2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

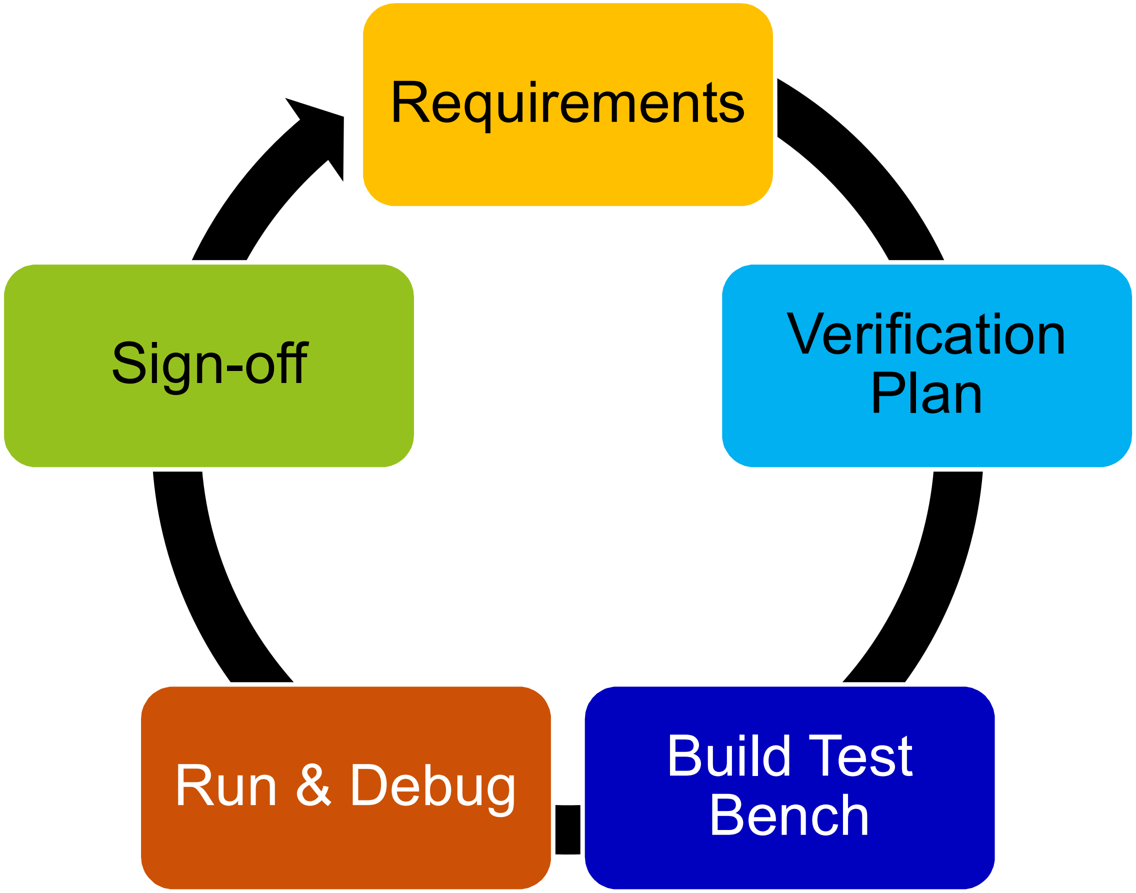
Using Mutation Coverage for Advanced Bug Hunting

Vladislav Palfy & Nicolae Tusinschi—OneSpin Solutions

onespin

making electronics reliable

The Verification Loop



Assessing Quality of Verification

If you don’t measure, you don’t know

When am I done?

- Have I written enough **stimuli** to cover all requirements?
- What part of the design has been **exercised** by my assertions/covers?
- Have I written **good quality** checks?
- **Which parts** of the design have been checked by my checkers?

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

Quantify MDV Overview

Multi-dimensional view—quantity and quality

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

Quantify

- ❖ Takes as input a hardware design and a formal test bench
- ❖ One push of a button produces a metric-driven sign-off report as output

Structural Coverage (Quantity)

- ❖ Control & Observation Coverage—provides quantitative assessment

Functional Coverage (Quality)

- ❖ Assertion Coverage—provides qualitative assessment

Coverage Solution: Provides Meaningful Metrics

Continuous feedback for design and verification

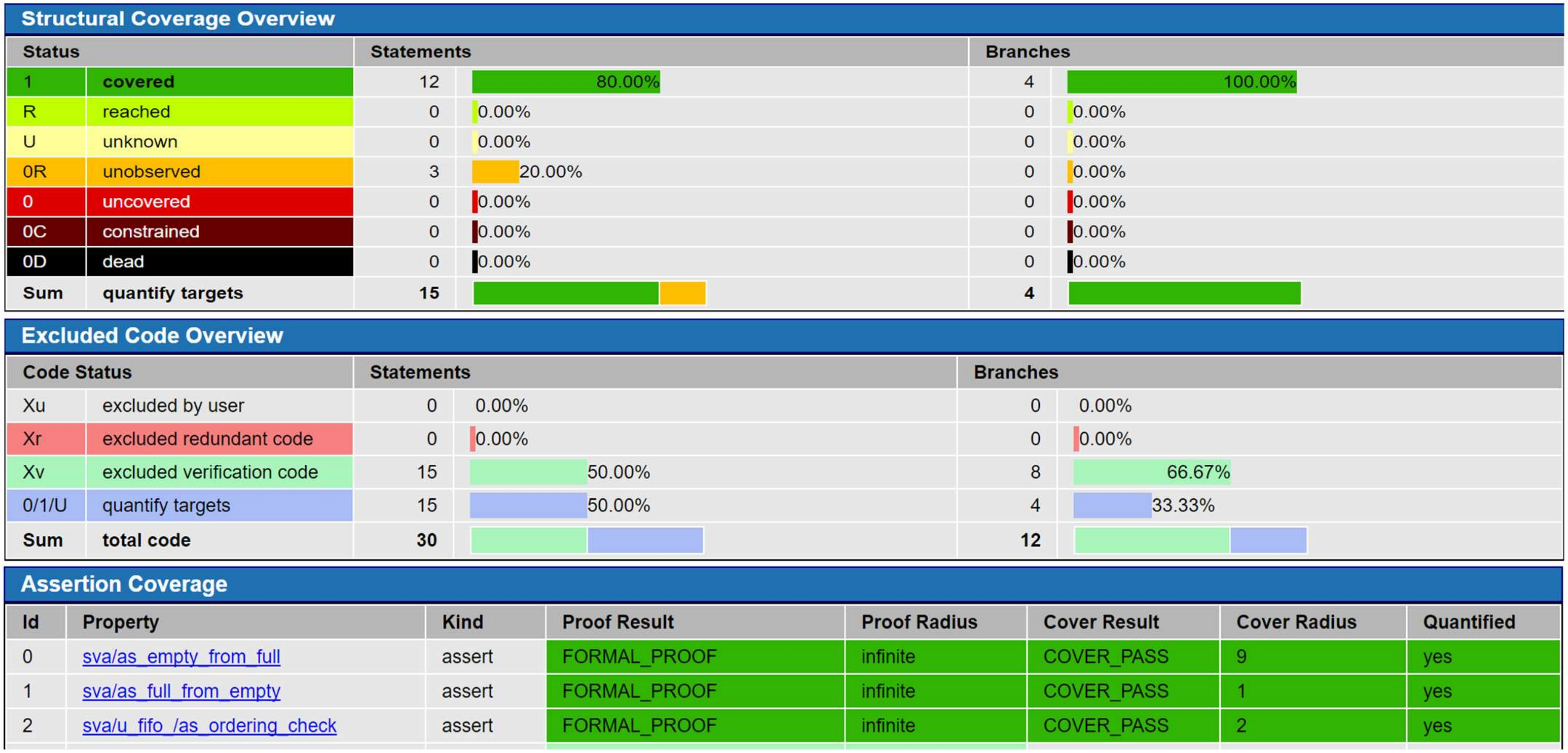
- **Designer Bring Up: Get feedback on the quality of design bring up**
  - Dead code; reachability
  - Redundant code
- **Verification: When quality and quantity both matter**
  - Metrics should indicate gaps in verification and show you where these are
    - Missing checks
    - Over-constraints
    - Find bugs

Quantify Report

Color-coded highway to sign-off

	Result	Meaning
Controllability	Reached	Reached by an Assertion
	Constrained	Unreachable & Unobservable Due to Constraints
	Dead	Unreachable and Unobservable
Observability	Uncovered	Not Reached and Not Observed
	Covered	Observed and Reached
	Unobserved	Reached and Not Observed

Quantify Dashboard View: Important 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

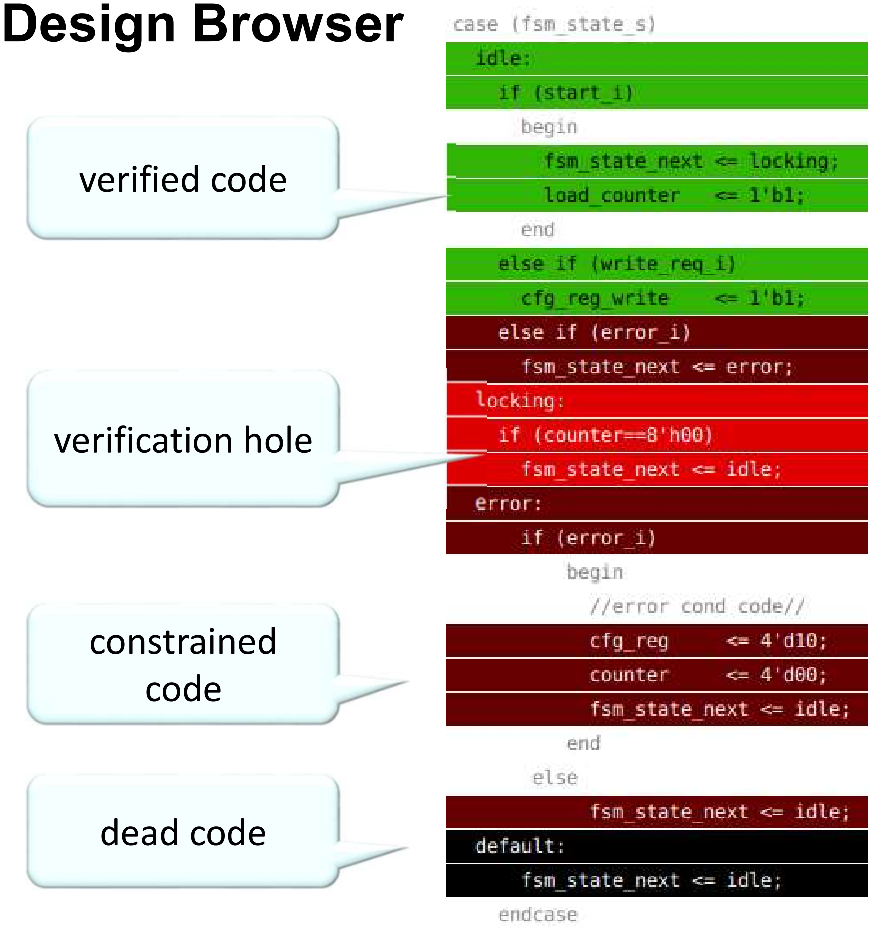
Code Status		Statements		Branches	
Xu	excluded by user	0	0.00%	0	0.00%
Xr	excluded redundant code	0	0.00%	0	0.00%
Xv	excluded verification code	15	50.00%	8	66.67%
0/1/U	quantify targets	15	50.00%	4	33.33%
Sum	**total code**	**30**		**12**	

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

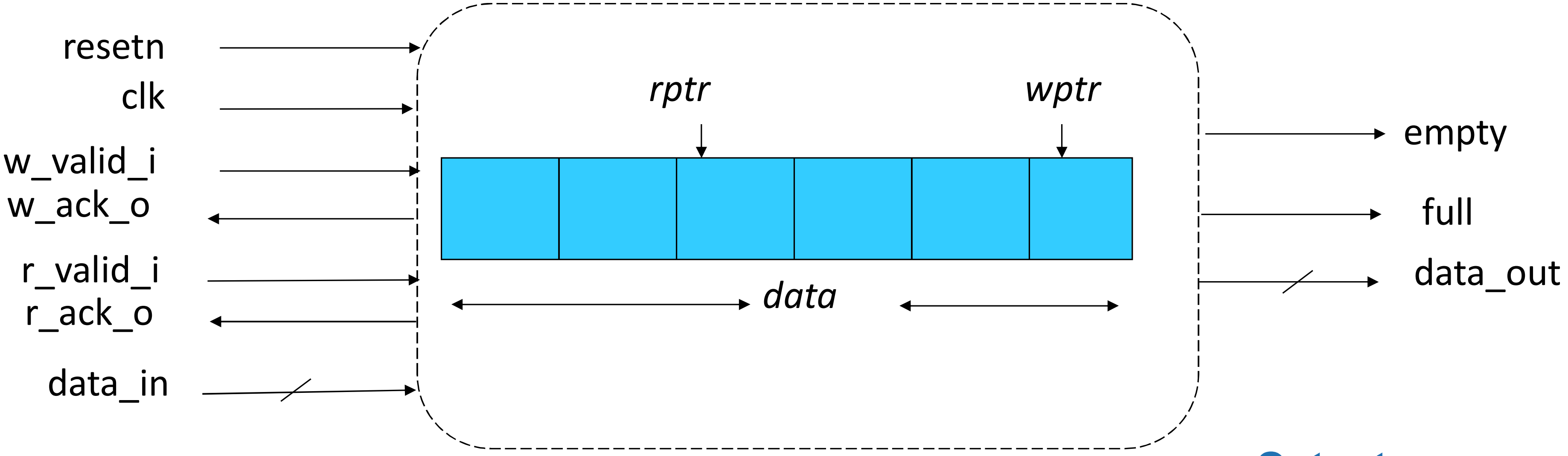
```
case (fsm_state_s)
  idle:
    if (start_i)
      begin
        fsm_state_next <= locking;
        load_counter   <= 1'b1;
      end
    else if (write_req_i)
      cfg_reg_write    <= 1'b1;
    else if (error_i)
      fsm_state_next <= error;
  locking:
    if (counter==8'h00)
      fsm_state_next <= 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
```

Quantify in Action

FIFO Example

FIFO

Interface of this design



Input

ABCDEFGH..

Output

ABCDEFGH.. ✓

ABCDFE GH.. ✗

ABC EFGH.. ✗

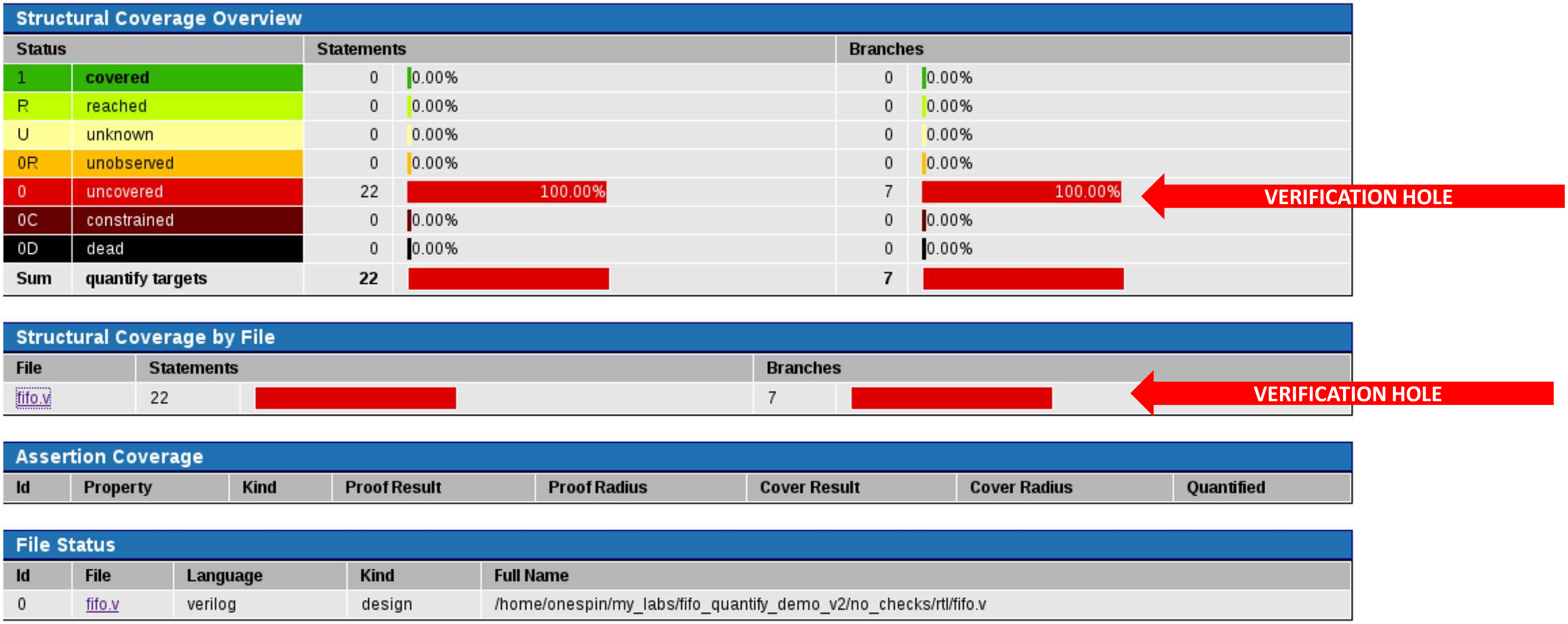
ABCDDEFGH.. ✗

Requirements for Verification

- Ordering is correct
- No duplication
- No data loss
- No data corruption
- Empty and full checks
  - ❖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 checks 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

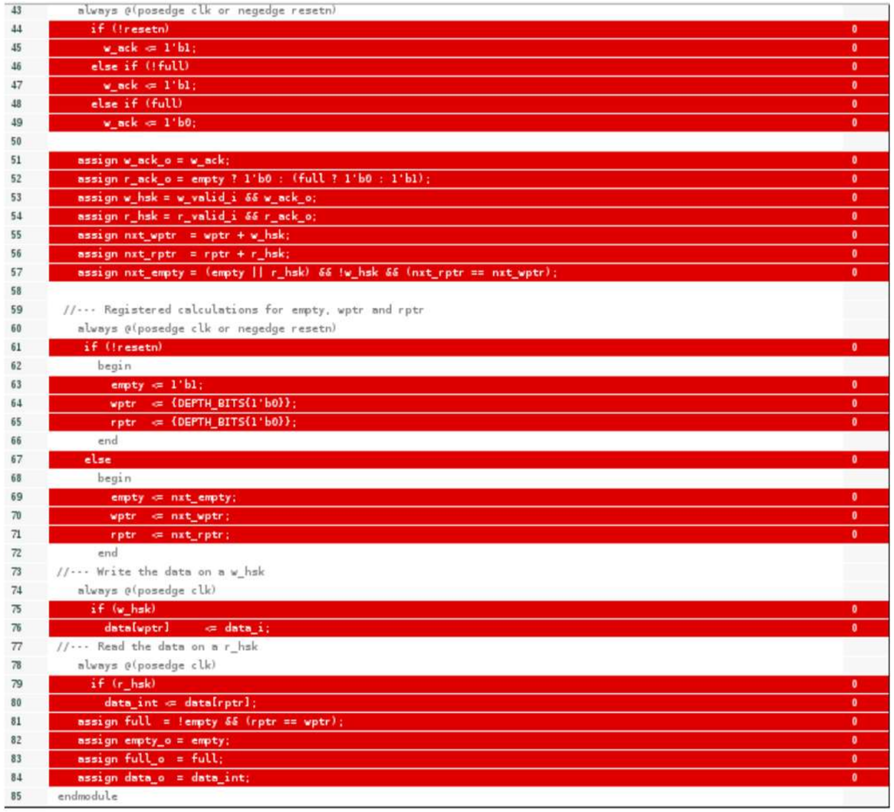
Id	Property	Kind	Proof Result	Proof Radius	Cover Result	Cover Radius	Quantified

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



```
    always @(posedge clk or negedge resetn)
      if (!resetn)                                                          0
        w_ack <= 1'b1;                                                      0
      else if (!full)                                                       0
        w_ack <= 1'b1;                                                      0
      else if (full)                                                        0
        w_ack <= 1'b0;                                                      0

    assign w_ack_o = w_ack;                                                 0
    assign r_ack_o = empty ? 1'b0 : (full ? 1'b0 : 1'b1);                   0
    assign w_hsk = w_valid_i && w_ack_o;                                    0
    assign r_hsk = r_valid_i && r_ack_o;                                    0
    assign nxt_wptr  = wptr + w_hsk;                                        0
    assign nxt_rptr  = rptr + r_hsk;                                        0
    assign nxt_empty = (empty || r_hsk) && !w_hsk && (nxt_rptr == nxt_wptr); 0

  //--- Registered calculations for empty, wptr and rptr
    always @(posedge clk or negedge resetn)
     if (!resetn)                                                           0
       begin
         empty <= 1'b1;                                                     0
         wptr  <= {DEPTH_BITS{1'b0}};                                       0
         rptr  <= {DEPTH_BITS{1'b0}};                                       0
       end
     else                                                                   0
       begin
         empty <= nxt_empty;                                                0
         wptr  <= nxt_wptr;                                                 0
         rptr  <= nxt_rptr;                                                 0
       end
 //--- Write the data on a w_hsk
    always @(posedge clk)
      if (w_hsk)                                                            0
        data[wptr]     <= data_i;                                           0
 //--- Read the data on a r_hsk
    always @(posedge clk)
      if (r_hsk)                                                            0
        data_int <= data[rptr];                                             0
    assign full  = !empty && (rptr == wptr);                                0
    assign empty_o = empty;                                                 0
    assign full_o  = full;                                                  0
    assign data_o  = data_int;                                              0
 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 check

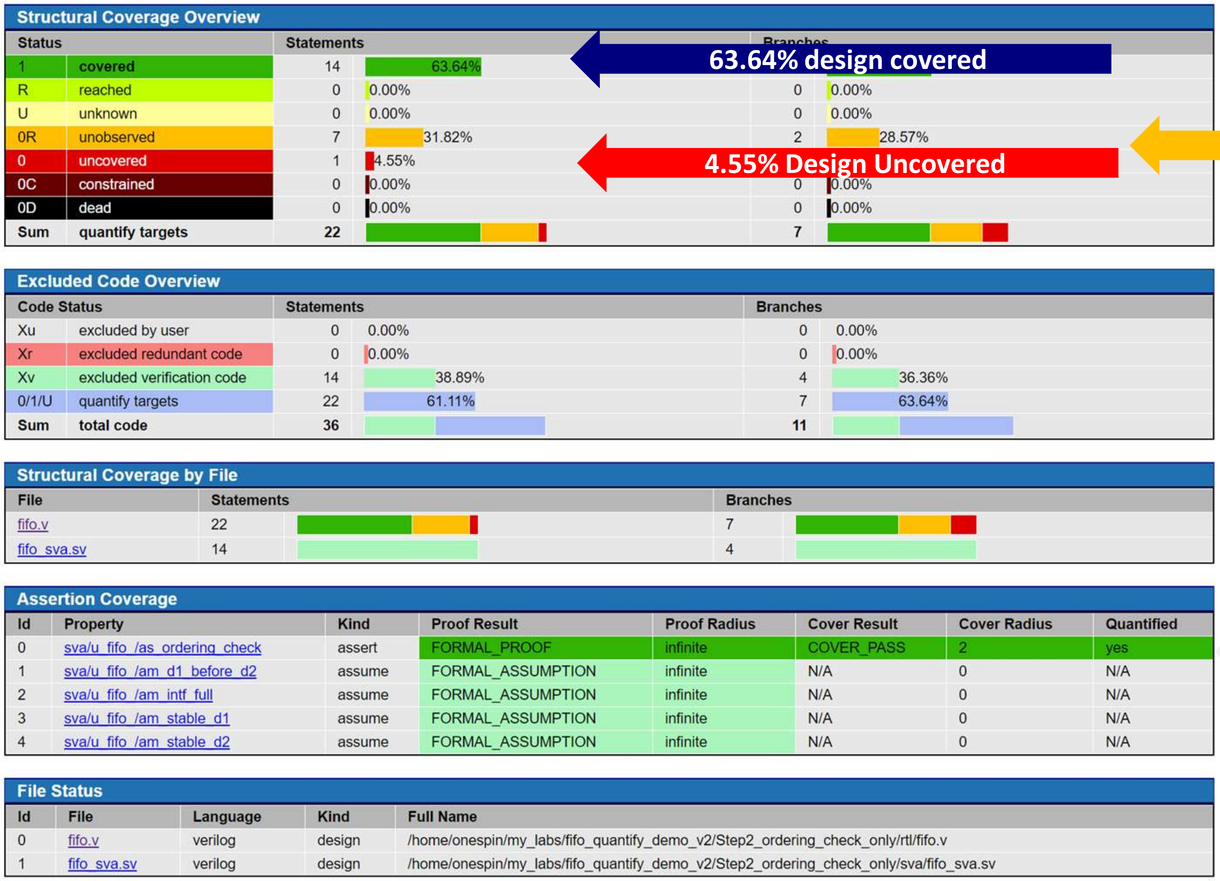
FIFO Ordering Check

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 check
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 check



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

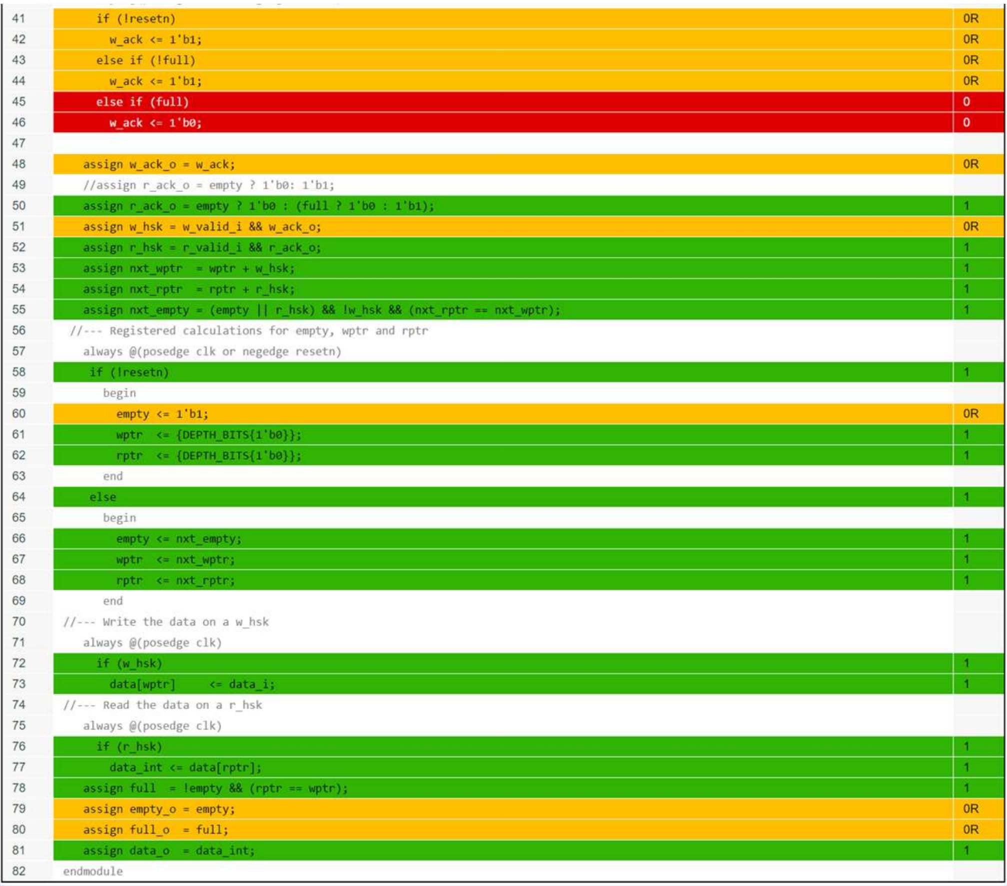
Single Check

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’s still missing?



```
      if (!resetn)                                                  0R
        w_ack <= 1'b1;                                              0R
      else if (!full)                                               0R
        w_ack <= 1'b1;                                              0R
      else if (full)                                                0
        w_ack <= 1'b0;                                              0

    assign w_ack_o = w_ack;                                         0R
    //assign r_ack_o = empty ? 1'b0: 1'b1;
    assign r_ack_o = empty ? 1'b0 : (full ? 1'b0 : 1'b1);           1
    assign w_hsk = w_valid_i && w_ack_o;                            0R
    assign r_hsk = r_valid_i && r_ack_o;                            1
    assign nxt_wptr  = wptr + w_hsk;                                1
    assign nxt_rptr  = rptr + r_hsk;                                1
    assign nxt_empty = (empty || r_hsk) && !w_hsk && (nxt_rptr == nxt_wptr);   1
  //--- Registered calculations for empty, wptr and rptr
    always @(posedge clk or negedge resetn)
     if (!resetn)                                                   1
       begin
         empty <= 1'b1;                                             0R
         wptr  <= {DEPTH_BITS{1'b0}};                               1
         rptr  <= {DEPTH_BITS{1'b0}};                               1
       end
     else                                                           1
       begin
         empty <= nxt_empty;                                        1
         wptr  <= nxt_wptr;                                         1
         rptr  <= nxt_rptr;                                         1
       end
 //--- Write the data on a w_hsk
    always @(posedge clk)
      if (w_hsk)                                                    1
        data[wptr]     <= data_i;                                   1
 //--- Read the data on a r_hsk
    always @(posedge clk)
      if (r_hsk)                                                    1
        data_int <= data[rptr];                                     1
    assign full  = !empty && (rptr == wptr);                        1
    assign empty_o = empty;                                         0R
    assign full_o  = full;                                          0R
    assign data_o  = data_int;                                      1
 endmodule
```

Missing coverage

- ❖ Unobserved
- ❖ Uncovered

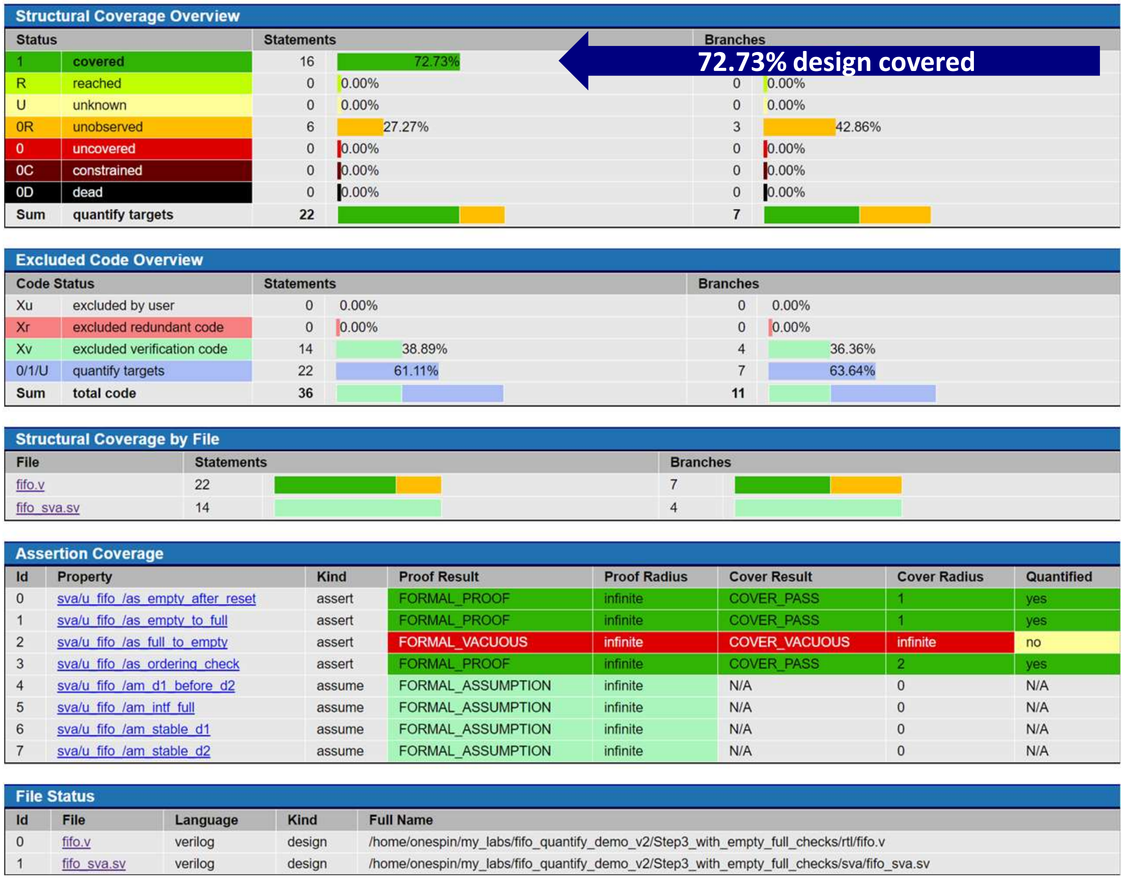
Quantify on FIFO Example—V

Let’s add checks on empty and full

```
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

How did we do now?



72.73% design covered

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	

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

File Status

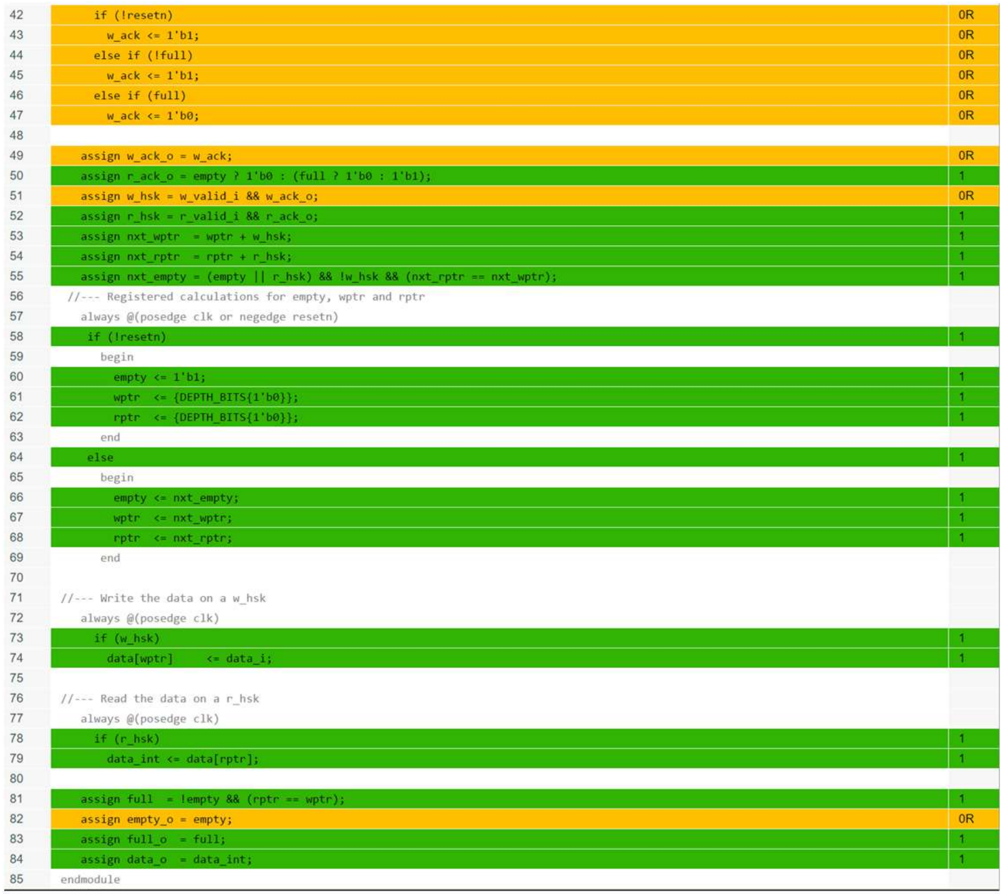
Id	File	Language	Kind	Full Name
0	fifo.v	verilog	design	/home/onespin/my_labs/fifo_quantify_demo_v2/Step3_with_empty_full_checks/rtl/fifo.v
1	fifo_sva.sv	verilog	design	/home/onespin/my_labs/fifo_quantify_demo_v2/Step3_with_empty_full_checks/sva/fifo_sva.sv

Vacuous Failure

Problem with debugging unreachables

Quantify on FIFO Example—VII

Where are the missing coverage targets?



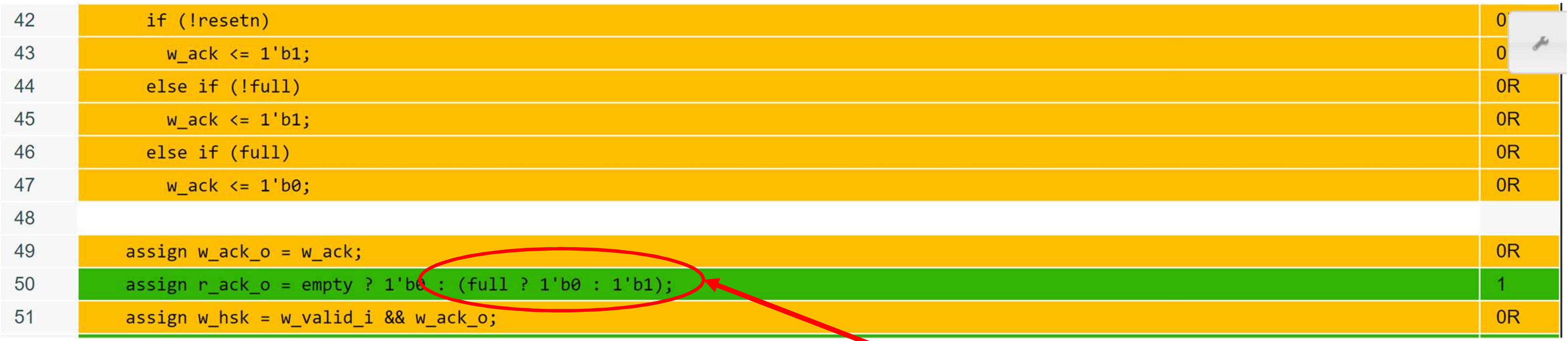
```
      if (!resetn)                                                  0R
        w_ack <= 1'b1;                                              0R
      else if (!full)                                               0R
        w_ack <= 1'b1;                                              0R
      else if (full)                                                0R
        w_ack <= 1'b0;                                              0R

    assign w_ack_o = w_ack;                                         0R
    assign r_ack_o = empty ? 1'b0 : (full ? 1'b0 : 1'b1);           1
    assign w_hsk = w_valid_i && w_ack_o;                            0R
    assign r_hsk = r_valid_i && r_ack_o;                            1
    assign nxt_wptr  = wptr + w_hsk;                                1
    assign nxt_rptr  = rptr + r_hsk;                                1
    assign nxt_empty = (empty || r_hsk) && !w_hsk && (nxt_rptr == nxt_wptr);   1
  //--- Registered calculations for empty, wptr and rptr
    always @(posedge clk or negedge resetn)
     if (!resetn)                                                   1
       begin
         empty <= 1'b1;                                             1
         wptr  <= {DEPTH_BITS{1'b0}};                               1
         rptr  <= {DEPTH_BITS{1'b0}};                               1
       end
     else                                                           1
       begin
         empty <= nxt_empty;                                        1
         wptr  <= nxt_wptr;                                         1
         rptr  <= nxt_rptr;                                         1
       end

  //--- Write the data on a w_hsk
    always @(posedge clk)
      if (w_hsk)                                                    1
        data[wptr]     <= data_i;                                   1

  //--- Read the data on a r_hsk
    always @(posedge clk)
      if (r_hsk)                                                    1
        data_int <= data[rptr];                                     1

    assign full  = !empty && (rptr == wptr);                        1
    assign empty_o = empty;                                         0R
    assign full_o  = full;                                          1
    assign data_o  = data_int;                                      1
  endmodule
```

Missing coverage

- ❖ Unobserved code
- ❖ Cannot observe empty!

Quantify on FIFO Example—VIII

A closer look

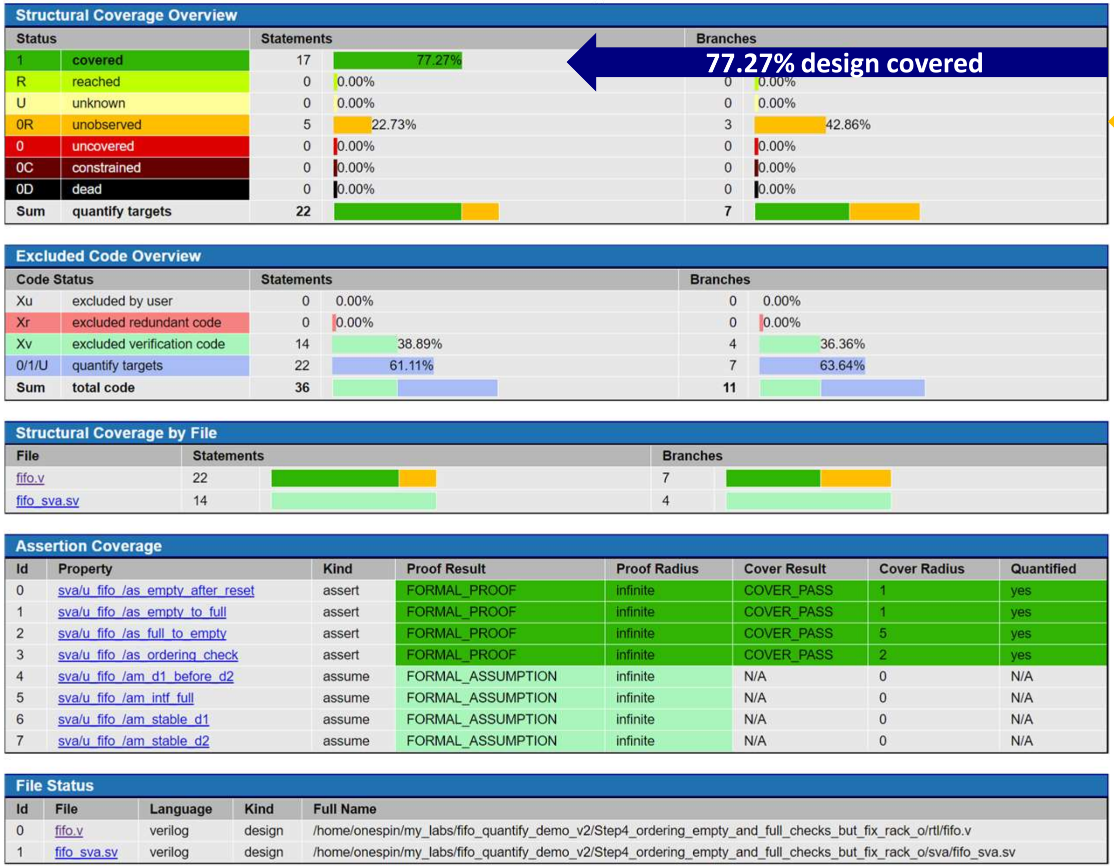


```
      if (!resetn)                                   0
        w_ack <= 1'b1;                               0
      else if (!full)                                0R
        w_ack <= 1'b1;                               0R
      else if (full)                                 0R
        w_ack <= 1'b0;                               0R

    assign w_ack_o = w_ack;                          0R
    assign r_ack_o = empty ? 1'b0 : (full ? 1'b0 : 1'b1);   1
    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 bug fix on r_ack_o design



77.27% design covered

Still 22.7% design unobserved

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**	

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_/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

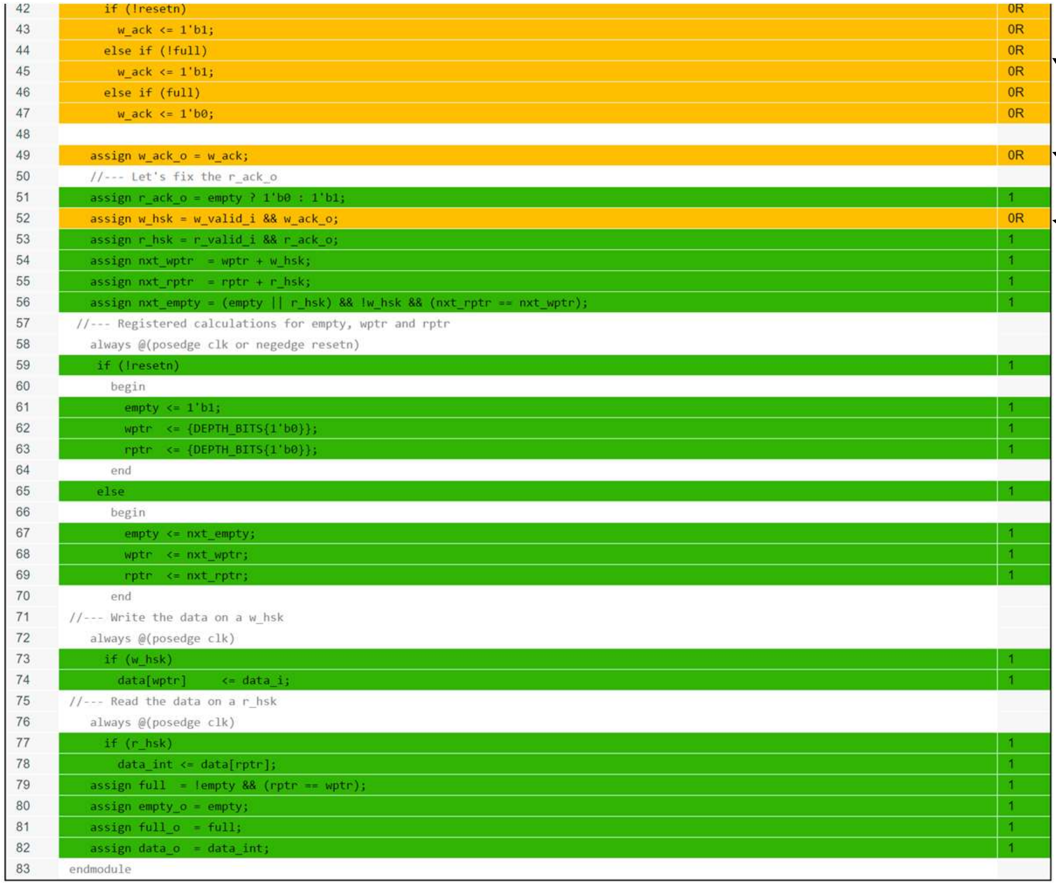
File Status

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

Design coverage increased to 77.27%
But still missing 22.73%!

Quantify on FIFO Example—X

Let’s dig deeper to find out why



```
     if (!resetn)                                                        0R
       w_ack <= 1'b1;                                                    0R
     else if (!full)                                                     0R
       w_ack <= 1'b1;                                                    0R
     else if (full)                                                      0R
       w_ack <= 1'b0;                                                    0R

   assign w_ack_o = w_ack;                                               0R
   //--- Let's fix the r_ack_o
   assign r_ack_o = empty ? 1'b0 : 1'b1;                                 1
   assign w_hsk = w_valid_i && w_ack_o;                                  0R
   assign r_hsk = r_valid_i && r_ack_o;                                  1
   assign nxt_wptr  = wptr + w_hsk;                                      1
   assign nxt_rptr  = rptr + r_hsk;                                      1
   assign nxt_empty = (empty || r_hsk) && !w_hsk && (nxt_rptr == nxt_wptr);   1
 //--- Registered calculations for empty, wptr and rptr
   always @(posedge clk or negedge resetn)
    if (!resetn)                                                         1
      begin
        empty <= 1'b1;                                                   1
        wptr  <= {DEPTH_BITS{1'b0}};                                     1
        rptr  <= {DEPTH_BITS{1'b0}};                                     1
      end
    else                                                                 1
      begin
        empty <= nxt_empty;                                              1
        wptr  <= nxt_wptr;                                               1
        rptr  <= nxt_rptr;                                               1
      end
//--- Write the data on a w_hsk
   always @(posedge clk)
     if (w_hsk)                                                          1
       data[wptr]     <= data_i;                                         1
//--- Read the data on a r_hsk
   always @(posedge clk)
     if (r_hsk)                                                          1
       data_int <= data[rptr];                                           1
   assign full  = !empty && (rptr == wptr);                              1
   assign empty_o = empty;                                               1
   assign full_o  = full;                                                1
   assign data_o  = data_int;                                            1
endmodule
```

Missing coverage on w_ack and w_hsk

Unobserved code

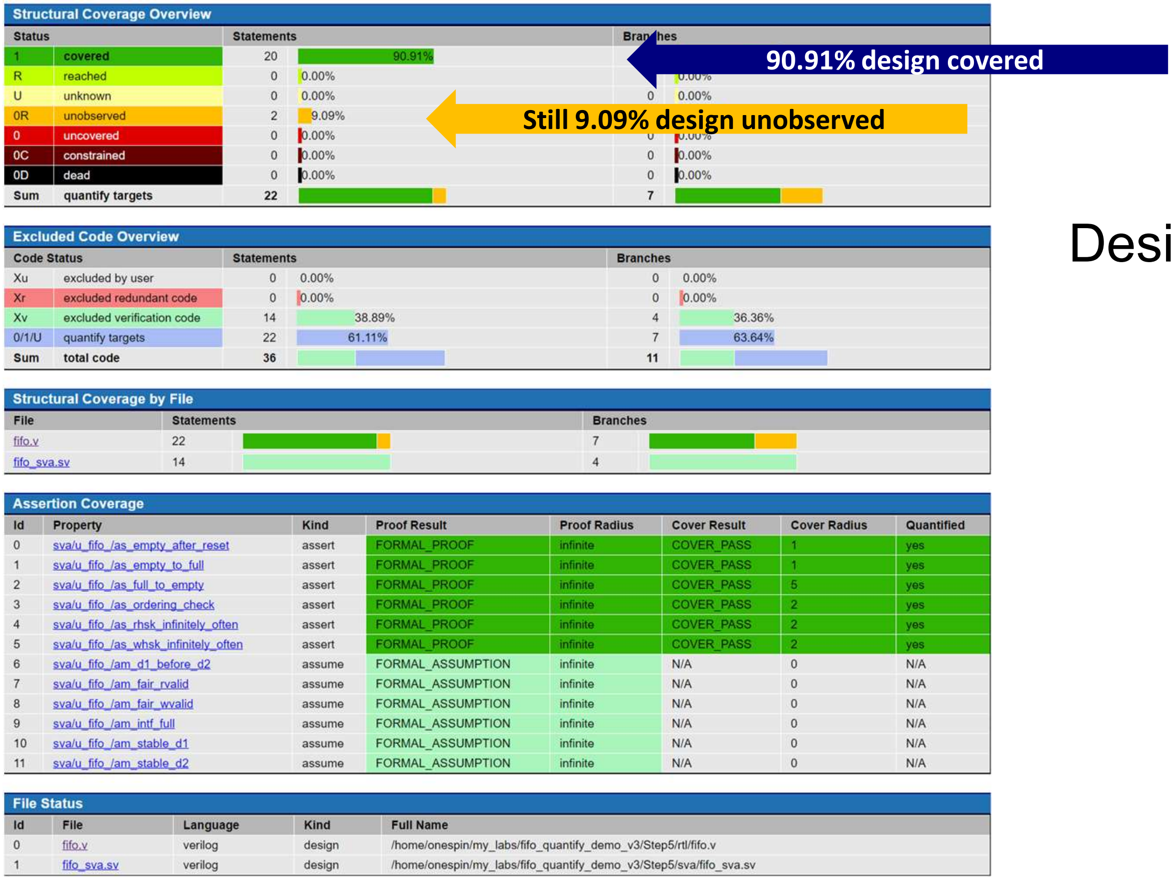
Quantify on FIFO Example—XI

Let’s go add the remainder checks

```
//-- 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 checks
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%	[illegible]	[illegible]
R	reached	0	0.00%	[illegible]	0.00%
U	unknown	0	0.00%	0	0.00%
0R	unobserved	2	9.09%	[illegible]	[illegible]
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**	

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

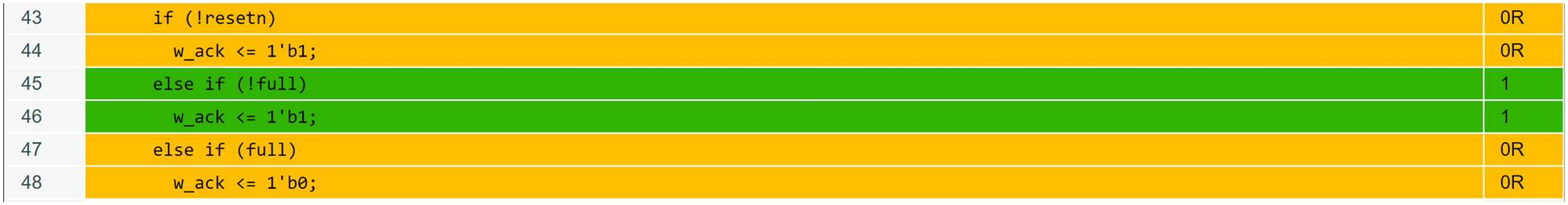
File Status

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

Design coverage increased to 90.91%

Quantify on FIFO Example—XIII

So what's going on now?



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.

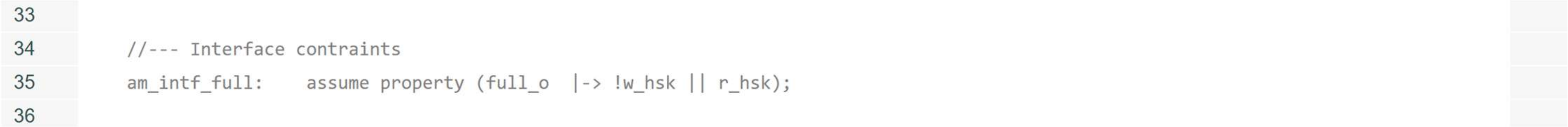
However, we only delay the write in the following cycle.

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



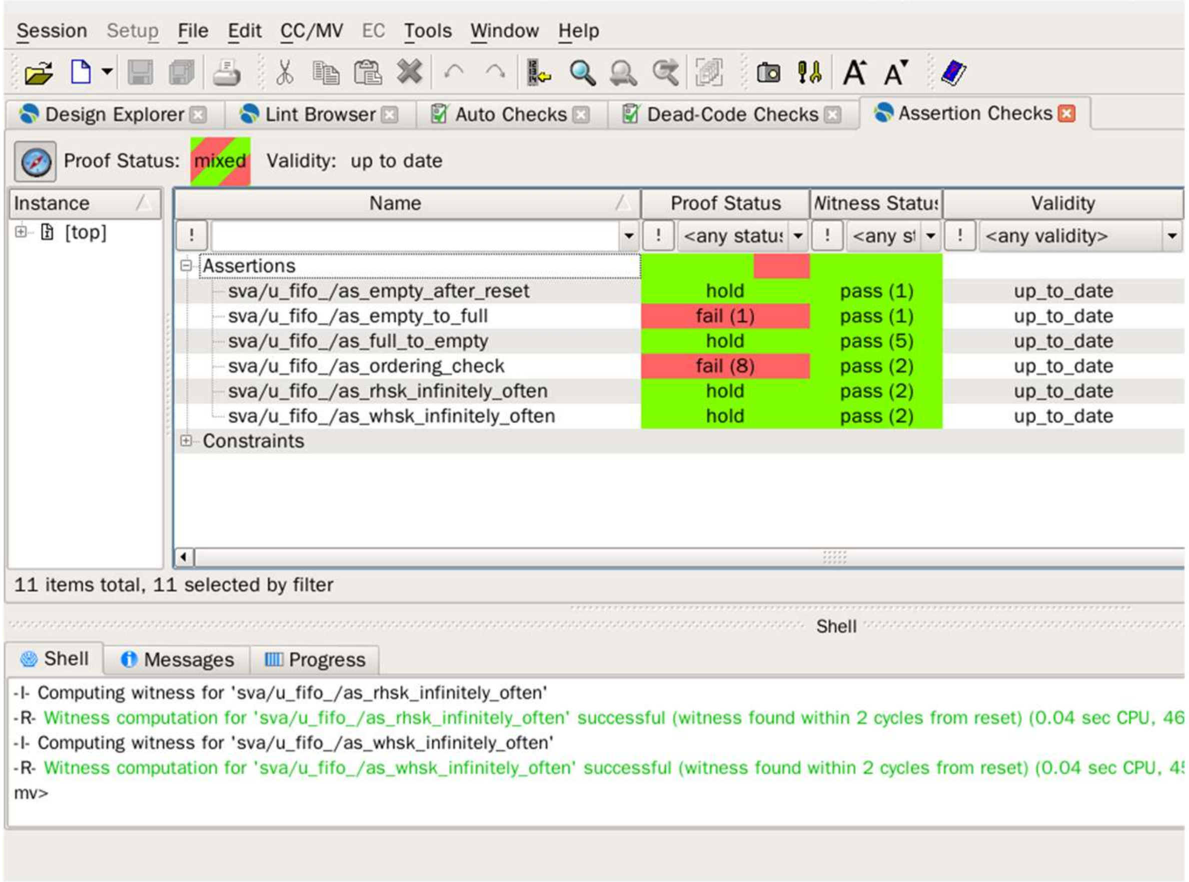
```
//--- Interface contraints
am_intf_full:    assume property (full_o  |-> !w_hsk || r_hsk);
```

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 proofs now? Two asserts failed



Proof Status: mixed Validity: up to date

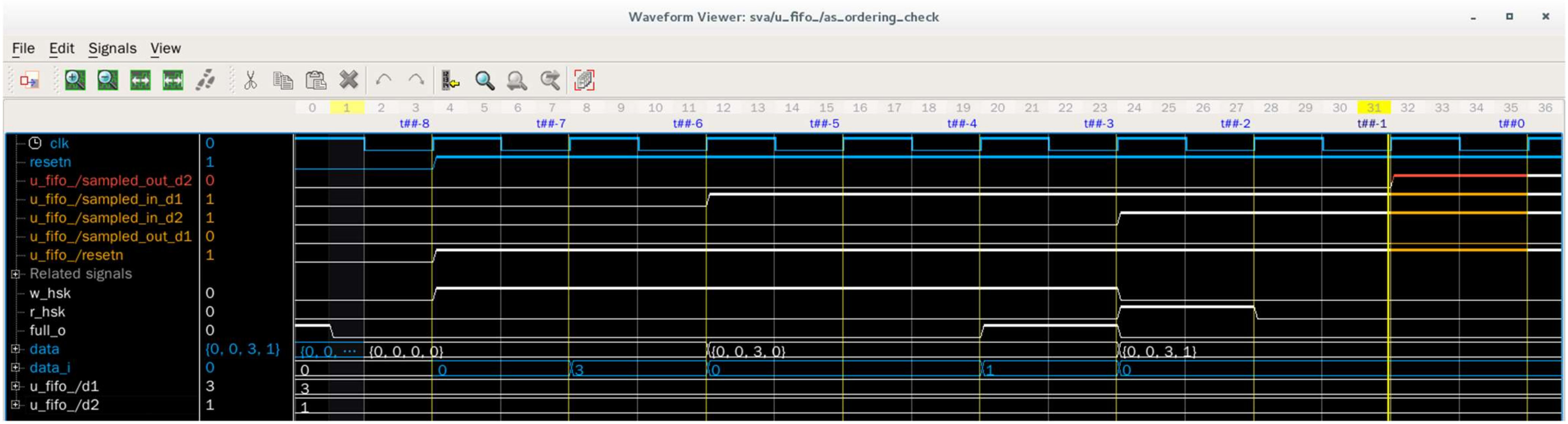
Name	Proof Status	Witness Status	Validity
	<any statu	<any st	<any validity>
Assertions			
sva/u_fifo_/as_empty_after_reset	hold	pass (1)	up_to_date
sva/u_fifo_/as_empty_to_full	fail (1)	pass (1)	up_to_date
sva/u_fifo_/as_full_to_empty	hold	pass (5)	up_to_date
sva/u_fifo_/as_ordering_check	fail (8)	pass (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

```
-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

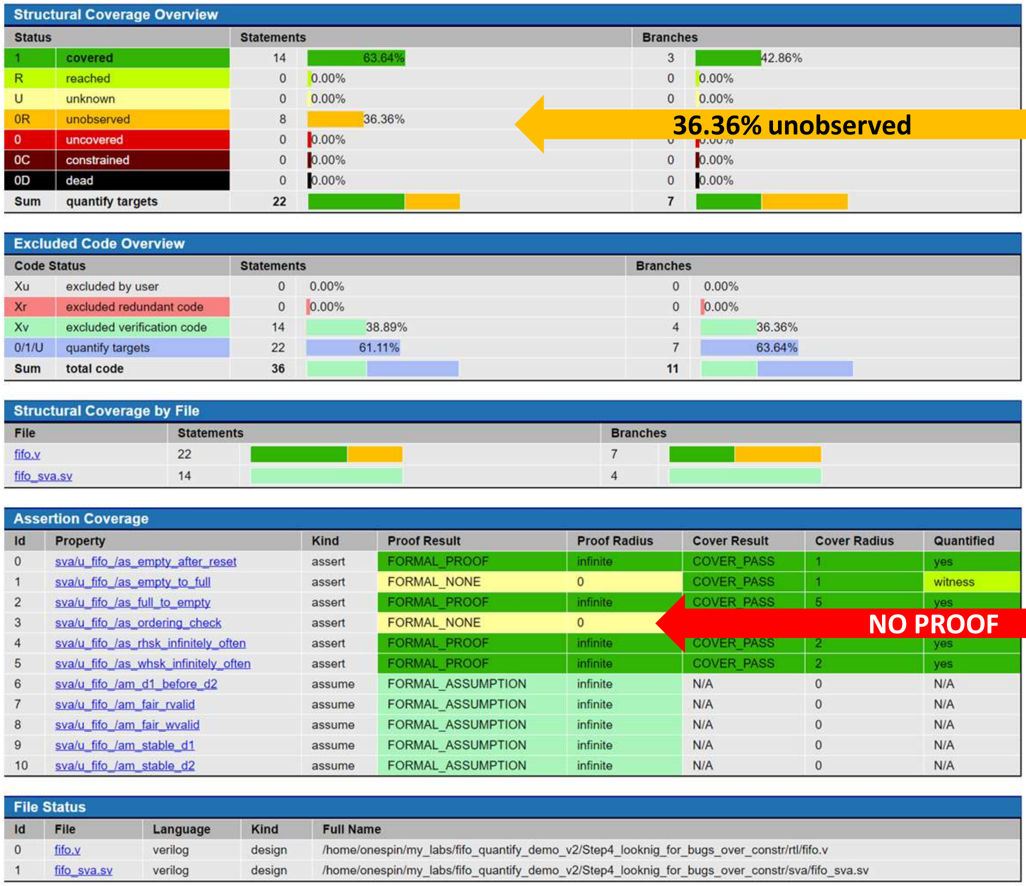


Waveform Viewer: sva/u_fifo_/as_ordering_check

Signal	Value
clk	0
resetn	1
u_fifo_/sampled_out_d2	0
u_fifo_/sampled_in_d1	1
u_fifo_/sampled_in_d2	1
u_fifo_/sampled_out_d1	0
u_fifo_/resetn	1
Related signals	
w_hsk	0
r_hsk	0
full_o	0
data	{0, 0, 3, 1}
data_i	0
u_fifo_/d1	3
u_fifo_/d2	1

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

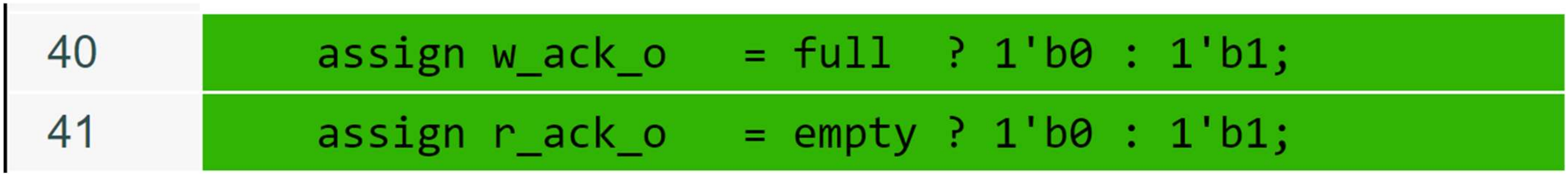
File Status

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%

Quantify on FIFO Example—XVIII

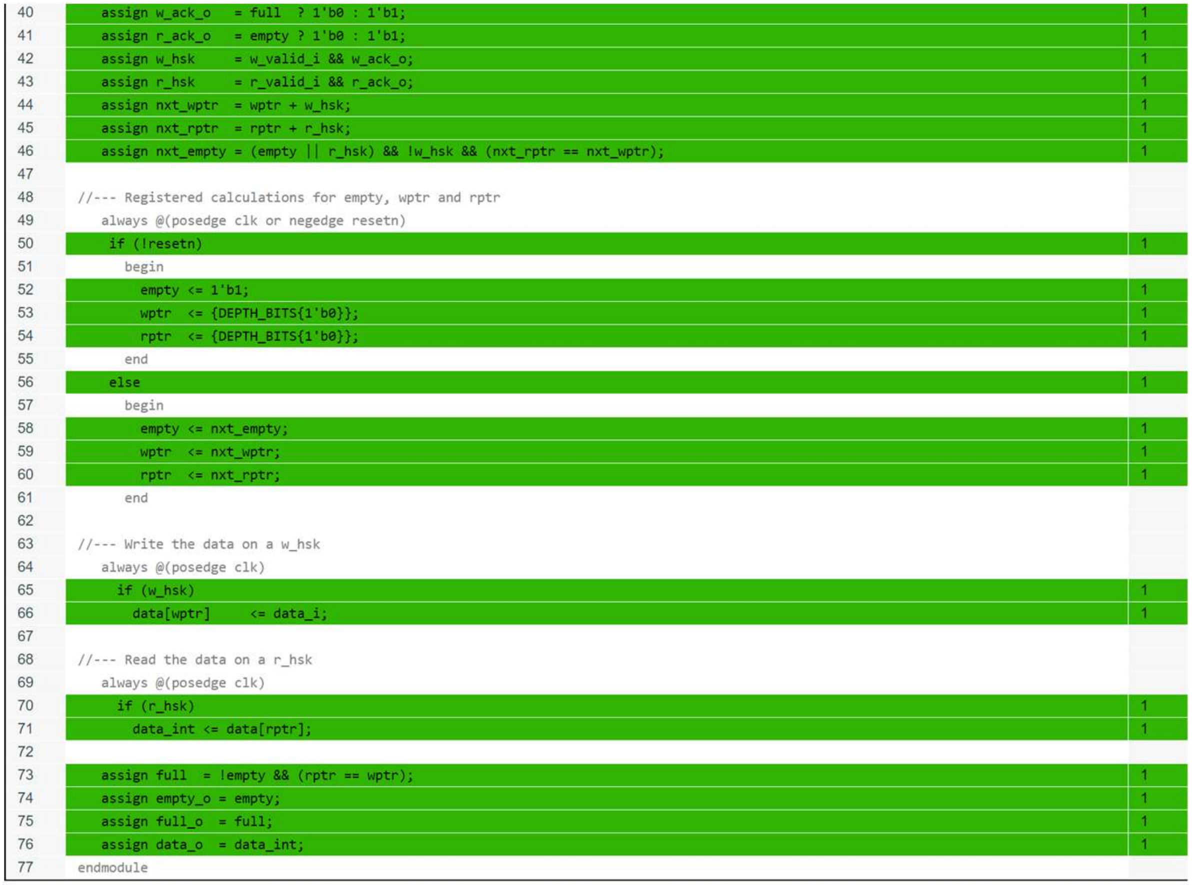
Fix the bug, prove, then Quantify



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

Quantify on FIFO Example—XVIII

Let’s fix the design and rerun proofs and Quantify

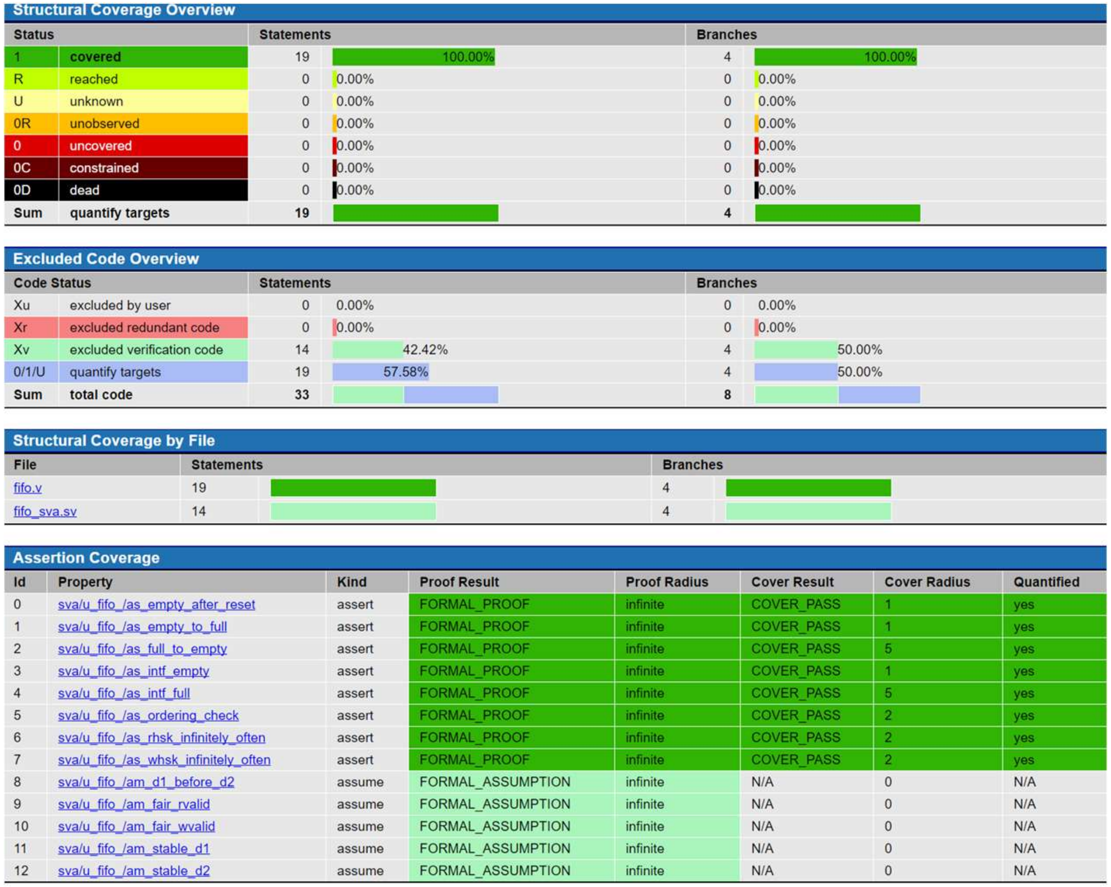


```
    assign w_ack_o   = full  ? 1'b0 : 1'b1;                                  1
    assign r_ack_o   = empty ? 1'b0 : 1'b1;                                  1
    assign w_hsk     = w_valid_i && w_ack_o;                                 1
    assign r_hsk     = r_valid_i && r_ack_o;                                 1
    assign nxt_wptr  = wptr + w_hsk;                                         1
    assign nxt_rptr  = rptr + r_hsk;                                         1
    assign nxt_empty = (empty || r_hsk) && !w_hsk && (nxt_rptr == nxt_wptr); 1

 //--- Registered calculations for empty, wptr and rptr
    always @(posedge clk or negedge resetn)
     if (!resetn)                                                            1
       begin
         empty <= 1'b1;                                                      1
         wptr  <= {DEPTH_BITS{1'b0}};                                        1
         rptr  <= {DEPTH_BITS{1'b0}};                                        1
       end
     else                                                                    1
       begin
         empty <= nxt_empty;                                                 1
         wptr  <= nxt_wptr;                                                  1
         rptr  <= nxt_rptr;                                                  1
       end

 //--- Write the data on a w_hsk
    always @(posedge clk)
      if (w_hsk)                                                             1
        data[wptr]     <= data_i;                                            1

 //--- Read the data on a r_hsk
    always @(posedge clk)
      if (r_hsk)                                                             1
        data_int <= data[rptr];                                              1

    assign full   = !empty && (rptr == wptr);                                1
    assign empty_o = empty;                                                  1
    assign full_o  = full;                                                   1
    assign data_o  = data_int;                                               1
 endmodule
```

100% covered!

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

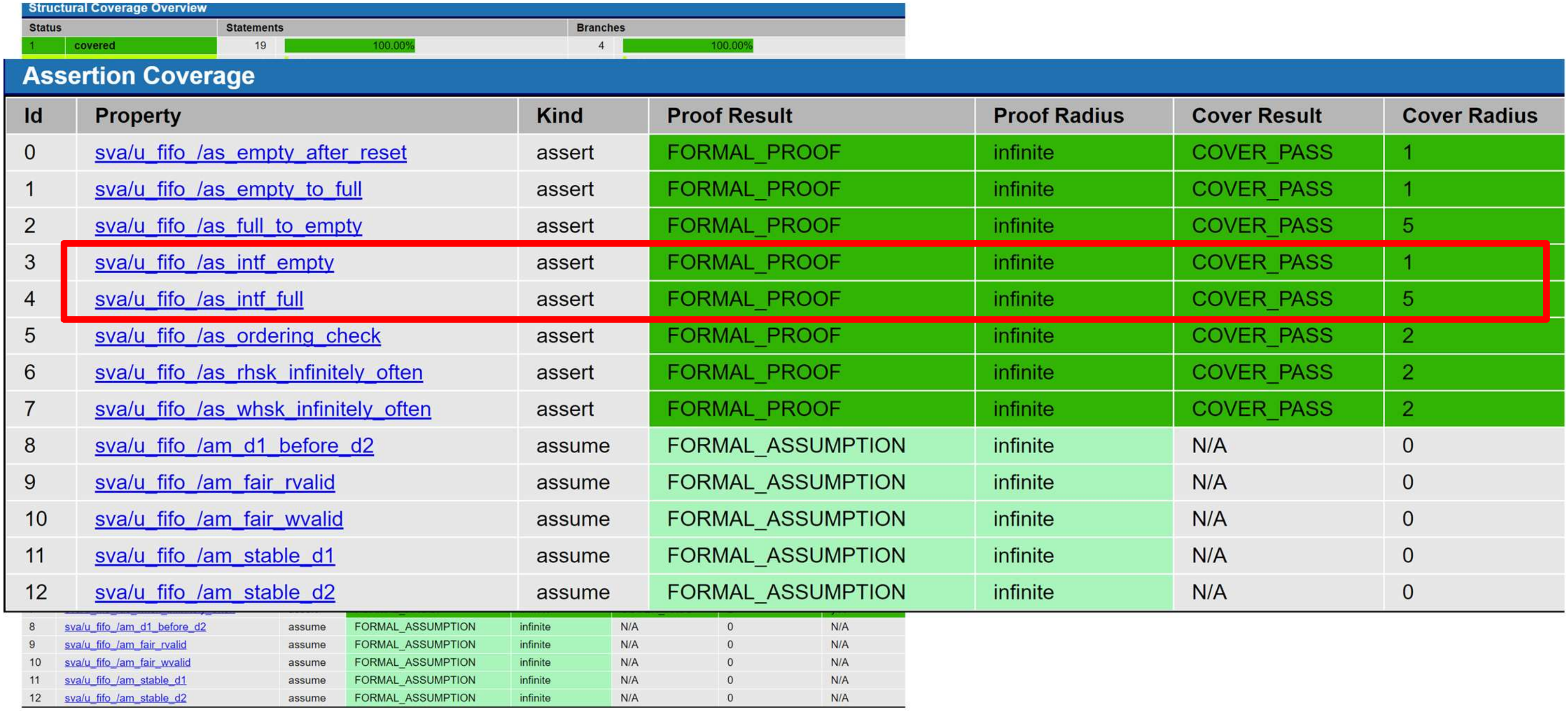
File	Statements		Branches	
fifo.v	19		4	
fifo_sva.sv	14		4	

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

Quantify on FIFO Example—XX

What happened to our constraint? It became a check!



Structural Coverage Overview

Status		Statements		Branches	
1	covered	19	100.00%	4	100.00%

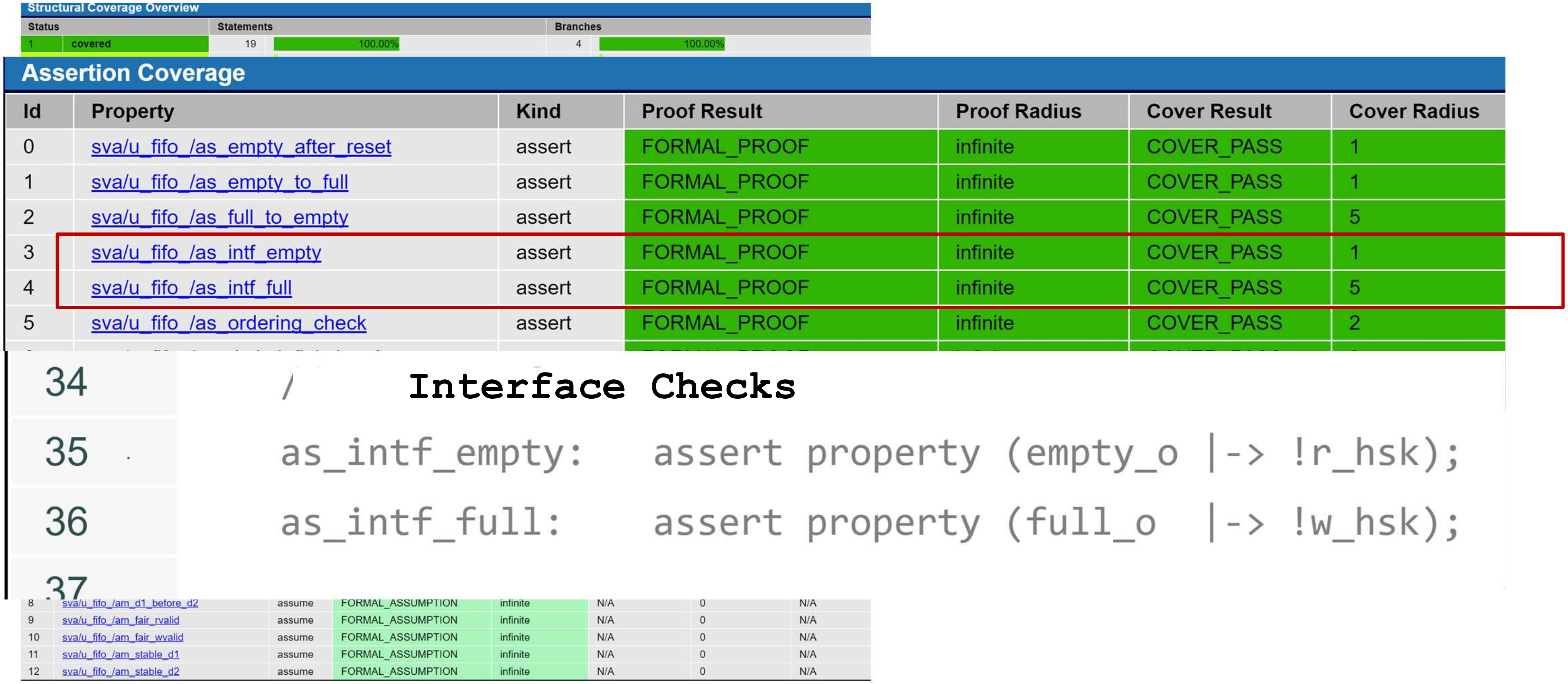
Assertion Coverage

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

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

Quantify on FIFO Example—XXI

We discover additional requirements on this design



Structural Coverage Overview

Status		Statements		Branches	
1	covered	19	100.00%	4	100.00%

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 Checks
as_intf_empty:   assert property (empty_o |-> !r_hsk);
as_intf_full:    assert property (full_o  |-> !w_hsk);
```

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

Recap of What We Showed—I

Using coverage for bug hunting

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

Recap of What We Showed—II

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 checks 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 checks

Verification of I²C Serial Protocol

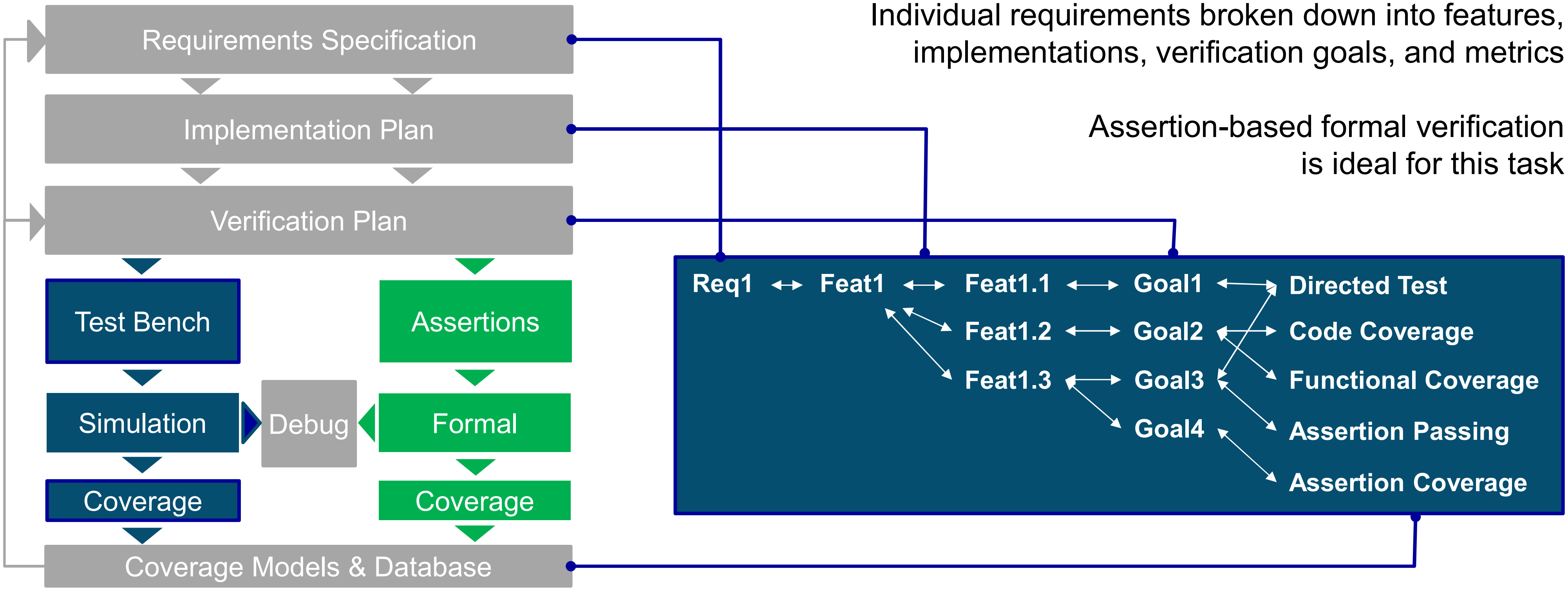
Case Study: Coverage’s Role in the Verification Process

Systematic Verification Flow

Requirement tracing and coverage are of paramount importance

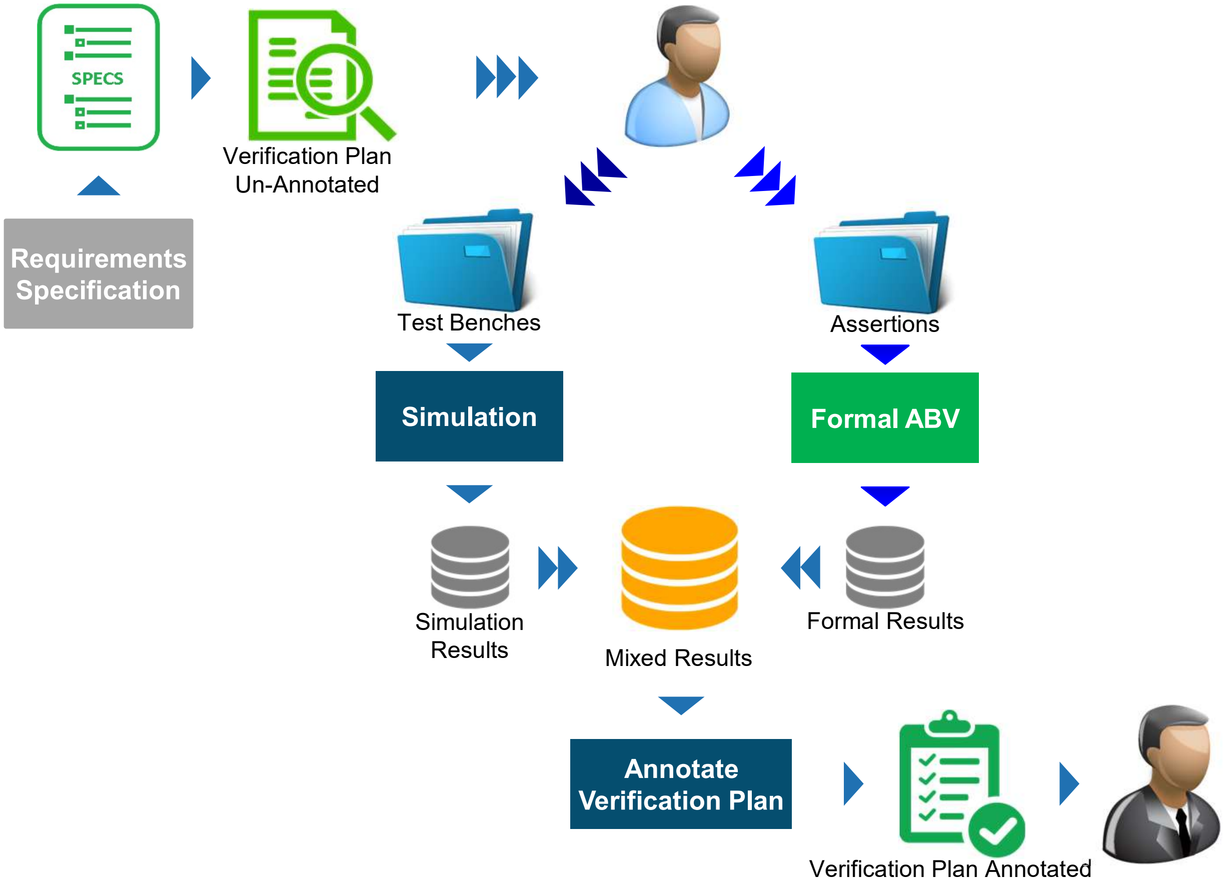
Individual requirements broken down into features, implementations, verification goals, and metrics

Assertion-based formal verification is ideal for this task



Getting the Big Picture of Verification

Integrated view of verification planning: formal and simulation


SPECS
Requirements Specification
Verification Plan Un-Annotated
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?

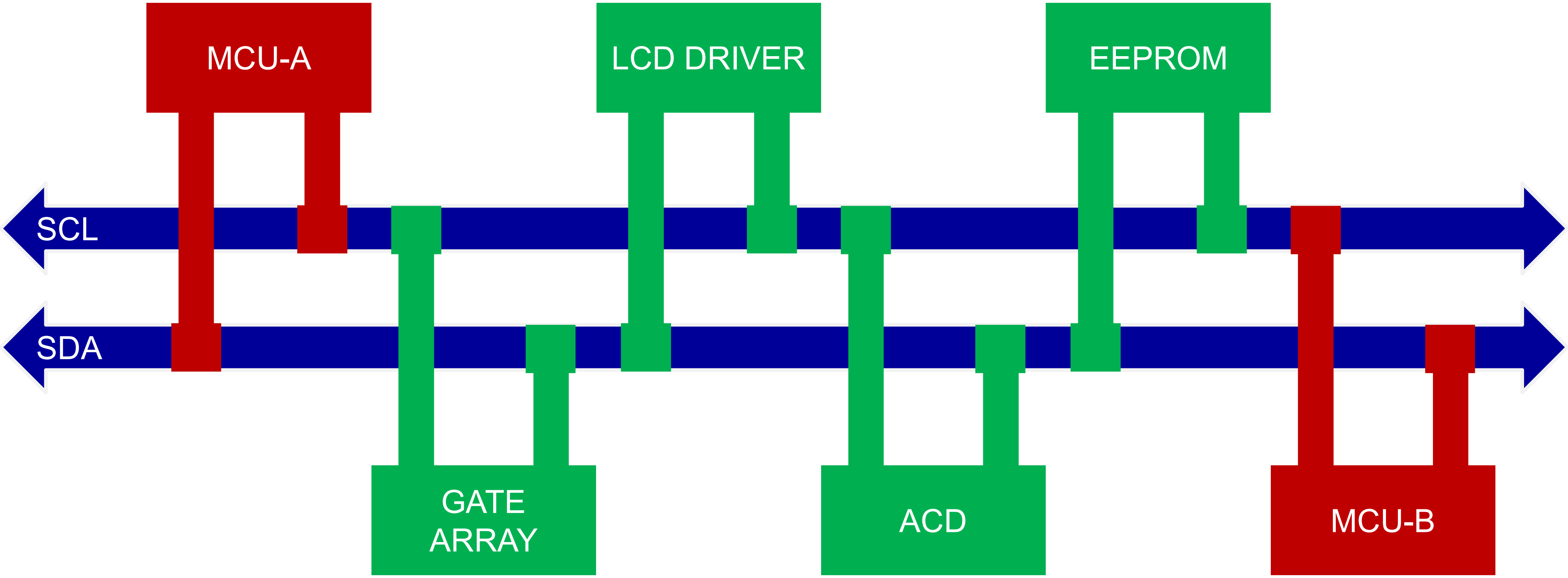
- Verify protocol compliance at the interfaces binding a VIP checker
- Make use of a scoreboard to check data integrity

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 check for many cycles in order to prove that a single byte is transferred correctly.*

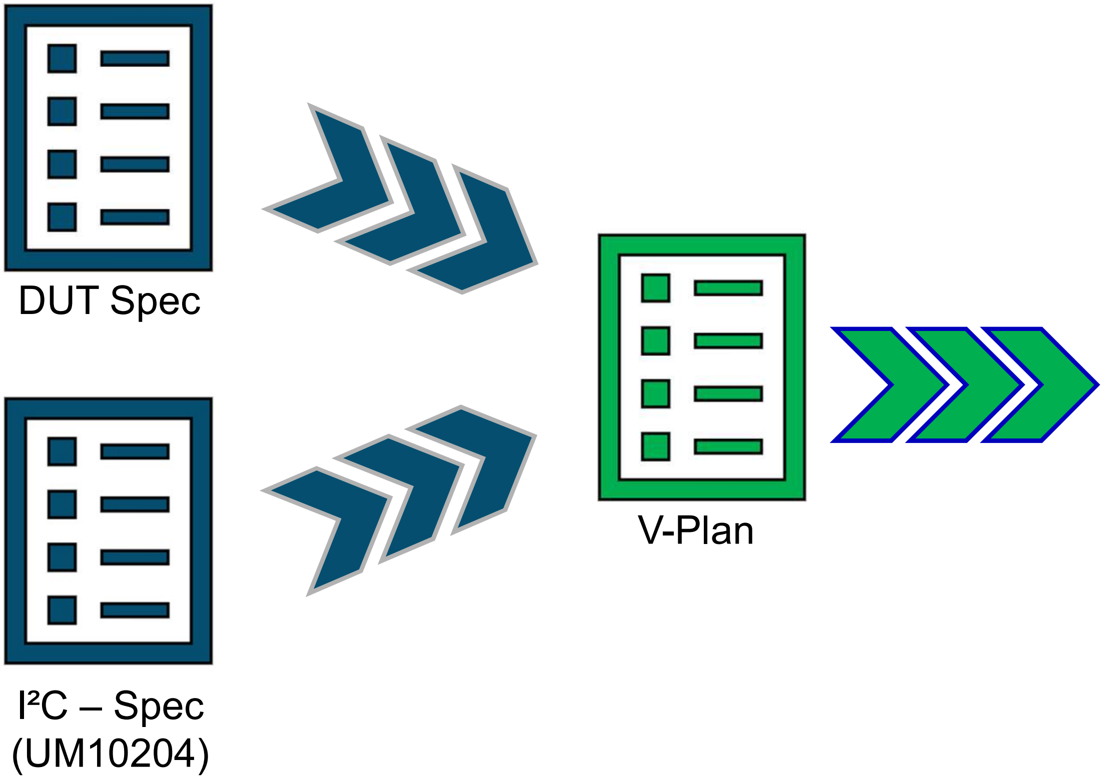
I²C Bus Protocol

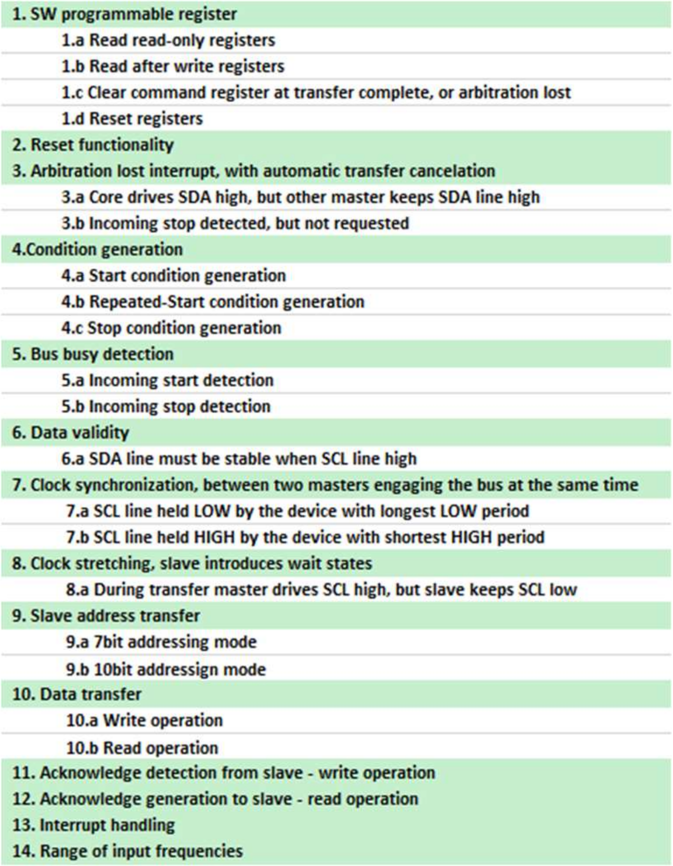
What is I²C about?



Coverage's Role in the Verification Process

Verification concerns: 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

Coverage's Role in the Verification Process

What is the very first step?

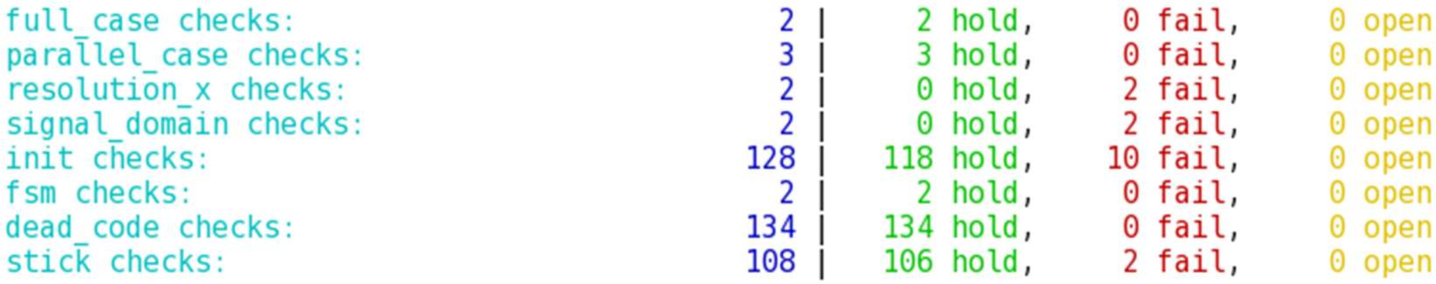
Let's analyze the design.

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*

Let's do an automatic inspection. Why?

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

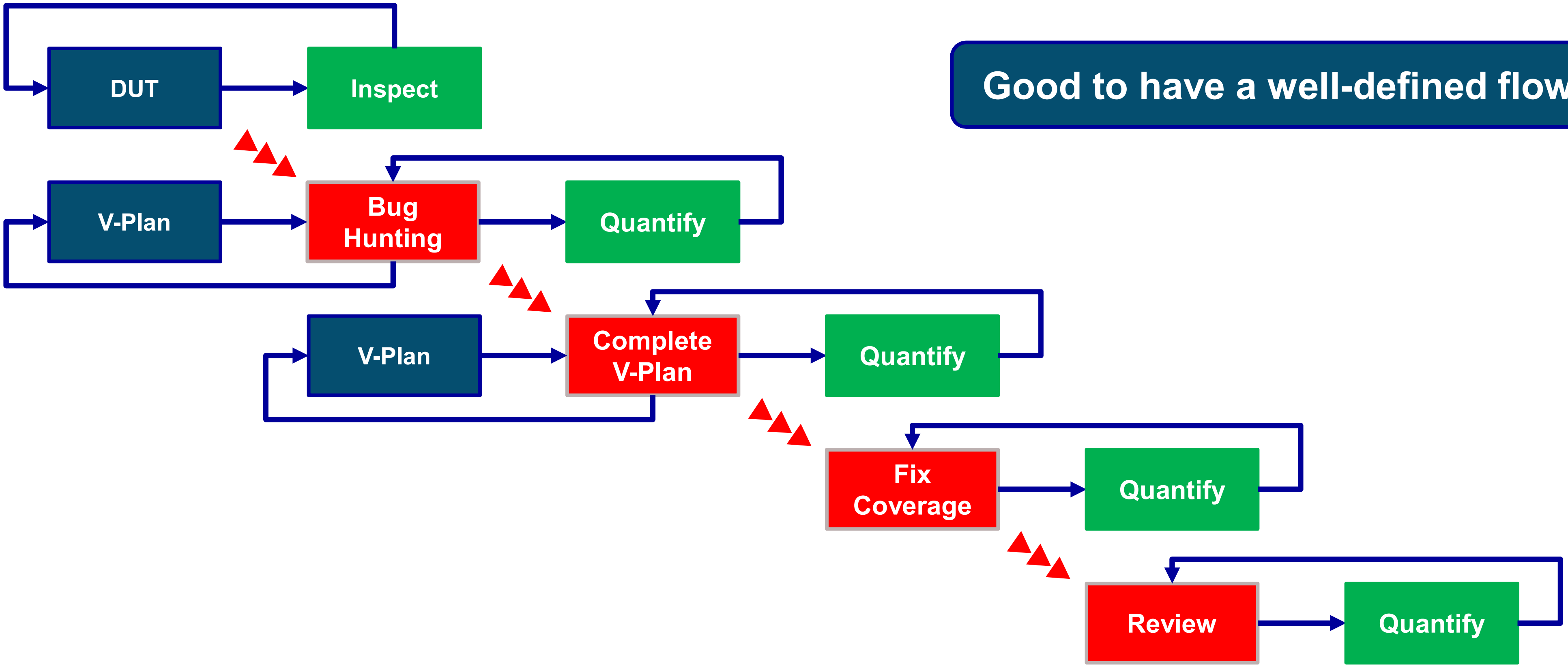
Validate results: Are failing checks expected?



```
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
```

Coverage's Role in the Verification Process

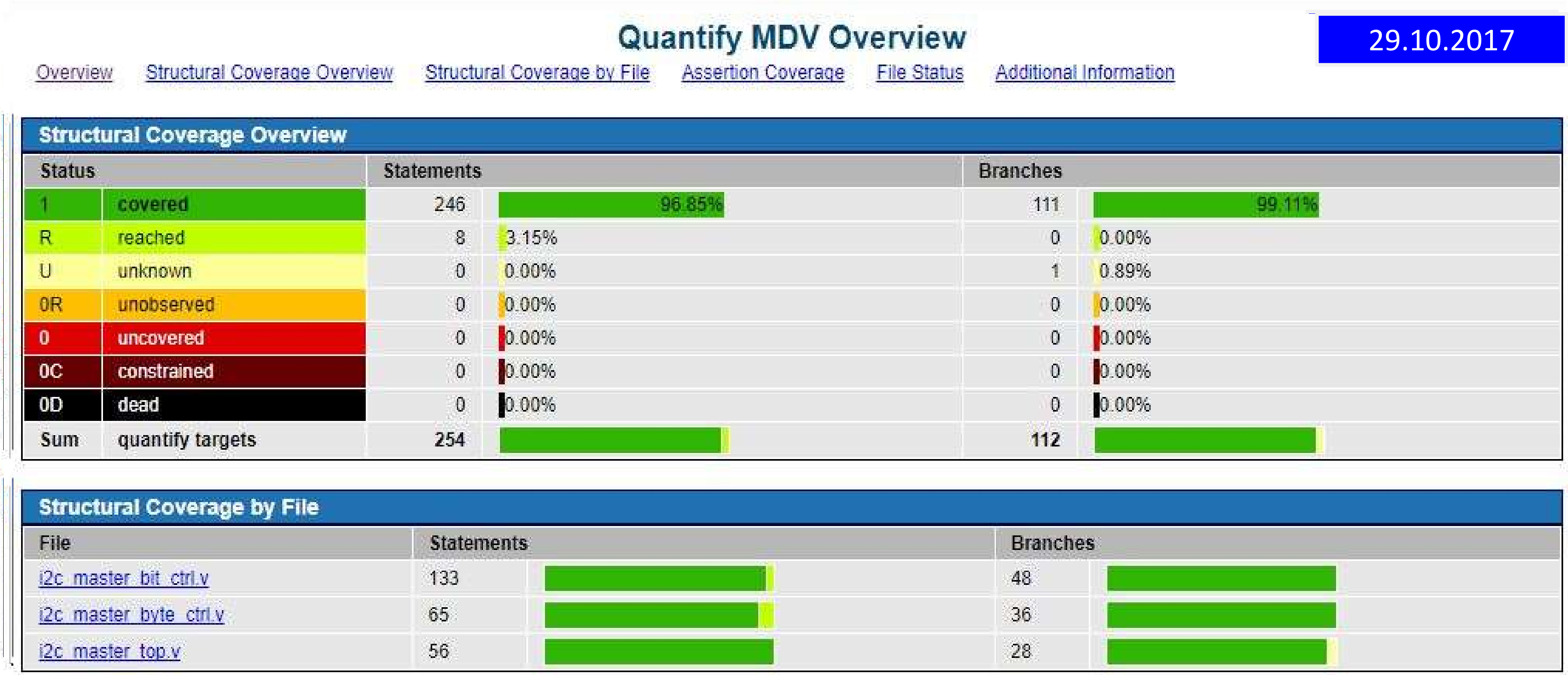
What is the approach?



Good to have a well-defined flow!

What Was Achieved?

Quantify MDV



29.10.2017

Quantify MDV Overview

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

Structural Coverage Overview

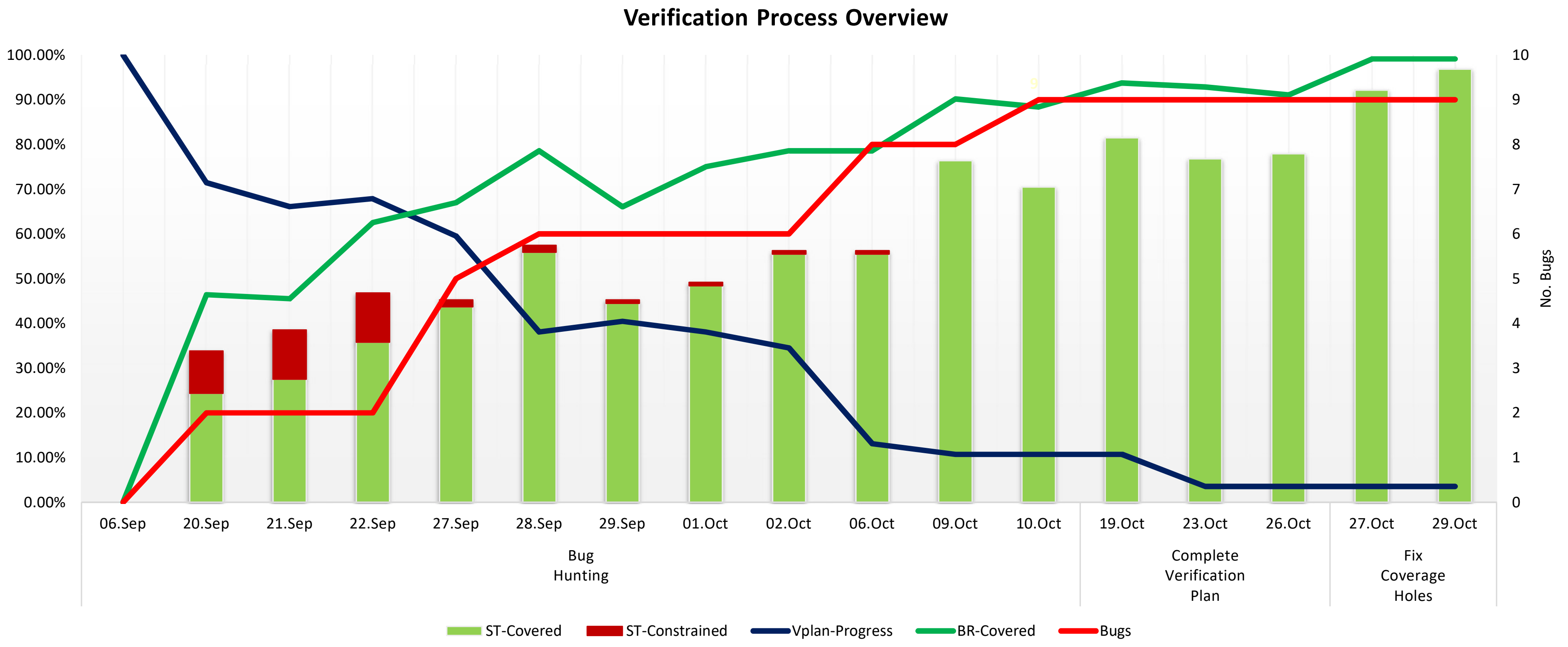
Status		Statements		Branches	
1	covered	246	96.85%	111	99.11%
R	reached	8	3.15%	0	0.00%
U	unknown	0	0.00%	1	0.89%
0R	unobserved	0	0.00%	0	0.00%
0	uncovered	0	0.00%	0	0.00%
0C	constrained	0	0.00%	0	0.00%
0D	dead	0	0.00%	0	0.00%
Sum	quantify targets	254		112	

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

Coverage's Role in the Verification Process

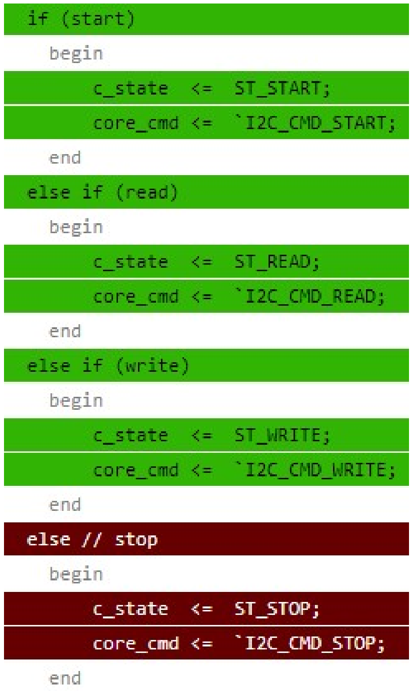
Process over time



Quantify in Action

Spotting over-constrained code—I

```
/****************************************************************/
/* 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
```

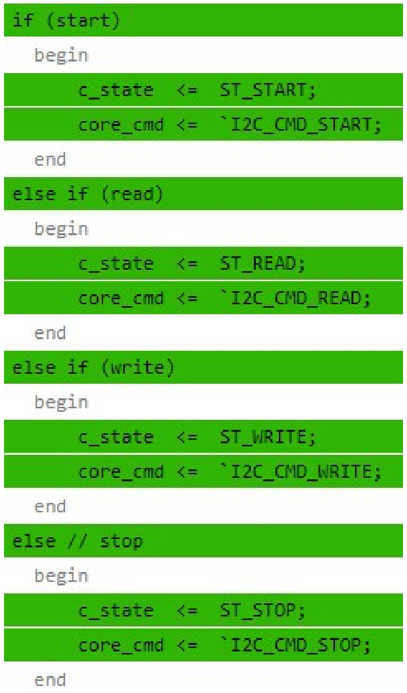
Quantify in Action

Spotting over-constrained code—II

```
/*****************************************************************/
/* 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]

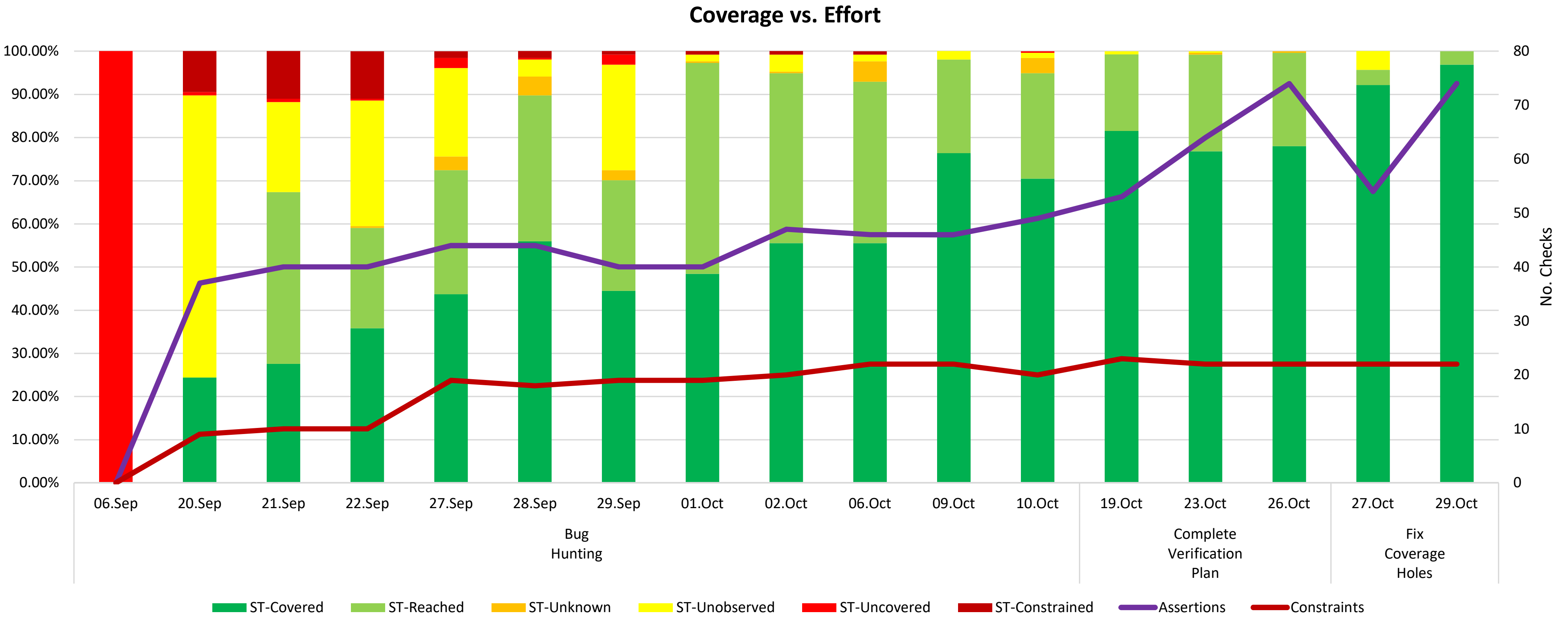
START immediately followed by a STOP is an illegal format



```
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
```

Coverage's Role in the Verification Process

Assertion effort vs. coverage



Summary

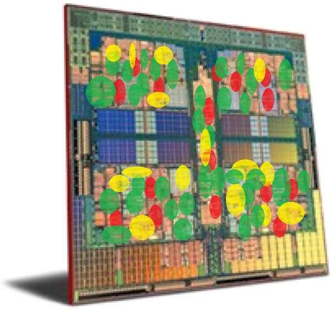
What is the motivation?

- 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: Scalable and Automated



- Push-button solution
- Unique patented technology
- Much more accurate than cone analysis
- Used by multiple customers on their most critical IP

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.

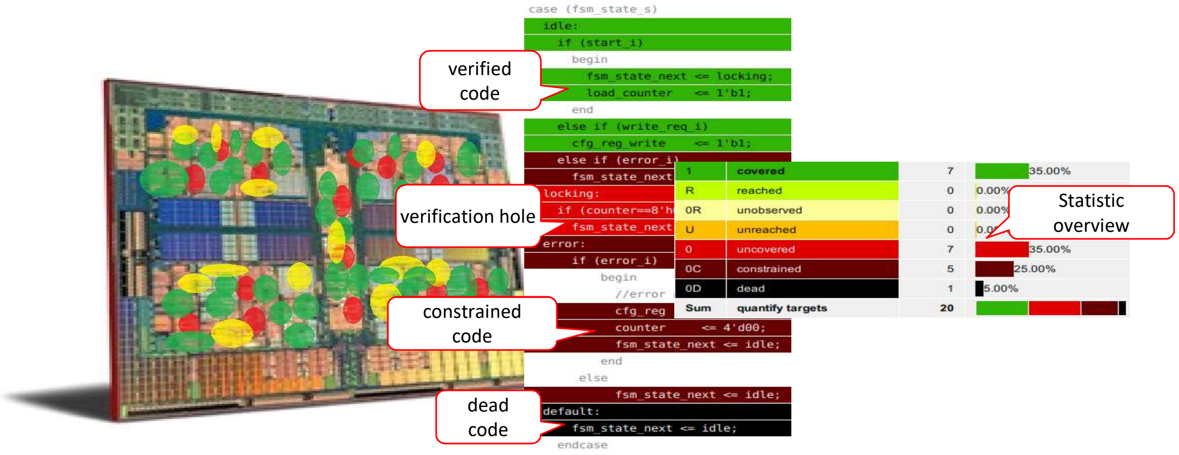
New assertions detected critical bugs.

Quantify now used to provide management metrics on all designs!

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

Quantification of Formal in ISO 26262

Coverage for safety-critical domains



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

1	covered	7	35.00%
R	reached	0	0.00%
0R	unobserved	0	0.00%
U	unreached	0	0.0
0	uncovered	7	35.00%
0C	constrained	5	25.00%
0D	dead	1	5.00%
Sum	quantify targets	20	

Problem

- ❖Quantitative assessment of formal verification environment needed
- ❖Example: Qualify verification environment for safety functions

Solution

- ❖Use observation coverage to identify coverage holes
- ❖Integrate coverage results with simulation coverage

Customer Case Study:

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

Quantify and Other Coverage Solutions

Why Quantify is a superior coverage tool

Cone of Influence

- Good to spot big gaps quickly
- Can get false optimism
- Can hide bugs in the design

Proof Core

- Result depends on selected proof engine
- More abstract engines produce pessimism
- Engine dependent results are confusing

Mutation

- Overall high run time—one fault at a time
- Some mutations can cause vacuity
- Intrusive—mutation applied on RTL
- Too many iterative compile and runs
- Covering all locations is expensive

Quantify

- Fast execution—multiple faults processed at once
- Not intrusive—alters the design model, *not* RTL
- Just-right level of abstraction
- Allows better observability
- Report is meaningful and linked to design browser

Summary

Continuous feedback for design and verification

- **Designer Bring Up**
  - What can you know about your design without any verification effort?
    - Reachability analysis—find design bugs as you bring up design
    - Redundant code—find wasted area in your design
- **Verification Quality and Metrics**
  - Metrics indicate gaps in verification and show you *where* these gaps are
  - Quantify tells you where checks are missing
  - More checks allow you to identify hidden bugs
  - Identify accidental over-constraints; focus on verification
  - Push-button, quick to run, easy to read, view linked to design browser
  - Single metric provides overall quality

Thank you!

Questions?