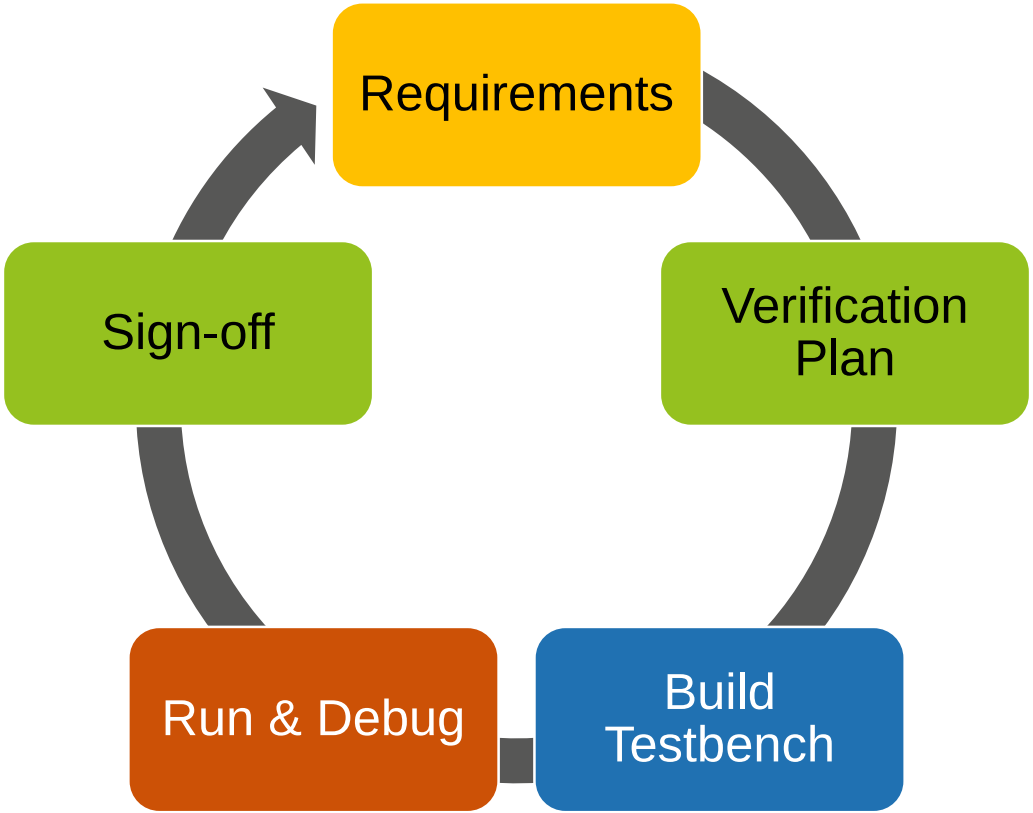
RedisCovering Coverage

Indeed the Grass is Greener on the Other Side

Ashish Darbari

Verification Futures 2017

The Verification Loop



 |

What's Different Now

A Verification Engineer's Perspective – Why Grass is Greener on the Other Side

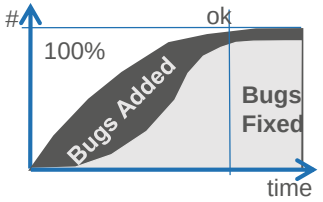
How did I sign-off my verification?

- Peer-reviews
- Structural coverage metrics
- Functional cover properties & Witness runs from assertions
- Over-constraint detection
- Hand crafted mutation
- Mutation based tools

What's changed now?

- Looked at OneSpin's coverage story and asked some difficult questions to the tool
  - Is the coverage solution answering the correct question?
  - Is it generating the correct metrics? – Quantitative View
  - How are metrics linked to the quality of verification? – Qualitative View
  - What is the cost of running coverage analysis versus returns?

Coverage Taxonomy



Progress Metric

Analytical — **Coverage**

Anecdotal — **Bug Rate**

Implementation artefacts

Structural

```
if (…)
  …
  if (…)
else …
```

Functional

Have I covered required scenarios?

Code — **DV-Inspect**

- Toggle
- Branch /Condition
- FSM
- Line / Statement / Block

Formal Assertion — **Quantify**

Complete — **Gap Free Analysis**

Formal Assertion — **Quantify**

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 has been checked by my checkers?
- Are all specified functions **implemented**?
- Are all specified functions **verified**?

Quantify

Gap Free

Quantify

Quantify MDV Overview

Multi-dimensional view – Quantity & 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

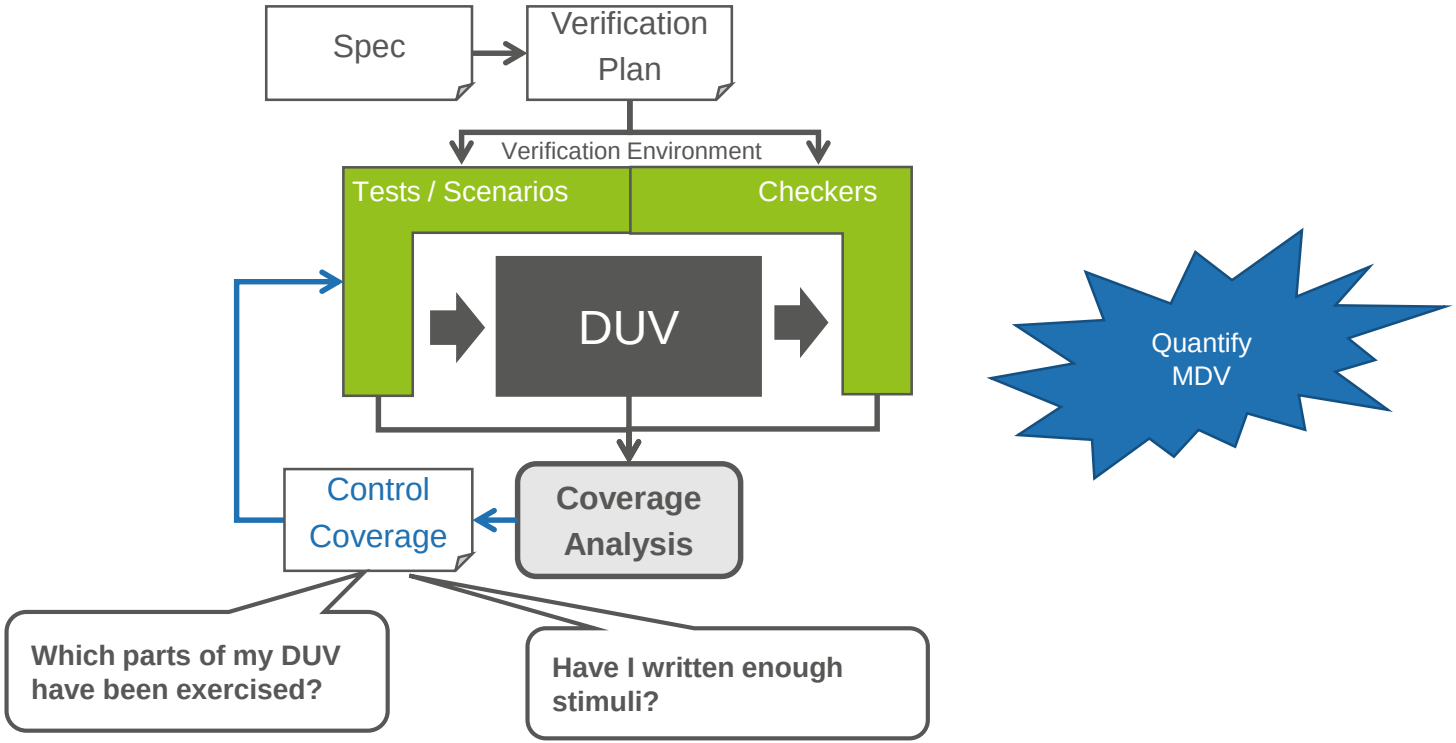
Structural Coverage (Quantitative)

- Control Coverage
  - Have all design signals been reached?
- Observation Coverage
  - Have all design signals been observed?

Functional Coverage (Qualitative)

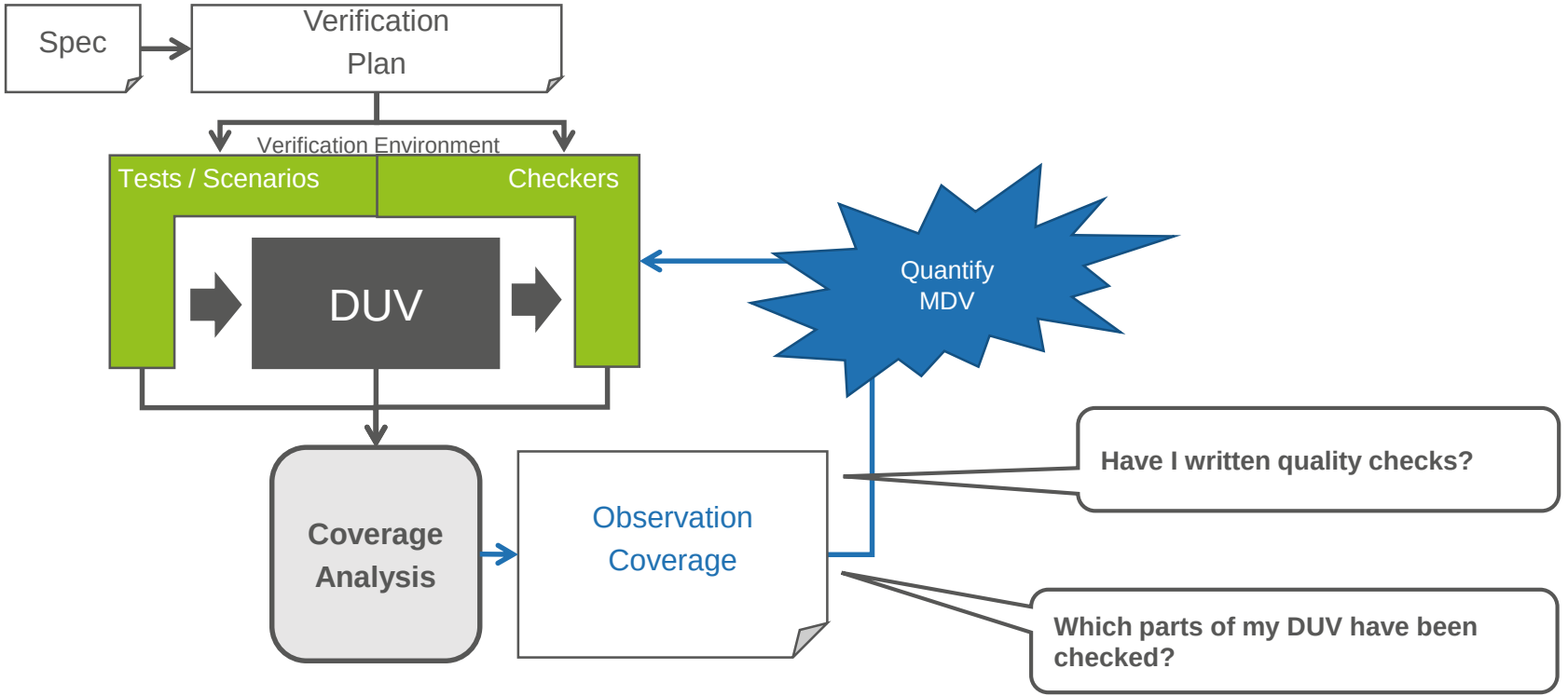
- Assertion Coverage – Provides qualitative assessment on functional checks

Control Coverage Flow



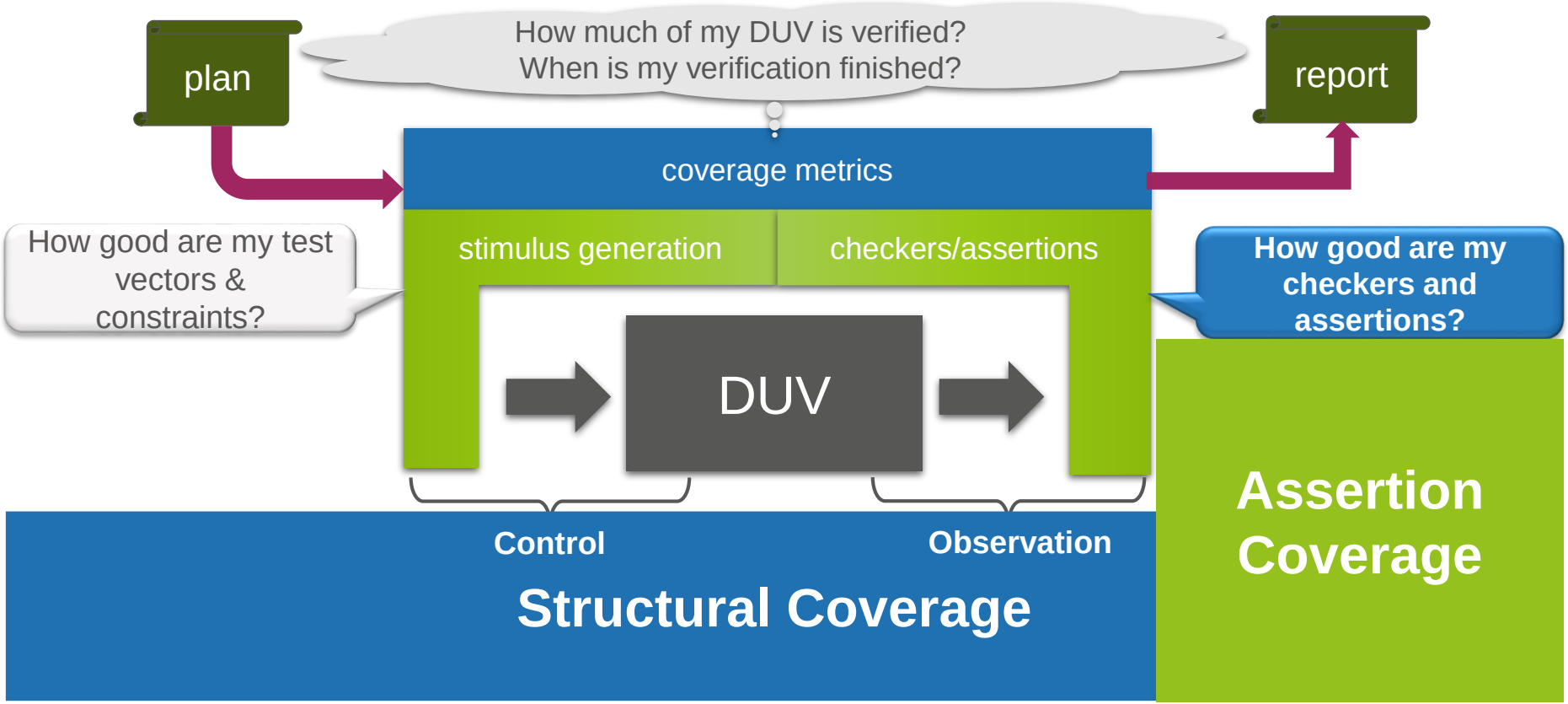
 |

Observation Coverage Flow



 |

Components of Quantify



Quantify Report
Control Coverage

	Result	Meaning
Controllability	Dead	Uncontrollable independent of constraints
	Reached	Controllable and reachable
	Constrained	Uncontrollable 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.

Quantify Report
Observability Coverage

	Result	Meaning
Observability	Uncovered	Not observed
	Covered	Observed by some assertion

Important to assess the “quality” of checking.

Have we observed all the design signals?

Have we got good quality assertions?

Quantify Report
Finding Redundant Code

	Result	Meaning
Exclusion	Redundant	No contribution to IO or assertions
	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 specify additional code to be excluded from quantification

Quantify Report – Structural View

	Result	Meaning
Controllability	Reached	Reachable
	Constrained	Unreachable and Unobservable due to constraints
	Dead	Unreachable and Unobservable
Observability	Uncovered	Not Reached and Not Observed
	Covered	Observed and Reached
Exclusion	Redundant	No contribution to IO or assertions
	Verification	Only used for verification
	Excluded	Excluded by user

Verification Hole

Two Key Points to Remember

- **Unreachable implies Unobservable**
- **Observable implies Reachable**

Quantify Dashboard View

onespin

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**	

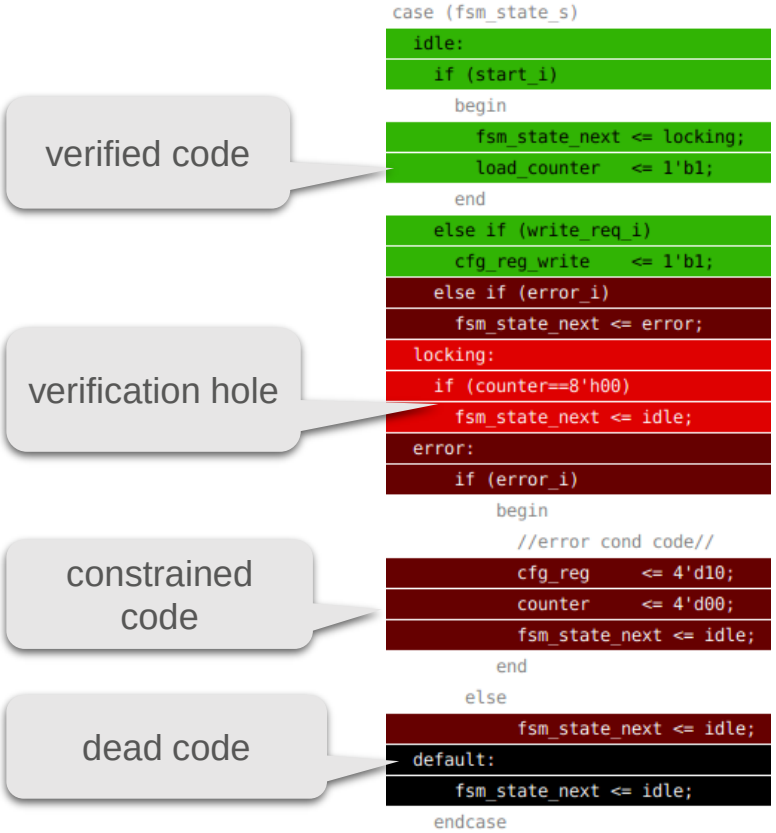
Excluded Code Overview

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

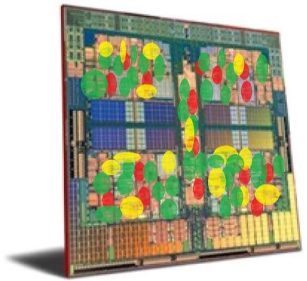
Id	Property	Kind	Proof Result	Proof Radius	Cover Result	Cover Radius	Quantified
0	sva/as_empty_from_full	assert	FORMAL_PROOF	infinite	COVER_PASS	9	yes
1	sva/as_full_from_empty	assert	FORMAL_PROOF	infinite	COVER_PASS	1	yes
2	sva/u_fifo_/as_ordering_check	assert	FORMAL_PROOF	infinite	COVER_PASS	2	yes

Quantify Dashboard View : Design Annotation



- Quantify MDV formally analyzes controllability and observability
- Statements and branches annotated in HTML report
- Verification holes concretely indicate missing assertions
- Constrained code quickly indicates over-constraining

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

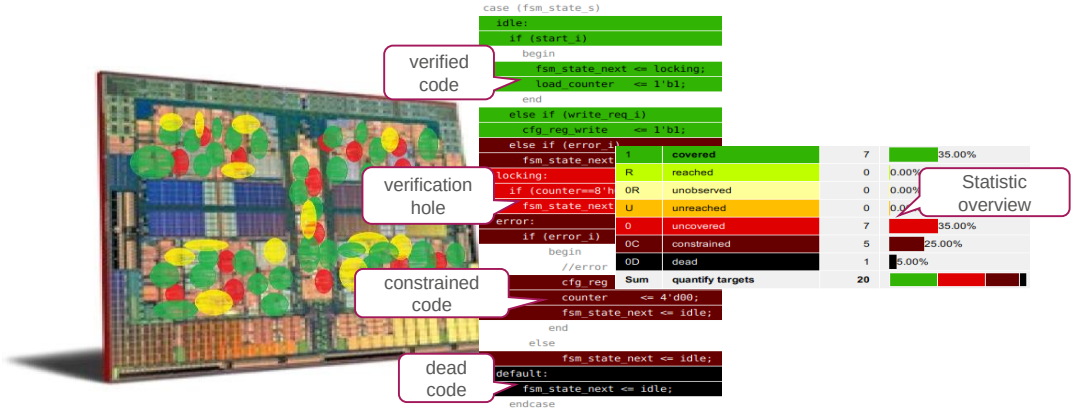
Design	#Code Lines	#Assertions
IFX-Aurix-1	25563	85
IFX-Aurix-2	27374	157
IFX-Aurix-3	57253	253

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



Customer Case:

“Formal Safety Verification With Qualified Property Sets” Holger Busch at DAC’14 in Accelerating Productivity Through Formal and Static Methods, Session 38.3

Problem

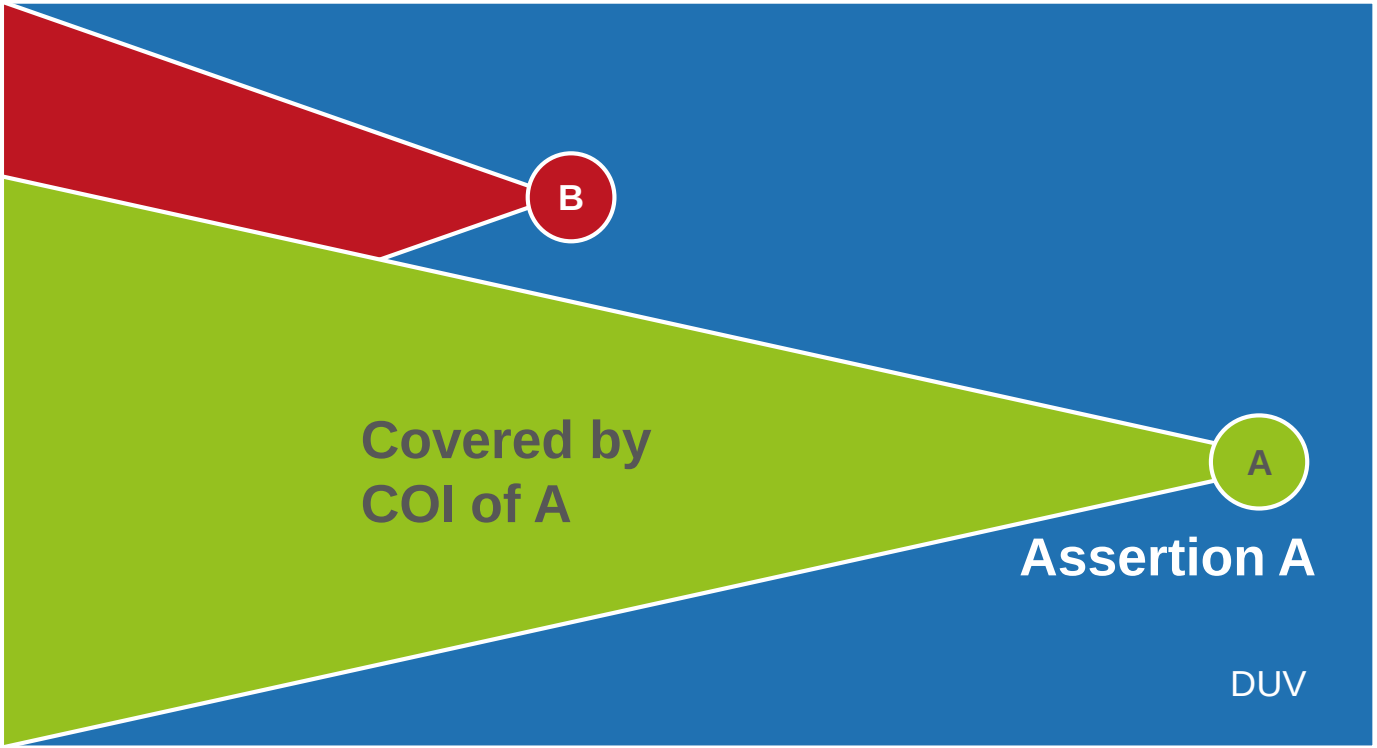
- Quantitative assessment of formal verification environment needed
- Example: Qualify verification environment for safety functions

Solution

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

Challenges with Other Solutions for Coverage

Cone-of-Influence Coverage



Why COI Coverage is Misleading

Example of a FIFO Check

```
property data_not_corrupted_p;

reg[WIDTH-1:0] dat;
       (empty & wr_en, dat=wr_data[WIDTH-1:0])
         ##1 !rd_en[*0:$]
         ##1 rd_en
        |=>
          (rd_data[WIDTH-1:0]==dat)
BAD  || (!full | empty);

endproperty;
```

COI Coverage: 100%

Quantify Coverage: 20%

Stronger Assertion Finds the Bug

```
sva/check/first_data_not_corrupted_a: assert property
(@(posedge clk)
disable iff ((! reset_n))
 [-]data_not_corrupted_p);
[x]
property data_not_corrupted_p;
  [-](((empty & wr_en), (dat
  = wr_data[WIDTH - 1 : 0]))
  ##1 (! rd_en)[*0:$] ##1
  rd_en |=> (rd_data[WIDTH - 1
  : 0] == dat));
[x]
(
  [+](empty & wr_en) ##0
  (1'b1, [+](dat =
  (wr_data[WIDTH - 1 : 0])))
  ##1 1[*0] ##1 [+](rd_en) |
  => [-]((rd_data[WIDTH - 1 :
  0]) == dat))
[x]
// signal values
    t ##2 check/rd_data == c
    dat == b
```

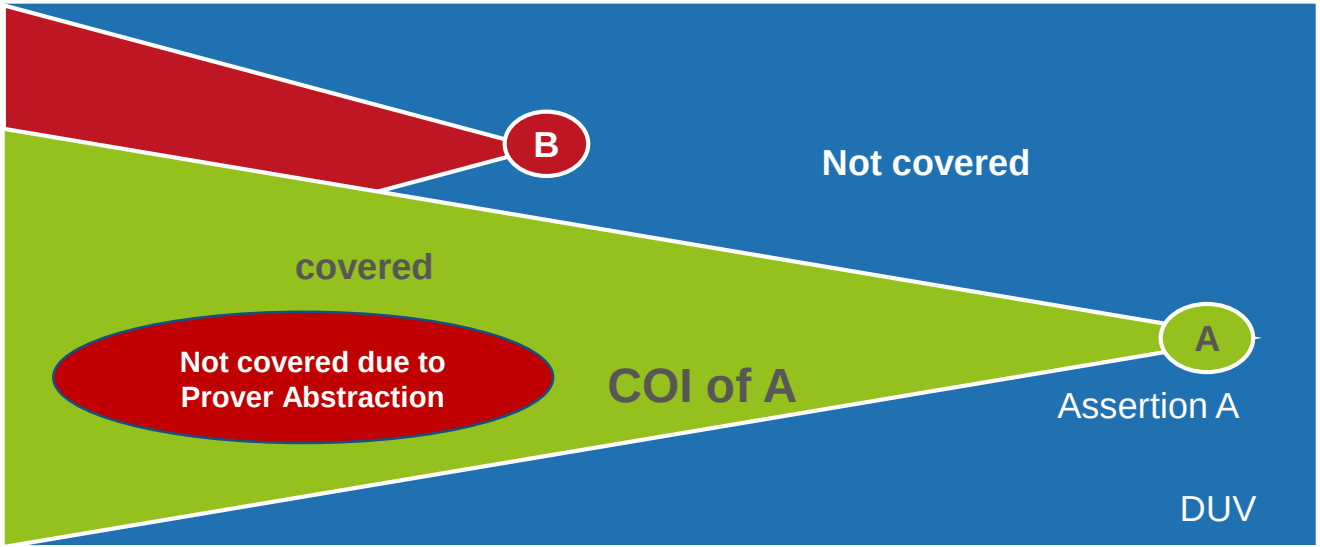
Path: [top] TimePoint: 7

```
118      begin
119         mem [ wr_pointer ] <= wr_data_enc + 1; // another bug
            55... 0->1              0
120      end
121  end
```

fifo.sv (read-only view) line 119, column 19

Don‘t trust the COIs!

Proof Core Coverage Not Correct



Fishy!

- Not covered what the prove engine did not need
- Corresponds to abstractions inside prove engines
- Each prove engine uses different abstractions
- **Better prove engines give lower coverage!**

Coverage for Safety Compliance

Cone-of-Influence (COI) Coverage

- Good to spot big gaps quickly
- Too coarse for sign-off

Proof Core Coverage

- Result depends on selected prove engine
- Not objective

Mutation Coverage

- Overall high run time – one fault at a time
- Some mutations can cause vacuity e.g., “a<=b” replaced by “a <= 1’b1”
- Intrusive – mutation applied on RTL, too many iterative compile & runs
- Covering all locations is expensive

Quantify Observation Coverage

- Fast execution as multiple fault models are processed at once
- Not intrusive – alters the “design model” not RTL code!
- Just “right level of abstraction” – allows better observability

Disqualified for Safety!

Qualified

Best for Safety

Summary

Showed you a coverage solution targeting ABV based formal verification

If you don’t use formal for verification then perhaps you should !!

If you use formal and don’t use OneSpin then you must !!

Only OneSpin provides a complete end-to-end coverage solution that provides

- Structural coverage through Quantify
- Functional Coverage through Quantify and GapFree Analysis
- Coverage for automotive and functional safety

Next Steps

- Integration of formal assertion coverage in the user verification plan
- Providing cross-vendor integration of assertion coverage
- Integration of formal assertion coverage with simulation coverage