DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2016

Oski TECHNOLOGY

# The Process and Proof for Formal Sign-Off – A Live Case Study

Ipshita Tripathi, Ankit Saxena, Anant Verma, Prashant Aggarwal

Oski Technology, Inc.

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

# Agenda

- Introduction – Formal Sign-off
- A Live Case Study – “Break the Testbench” challenge at DAC 2015

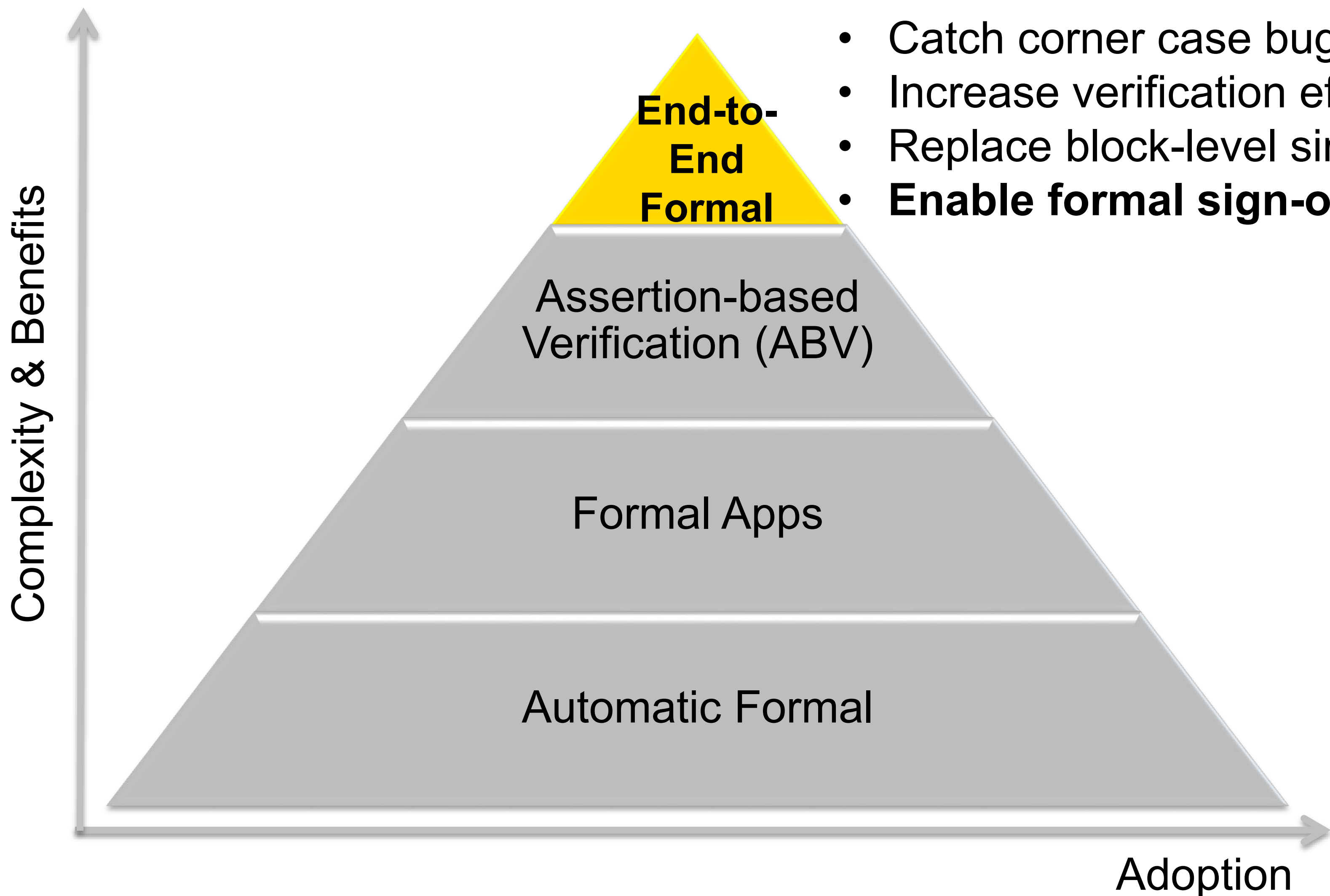
 Prashant Aggarwal, Oski Technology, Inc. All rights reserved. 

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2016

# Introduction – Formal Sign-off


Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

# End-to-End Formal Enables Formal Sign-off

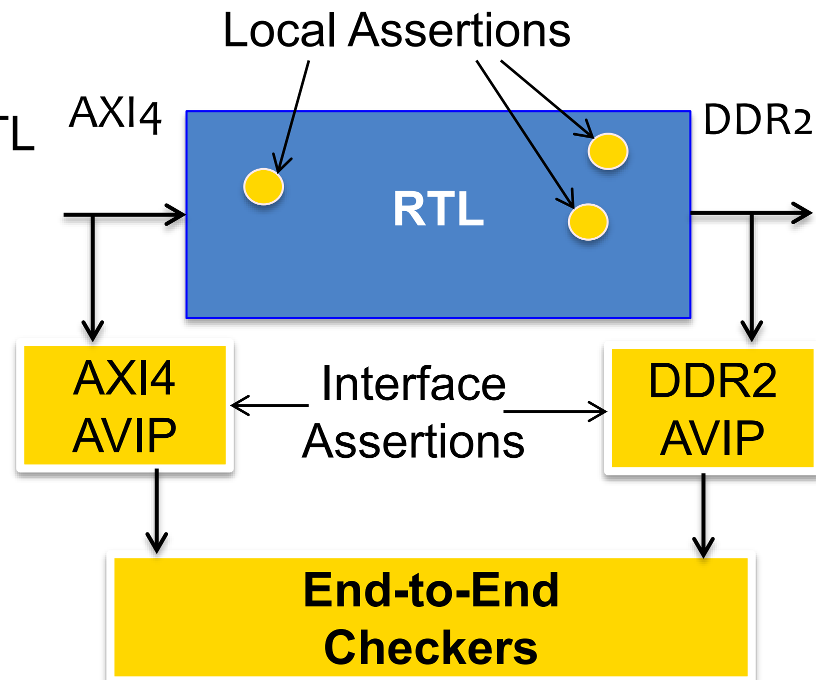



- Catch corner case bugs early
- Increase verification efficiency
- Replace block-level simulation
- **Enable formal sign-off**


Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

# What is End-to-End Formal?

- Local Assertions: Easier to verify
  - Internal RTL assertions, embedded in RTL
- Interface Assertions: Harder to verify
  - Relate to inputs/outputs
  - E.g. ACE, AXI4, OCP, DDR2, …
- End-to-End Checkers: Hardest to verify
  - Model end-to-end functionality
  - Often require Abstraction Models to manage complexity
  - **Can replace simulation**





Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

# Designs Best Suited for End-to-End Formal

**"Control", "Data Transport" designer size blocks:**

- Arbiters of many kinds
- Interrupt controller
- Power management unit
- Credit manager block
- Tag generator
- Scheduler
- Bus bridge
- Memory controller
- DMA controller
- Host bus interface
- Standard interface (PCIe, USB)
- Clock disable unit

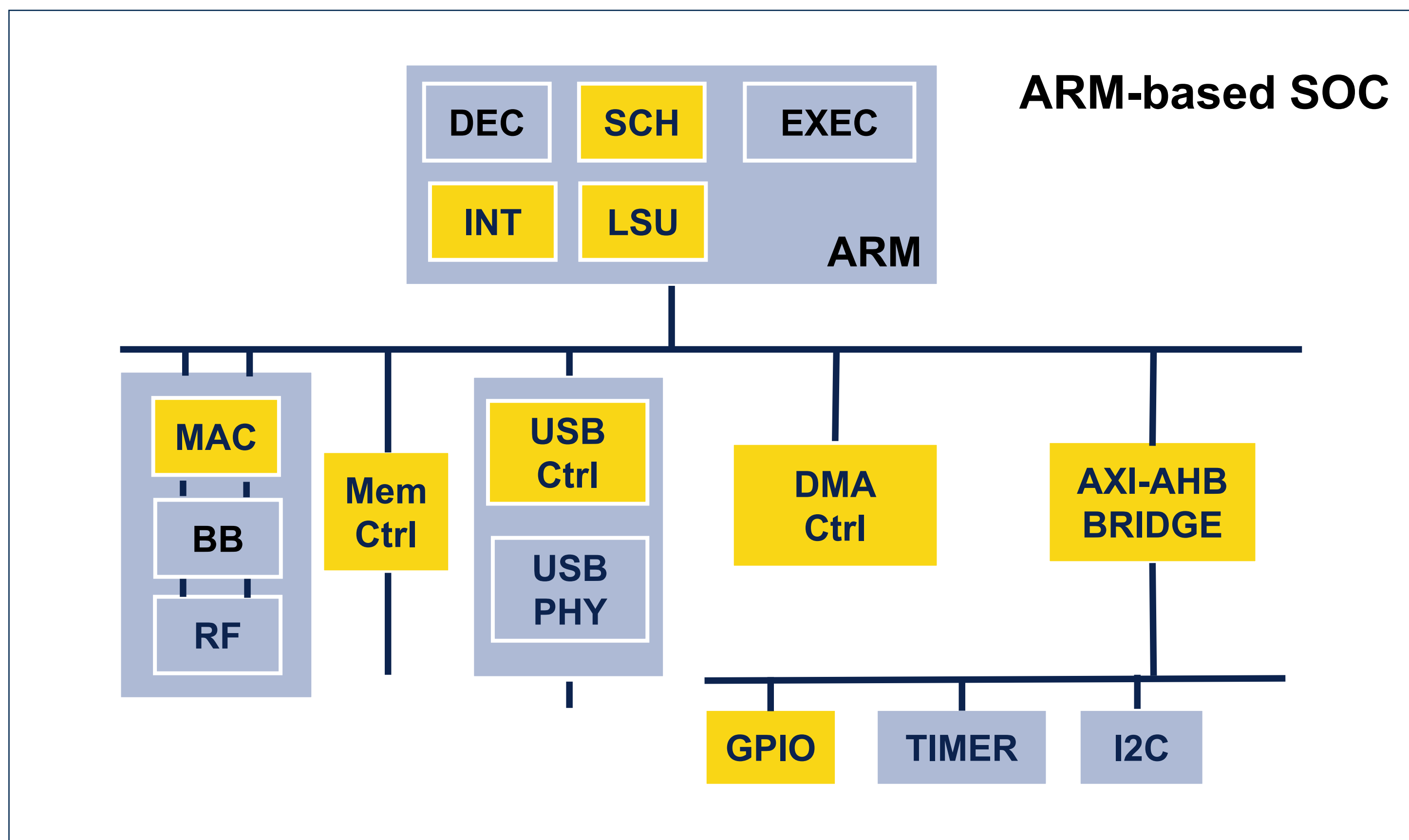
 Prashant Aggarwal, Oski Technology, Inc. All rights reserved. 

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2016

# Many SoC Blocks Can Be Verified with End-to-End Formal

End-to-End Formal

Simulation

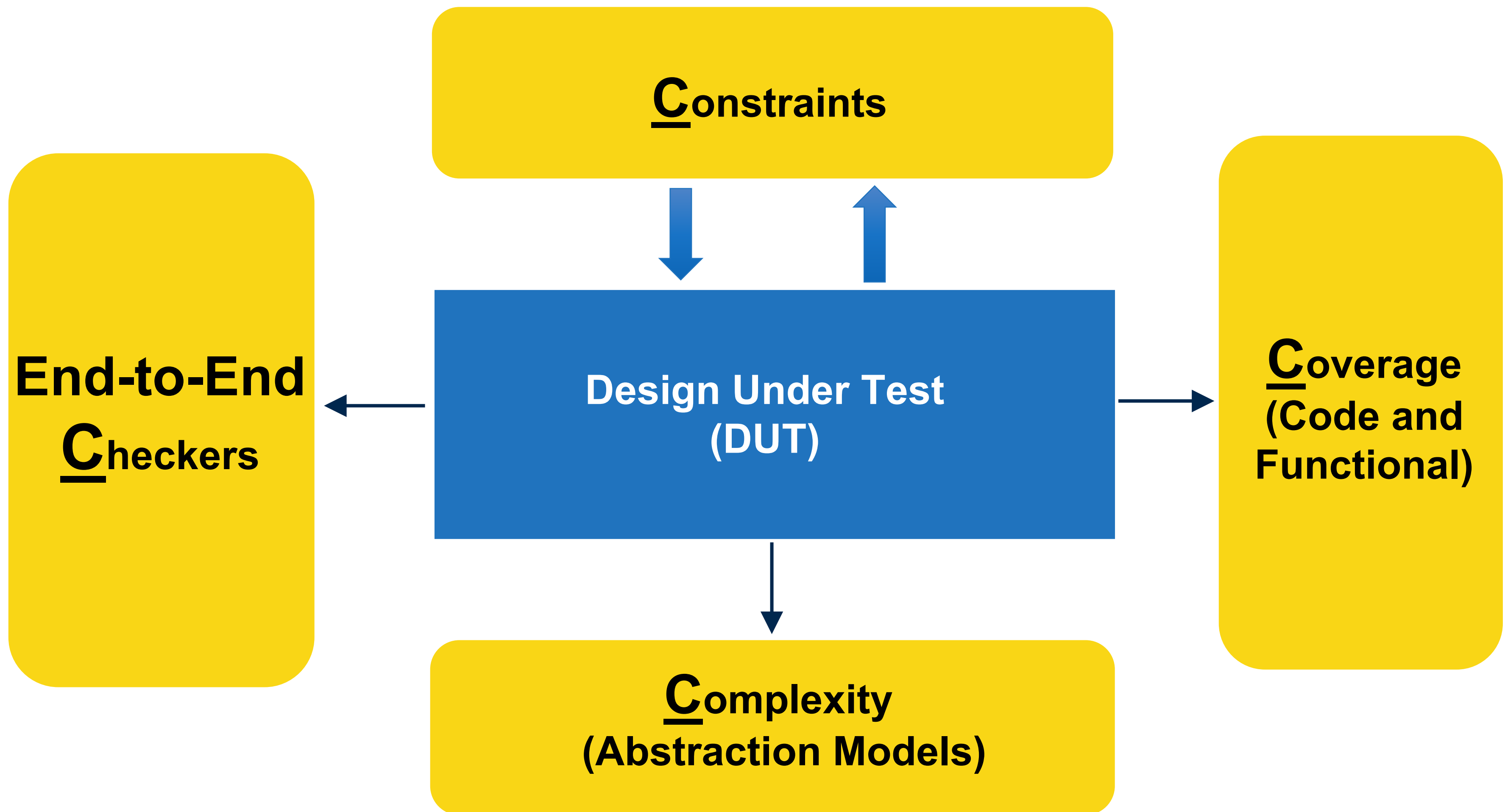



- Planning at the micro-architectural design stage is critical
- **End-to-End formal can fully replace simulation for many blocks**


Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

# Oski Formal Sign-off Methodology

## Quality of Formal Depends on all 4 Cs!





Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

DESIGN AND VERIFICATION™ DVCON 2016 CONFERENCE AND EXHIBITION UNITED STATES

# End-to-End Formal is Complete

For End-to-End formal to be complete, ideal metrics answer:

- Constraints: Have I unintentionally over-constrained any inputs?
- Complexity: Have all my checkers reached the Required Proof Depth?
- Checkers: Does my list of checkers fully embody the specified behaviour of the design?

**We will use Formal Coverage to quantify each of these three questions**


Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

# Constraints: Ensuring No Unintentional Over-constraints

- Review the list of constraints with the designer

- Validate absence of unintentional over-constraints:
  1. Instantiate of constraints as assertions in simulation
  2. Use cross-proof with neighboring blocks
  3. Use of formal coverage


Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2016

# Complexity: Reaching the Required Proof Depth (RPD)

- Use the 6-step methodology to derive the RPD*
- Use formal coverage to quantify RPD
- All End-to-End checkers need to reach the RPD

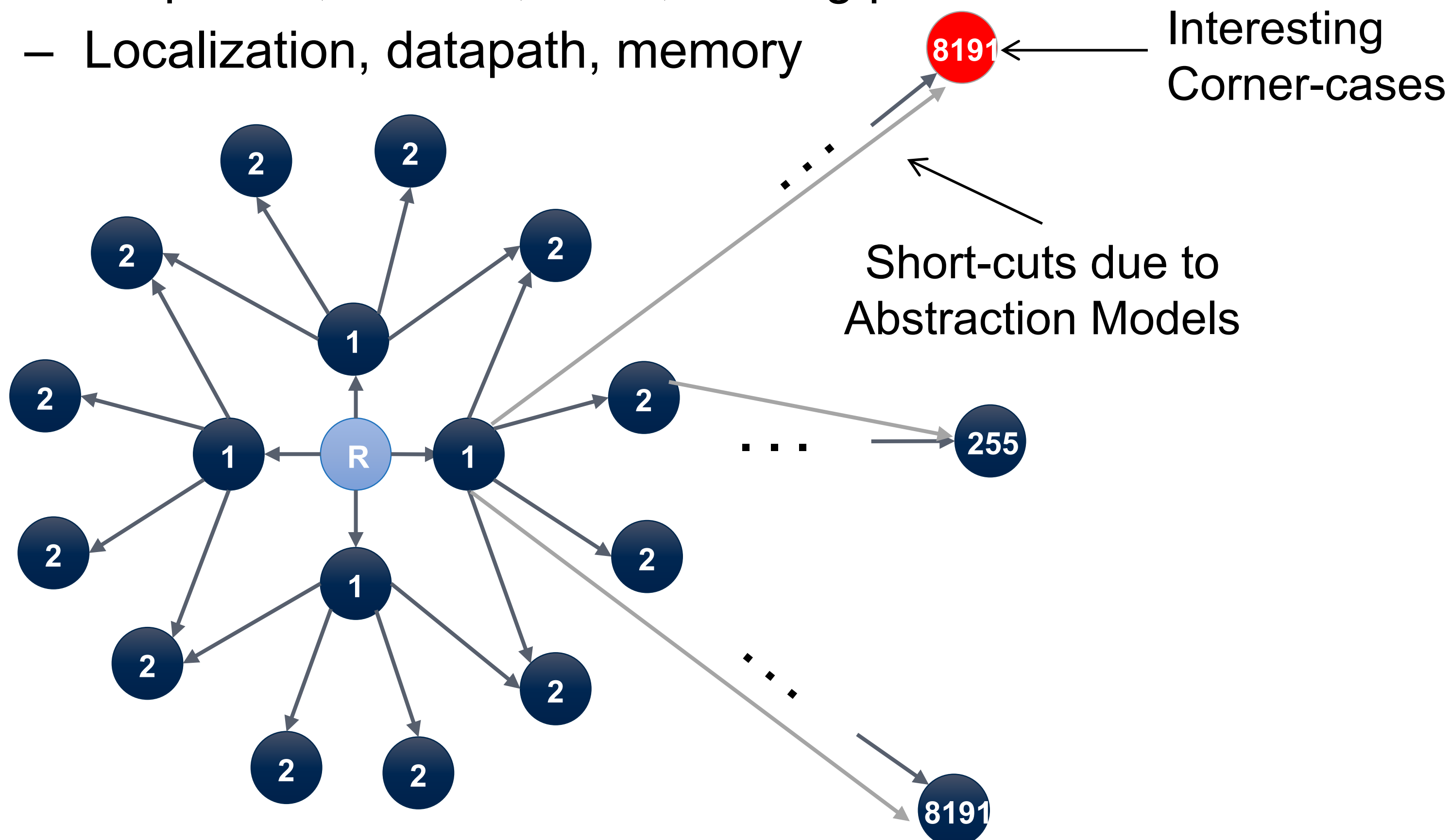
*Kim, N., et al. "Sign-off with Bounded Formal Verification Proofs," in DVCon 2014


Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

# Complexity: Using Creative Techniques to Reduce RPD

Resolving complexity challenges using Abstraction Models, or other techniques

- Sequence, counter, reset, floating pulse ...
- Localization, datapath, memory





Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

# Checkers: Ensuring Completeness of End-to-End Checkers

- Review the list of checkers with the designer
  - Ensure each output has an End-to-End checker, unless the designer determines the output does not need one
    - For example no need to verify profiling signals or test signals that are not related to design functionality
- Use negative testing (design mutation)
  - Randomly/Intelligently insert design bugs manually to make sure they are caught by existing checkers
  - Verify that every bug found by simulation is also found by existing checkers
- Use formal coverage (proof core) to ensure 100% code coverage


Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

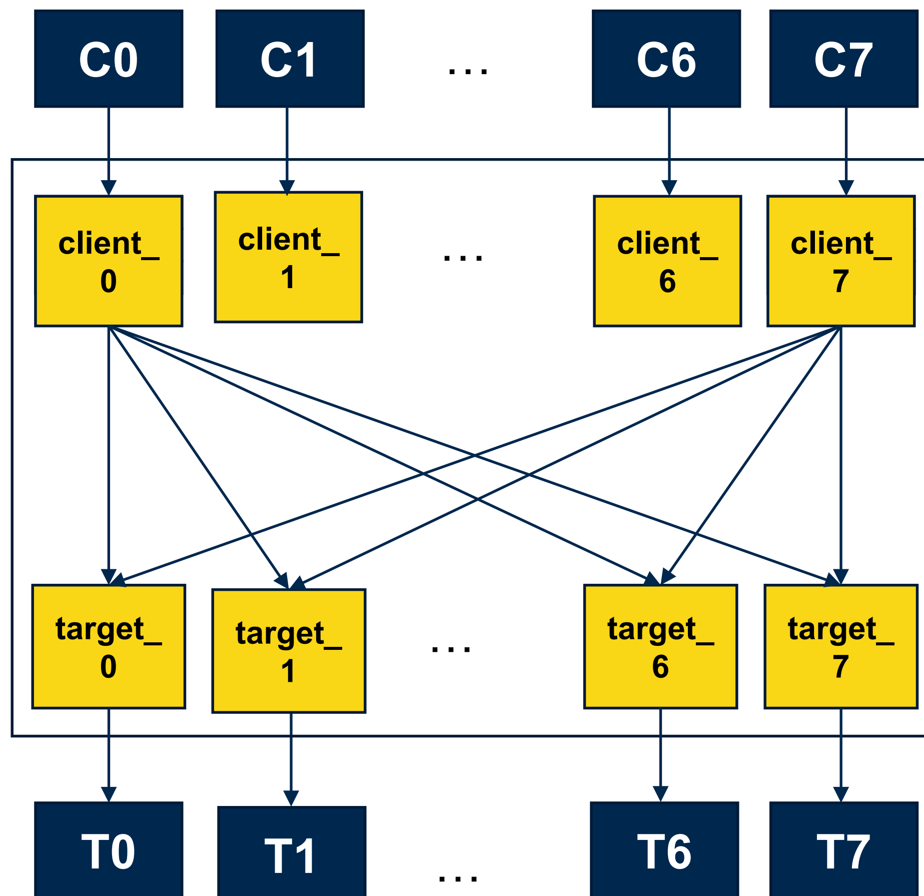
# Live Case Study – “Break the Testbench” Challenge at DAC 2015


Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2016

# Multicast Crossbar Design Specifications

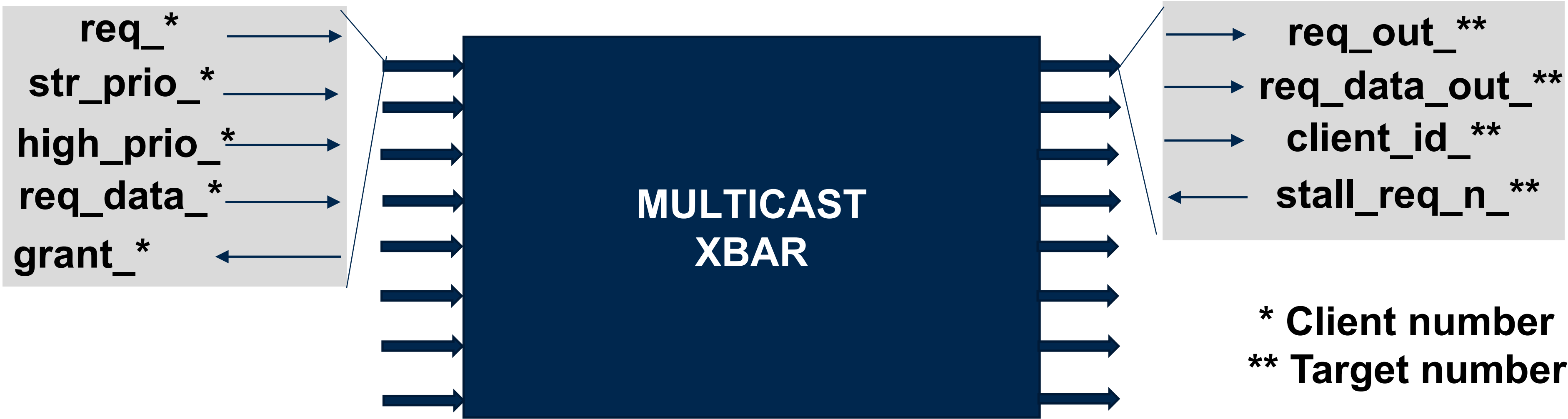
- A client can send request along with data to any target
- A client request can go to multiple targets (multicast)
- Each target has an arbiter that determines which client's request gets forwarded





Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

# Design Stats




| Design | |
|---|---|
| Inputs | 40 |
| Outputs | 32 |
| Flops | 312 |
| Lines of RTL code | 1,229 |
| Files | 6 |


Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION 2016 UNITED STATES

# English List of End-to-End Checkers

- Arbitration checker on req_out and client_id
  1. Among multiple requests, a request with highest priority type (strict, high, normal) should be seen at output
  2. Among multiple requests of the same priority, a given client should not get a grant twice before the other client has been given a grant
- Consistency checker on req_out and client_id
  - Any output from a target should have had an associated client request (i.e. no spurious outputs are seen)
- Consistency checker on grant
  - Any grant at a client should have had an associated client request (i.e. no spurious grants are seen)
- **Consistency checker on req_data_out**
  - Data seen at target output should be consistent with data seen at client input (i.e. data should not be corrupted, duplicated, reordered or dropped)
- Forward progress checker on req_out
  - A client request should be seen at some target output within finite time
- Forward progress checker on grant
  - A client request should get a grant within finite time


Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

# English List of Design Constraints

- If strict or high priority is asserted by a client, request must be asserted by the client
- Once request is asserted by a client; request, strict priority, high priority and data must be held stable until grant is asserted
- Only one client will send a strict priority request at a time


Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

# Using Symbolic Variables in Formal Testbench

```
##1(sym_client ==
    $past(sym_client))
```

**Select symbolic random client**

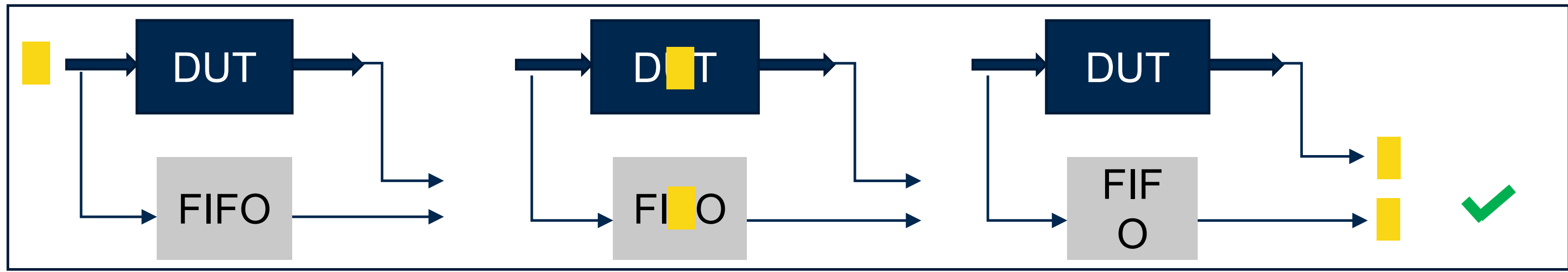
```
##1(sym_target ==
    $past(sym_target))
```

**Select symbolic random target**

- 8 clients and 8 targets, each client can send a request to multiple targets
  - Large number of combinations need to be tracked to fully verify the design
- Select symbolic random client and symbolic random target
  - Track all requests from symbolic client to symbolic target
  - All possible combinations exercised by formal tool in single run

 Prashant Aggarwal, Oski Technology, Inc. All rights reserved. 

# Data Consistency Checking Using FIFO-Based Scheme




How FIFO based scheme works?

**Data Drop**

DUT

X

FIFO

**Data Duplication**

DUT

X

FIFO

**Data Reorder**

DUT

X

FIFO

**Data Corruption**

DUT

X

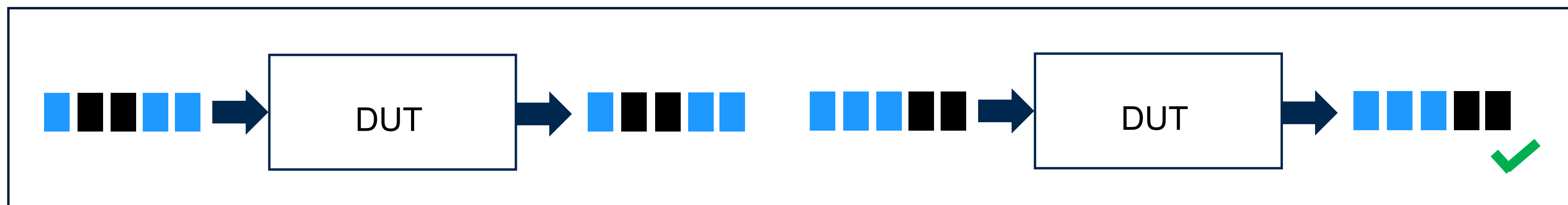
FIFO

**Random data**

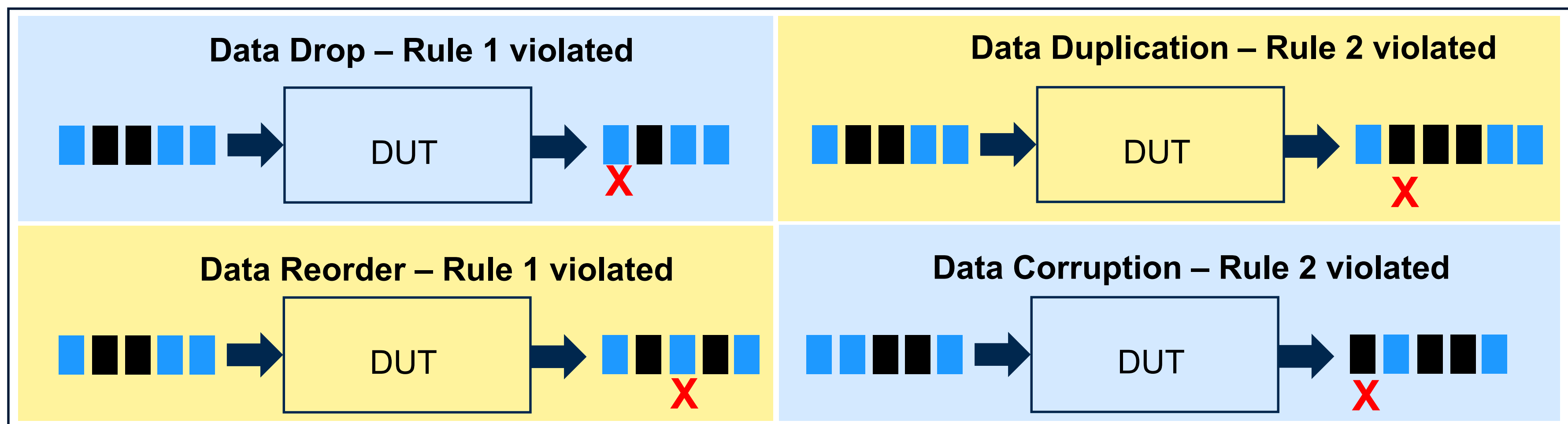

Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

# Data Consistency Checking Using Wolper Coloring Technique

| Wolper Coloring Technique Rules | | |
|---|---|---|
| $0^*110^\omega$ | If first 1 is seen, next input/output should be 1 | If two 1's have been seen, only 0's should be seen |




How Wolper coloring technique ($0^*110^\omega$) works?




0 1


Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

# Constraints: Ensuring No Unintentional Over-constraints

- Reviewed list of constraints with the designer
- Validated using formal coverage (no over-constraints)
  - Line Coverage
    - Total lines: 288
    - Covered lines: 288 (100%)
  - Condition Coverage
    - Total lines: 88
    - Covered lines: 88 (100%)

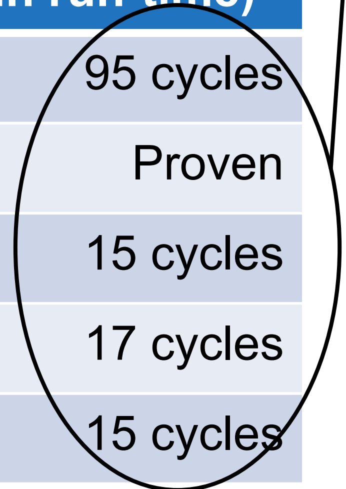

Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

# Complexity: Reaching the Required Proof Depth (RPD)

- Deepest cover point is 5 cycles deep
- Required Proof Depth is 13 cycles
  - Found using 6-step process
  - Not optimistic!

**No need to use complexity reduction techniques as Bound Achieved > RPD**

| Checker | Bound Achieved (in 10min run-time) |
|---|---|
| Arbitration checker on req_out and client_id | 95 cycles |
| Consistency checker on req_out and client_id | Proven |
| Forward progress checker on req_out | 15 cycles |
| Forward progress checker on grant | 17 cycles |
| Consistency checker on req_data_out | 15 cycles |




Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

# Checkers: Ensuring Completeness of End-to-End Checkers

- List of checkers reviewed with the designer

- 73 artificial functional bugs were manually inserted in the DAC 2015 Challenge
  – Checkers found all of them!


Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

# Example Bugs Inserted During the Challenge

| Bug Category | Original RTL | Buggy RTL | #Failing checkers |
|---|---|---|---|
| Arbitration Scheme | `// arbiter.sv`<br>`60 assign high_prio_req = (\|str_prio_req) ? {NUM_CLIENT{1'b0}} : (high_prio & req);` | `// arbiter.sv`<br>`60 assign high_prio_req = (\|(str_prio_req \| high_prio_req)) ? {NUM_CLIENT{1'b0}} : req;` | 4 |
| Arbitration Scheme | `// rr_scheme.sv`<br>`56 assign shft_req = {req, req};` | `// rr_scheme.sv`<br>`56 assign shft_req = {8'd0, req};` | 2 |
| Arbitration Scheme | `// rr_scheme.sv`<br>`65 for(i = 0; i < (2 * NUM_CLIENT); i = i + 1) begin` | `// rr_scheme.sv`<br>`66 for(i = 0; i < (2 * NUM_CLIENT - 1); i = i + 1) begin` | 2 |
| Connectivity | `// xbar_8x8.sv`<br>`274 .high_prio(high_prio_4),` | `// xbar_8x8.sv`<br>`274 .high_prio(high_prio_3),` | 4 |
| Grant generation | `// one_dly.sv`<br>`71 assign gnt_i = has_data ? outgoing_data : 1'b1;` | `// one_dly.sv`<br>`71 assign gnt_i = has_data ? outgoing_data : 1'b0;` | 2 |
| Wrong operator | `// target.sv`<br>`89 assign t2c_grant = {NUM_CLIENT{ext_grant_pp}} & arb_gnt_pp;` | `// target.sv`<br>`89 assign t2c_grant = {NUM_CLIENT{ext_grant_pp}} && arb_gnt_pp;` | 8 |


Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

# Non-bug #1: Round Robin Arbiter Initial Value Change

```
Original RTL:
for(i = 0; i < (2 * NUM_CLIENT); i = i + 1)

Modified RTL:
for(i = 1; i < (2 * NUM_CLIENT); i = i + 1)
```

**`i=1`** makes an RTL optimization to use an n-iteration loop instead of an n+1-iteration loop

- Makes the design slightly better, area-wise
- No functional impact


Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

# Non-bug #2: Grant is 0 After Counter Reaches Threshold

```
Original RTL:
assign high_prio_gnt = tmp_high_prio_gnt;
Modified RTL:
assign high_prio_gnt[0] = tmp_high_prio_gnt[0] &
                          ~(my_bug_delay ==
                            1024'12345678910121З);
always @ (posedge clk) begin
     if (rst) my_bug_delay <= 1024'b0;
     else my_bug_delay <= my_bug_delay + 1'b1;
end
```

- Bug was inserted with a "malicious intent"
  - Used knowledge of the verification methodology to specifically change the design, such that the defect cannot be caught by the test-bench
  - Defeats the purpose of "true" verification i.e. find all "naturally occurring" bugs
- Increased the Required Proof Depth, making bug impossible for formal (and sometimes simulation) to find, without using Abstraction Models


Prashant Aggarwal, Oski Technology, Inc. All rights reserved.

# Summary

- Significant design blocks in SoC, processor and networking chips can be verified with formal

- Formal sign-off offers ultimate confidence in verification - No bug left behind

- Formal sign-off can be achieved by ensuring
  1. No unintentional over-constraints
  2. List of checkers is complete
  3. All checkers reach Required Proof Depth


Prashant Aggarwal, Oski Technology, Inc. All rights reserved.