrl.v	65		36	
i2c_master_top.v	56		28	

Tracking Progress Over Time



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 );
/*****************************************************************/
```

```
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
```

Quantify Coverage Results

Detection of over-constrained code

```
/*****************************************************************/
/* 29 SEP */
/*****************************************************************/

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

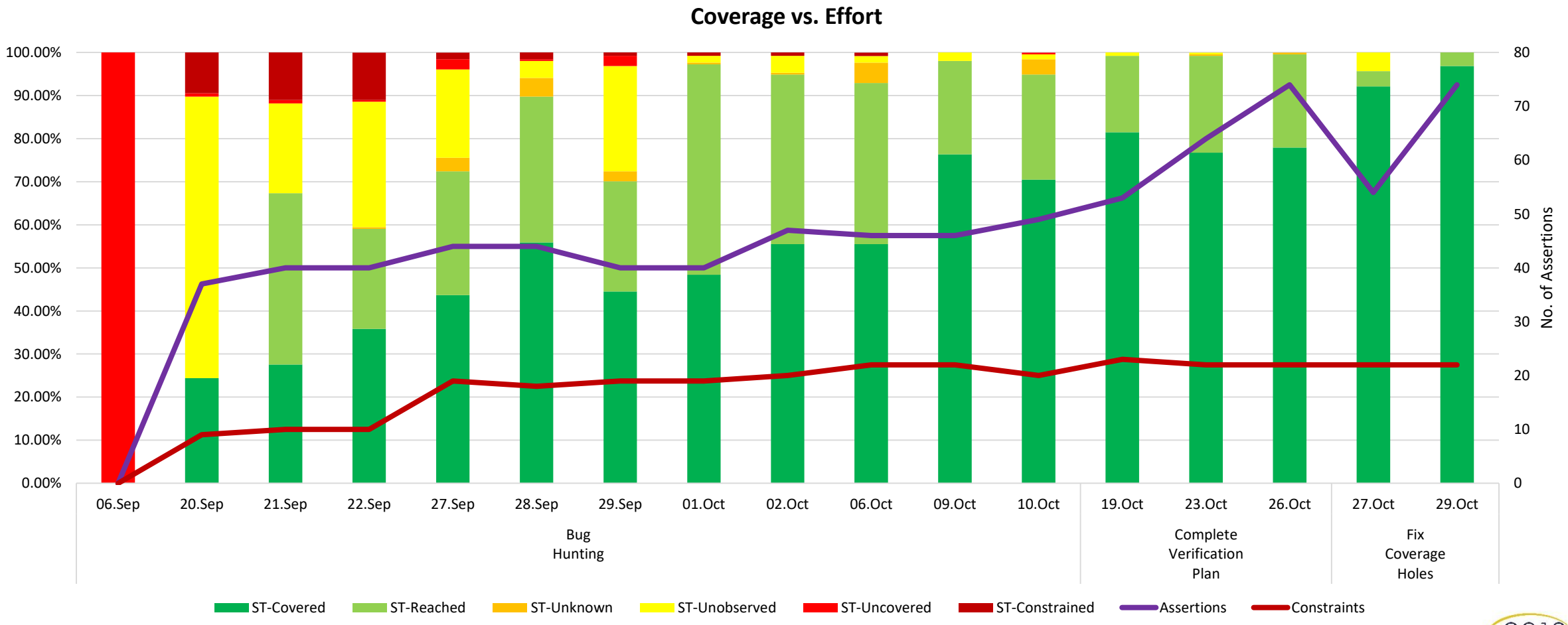
Void message [UM10204-Notes Page.14]

START immediately followed by a STOP is an illegal format

```
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
```

Tracking Progress Over Time

Coverage vs effort



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

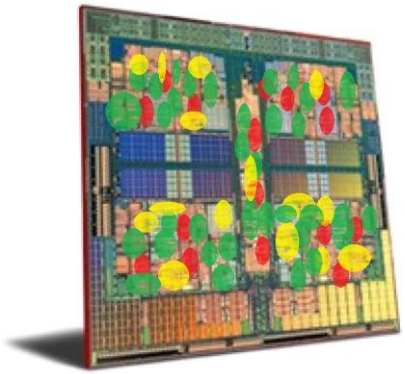
Why does the approach matter?

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



Design	#Code Lines	#Assertions	Runtime
FIFO	321	30	100s
FSM-DDR2-Read	839	6	106s
vCore-Processor	295	8	204s
Arithmetic Block	383	2	257s

Interactive use on single modules to improve verification

Design	#Code Lines	#Assertions
IFX-Aurix-1	25563	85
IFX-Aurix-2	27374	157
IFX-Aurix-3	57253	253

Real example at Infineon

Quantify identified verification holes and guided assertion development.
New assertions detected critical bugs.

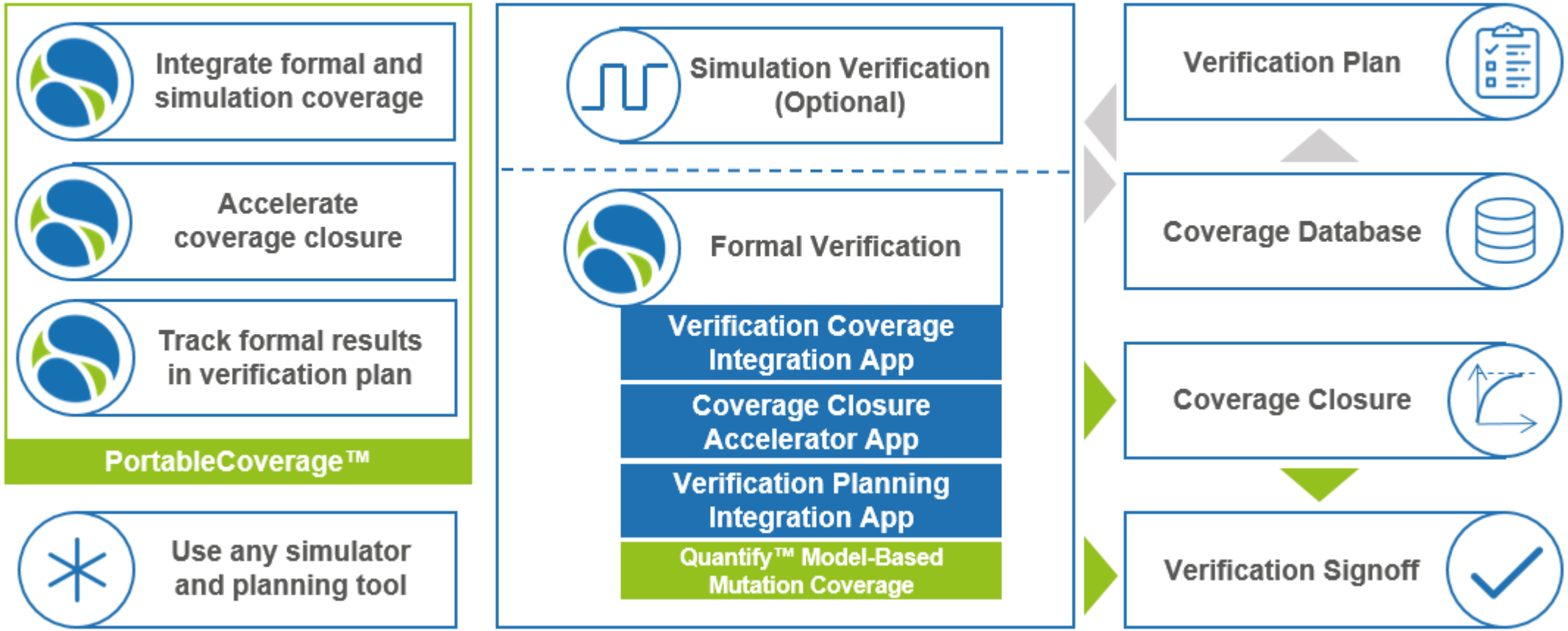
Quantify now used to provide management metrics on all designs!

Formal Safety Verification with Qualified Property Sets

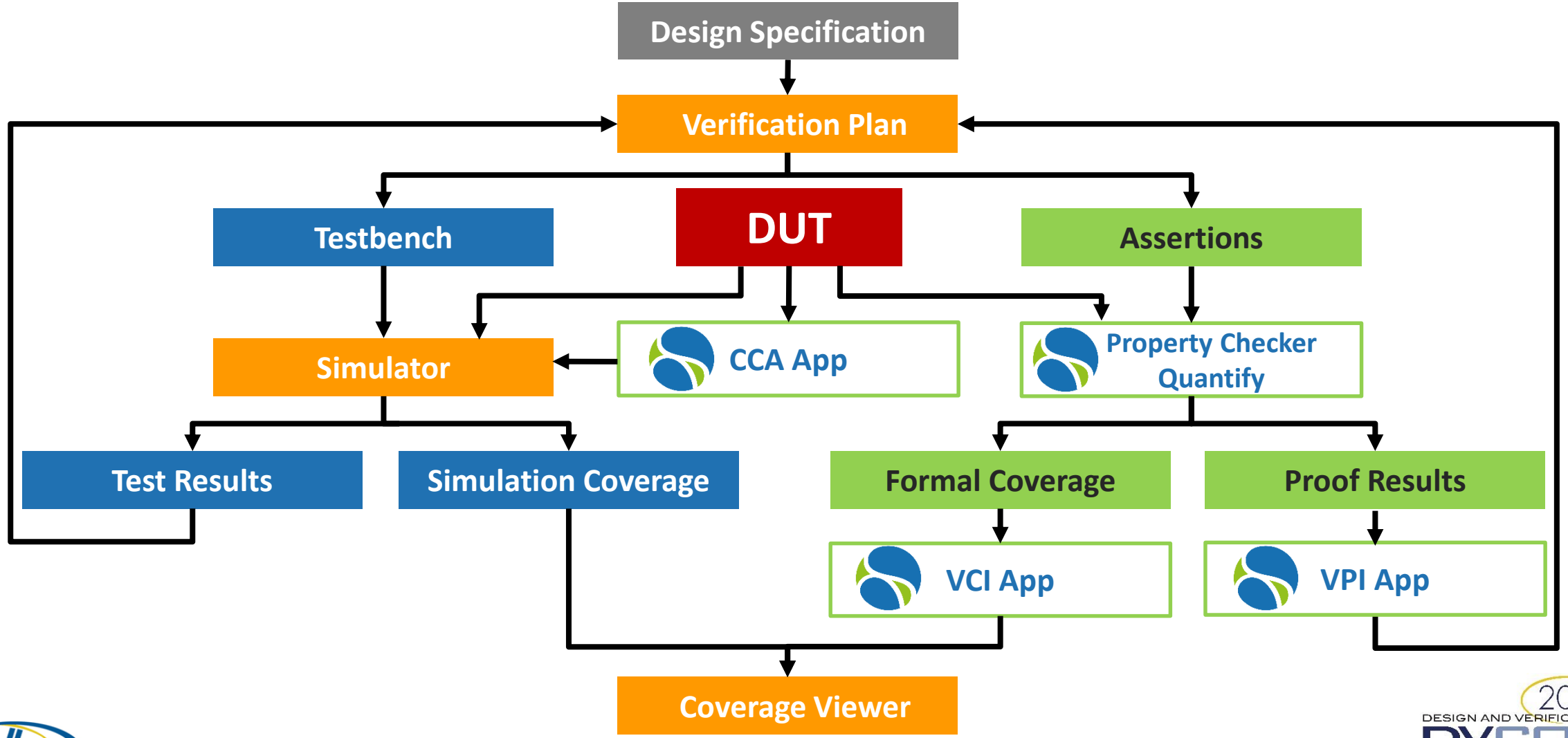
Holger Busch at DAC'14 in Accelerating Productivity Through Formal and Static Methods (Session 38.3)

Interoperable Coverage Solution

PortableCoverage



Formal-Simulation Seamless Integration



Side-by-Side Analysis of Coverage Contributions

Verification Coverage Integration (VCI) App

Simulation Coverage

Aldec Coverage Report

file:///C:/Users/nt88/Documents/results_22112017/Aldec_SimReport/index.html

Design

Cumulative : SC 48% : BC

- work.i2c_master_top : 7
- work.i2c_master_byte_c
- work.i2c_master_bit_ctr

Hierarchy | Units

- work.i2c_master_byte_ctrl
  - @ASSIGN#167_0@
  - @ASSIGN#170_1@
  - @ALWAYS#173_2@
  - @ALWAYS#184_3@
  - @ASSIGN#194_4@
  - @ALWAYS#201_5@

Statement | Branch

Summary - work.i2c_master_byte_ctrl | Statement - work.i2c_master_b

```
202  1   if (!nReset)
203      begin
204  1       core_cmd <=  `I2C_CMD_NOP;
205  1       core_txd <=  1'b0;
206  1       shift    <=  1'b0;
207  1       ld       <=  1'b0;
208  1       cmd_ack  <=  1'b0;
209  1       c_state  <=  ST_IDLE;
210  1       ack_out  <=  1'b0;
211      end
212  1   else if (rst | i2c_al)
213      begin
214  0       core_cmd <=  `I2C_CMD_NOP;
215  0       core_txd <=  1'b0;
216  0       shift    <=  1'b0;
217  0       ld       <=  1'b0;
218  0       cmd_ack  <=  1'b0;
219  0       c_state  <=  ST_IDLE;
220  0       ack_out  <=  1'b0;
221      end
222  else
223      begin
224          // initially reset all signals
225  1       core_txd <=  sr[7];
226  1       shift    <=  1'b0;
227  1       ld       <=  1'b0;
228  1       cmd_ack  <=  1'b0;
229
230  1       case (c_state) // synopsys full_case paral
231          ST_IDLE:
232  1         if (go)
233            begin
234  1           if (start)
235              begin
236  1             c_state  <=  ST_START;
237  1             core_cmd <=  `I2C_CMD_STAR
238              end
239  0           else if (read)
240              begin
241  0             c_state  <=  ST_READ;
242  0             core_cmd <=  `I2C_CMD_READ
243              end
244  0           else if (write)
245              begin
```

Quantify Formal Coverage

VCI App

Aldec Coverage Report

file:///C:/Users/nt88/Documents/results_22112017/QTF_Riviera_Report/index.html

Design

Cumulative : SC 100% : BC

- work.i2c_master_top : 1
- work.i2c_master_byte_c
- work.i2c_master_bit_ctr

Hierarchy | Units

- work.i2c_master_byte_ctrl
  - @ASSIGN#167_0@
  - @ASSIGN#170_1@
  - @ALWAYS#173_2@
  - @ALWAYS#184_3@
  - @ASSIGN#194_4@
  - @ALWAYS#201_5@

Statement | Branch

Summary - work.i2c_master_byte_ctrl | Statement - work.i2c_master_

```
202  *1  if (!nReset)
203      begin
204  1       core_cmd <=  `I2C_CMD_NOP;
205  1       core_txd <=  1'b0;
206  1       shift    <=  1'b0;
207  1       ld       <=  1'b0;
208  1       cmd_ack  <=  1'b0;
209  1       c_state  <=  ST_IDLE;
210  1       ack_out  <=  1'b0;
211      end
212  *1  else if (rst | i2c_al)
213      begin
214  1       core_cmd <=  `I2C_CMD_NOP;
215  1       core_txd <=  1'b0;
216  1       shift    <=  1'b0;
217  1       ld       <=  1'b0;
218  1       cmd_ack  <=  1'b0;
219  1       c_state  <=  ST_IDLE;
220  1       ack_out  <=  1'b0;
221      end
222  else
223      begin
224          // initially reset all signals
225  (1)     core_txd <=  sr[7];
226  (1)     shift    <=  1'b0;
227  (1)     ld       <=  1'b0;
228  (1)     cmd_ack  <=  1'b0;
229
230  (1)     case (c_state) // synopsys full_case
231          ST_IDLE:
232  (1)       if (go)
233            begin
234  (1)         if (start)
235              begin
236  (1)             c_state  <=  ST_STAR
237  (1)             core_cmd <=  `I2C_CM
238              end
239  (0)         else if (read)
240              begin
241  (0)             c_state  <=  ST_READ
242  (0)             core_cmd <=  `I2C_CM
243              end
244  (0)         else if (write)
245              begin
```

accellera SYSTEMS INITIATIVE

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

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
http://testandverification.com/DVClub/18_Nov_2013/Infineon-HolgerBusch.pdf

Thank you!

Questions?