2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION

UNITED STATES

SAN JOSE, CA, USA

FEBRUARY 28 - MARCH 3, 2022

Raising the level of Formal Signoff with End-to-End Checking Methodology

Ping Yeung, Arun Khurana, Dhruv Gupta,
Ashutosh Prasad, Achin Mittal

Oski Technology

San Jose, CA, Gurugram India

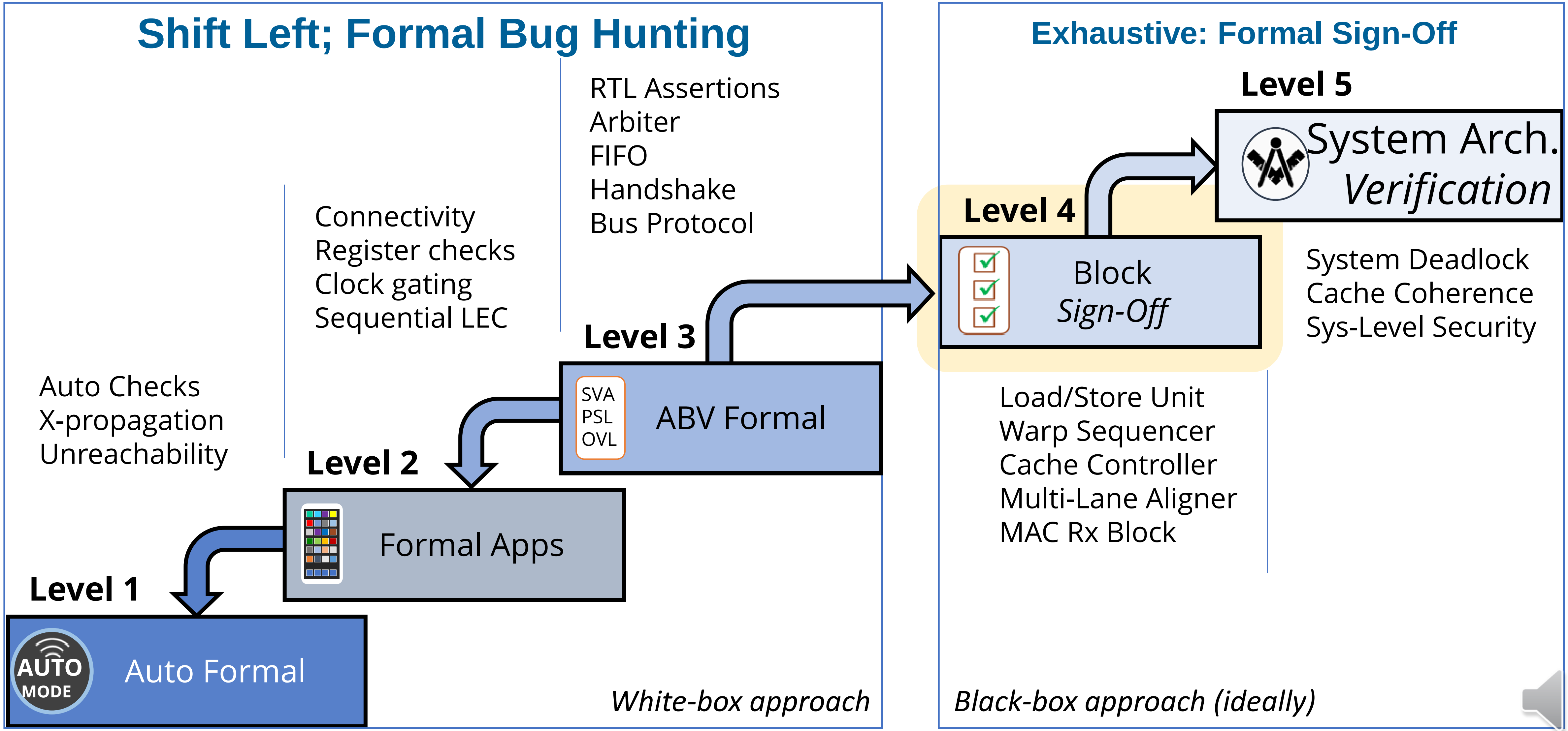
Oski TECHNOLOGY

accellera SYSTEMS INITIATIVE

Agenda

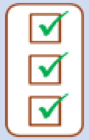
- Formal Verification Usage Levels
- End-to-End Checking Methodology
- End-to-End Checkers
- Abstraction Techniques and Modeling
- Testcases
  - Parameterized Multi-cast Crossbar Design
  - GPU Level 2 Cache Request Coalescer (LRC) unit
  - NOC Configurable Cache Controller

Formal Verification Usage Levels



Block-Level Formal Signoff

Level 4



Block
Sign-Off

Different from traditional Assertion-based Verification

- Black-box approach; use end-to-end checkers; does not depend on RTL
- Divide-and-conquer with multiple formal testbenches

Block-Level Formal Signoff

Level 4

Block
Sign-Off

Different from traditional Assertion-based Verification

- Black-box approach; use end-to-end checkers; does not depend on RTL
- Divide-and-conquer with multiple formal testbenches

Early deployment

- Identify incomplete or ambiguous specifications early in the design cycle,
- Provide clear value to the project team because they map directly to the functional specification
- Find bugs and verify the block while the designer is coding the RTL

Block-Level Formal Signoff

Level 4

Block
Sign-Off

Different from traditional Assertion-based Verification

- Black-box approach; use end-to-end checkers; does not depend on RTL
- Divide-and-conquer with multiple formal testbenches

Early deployment

- Identify incomplete or ambiguous specifications early in the design cycle,
- Provide clear value to the project team because they map directly to the functional specification
- Find bugs and verify the block while the designer is coding the RTL

Exhaustiveness

- Replace simulation entirely and do a formal signoff of the block,
- Find deep or unaware corner case issues

Reusability

- Use to confirm RTL fixes; ensure all scenarios are covered
- Reuse the formal testbench to verify new RTL code

Agenda

- Formal Verification Usage Levels
- End-to-End Checking Methodology
- End-to-End Checkers
- Abstraction Techniques and Modeling
- Testcases
  - Parameterized Multi-cast Crossbar Design
  - GPU Level 2 Cache Request Coalescer (LRC) unit
  - NOC Configurable Cache Controller

End-to-End Checking Methodology

Task	Planning	Implementation	Closure
Management	Formal expertise Schedule & milestones	Allocate formal engineer resources	Plan extra compute, vendor resources

Management

- Need a team of formal experts and engineers
  - Formal experts with years of experience required for formal planning
  - Formal engineers required for formal testbench implementation
  - Careful partnering of formal engineers with design team members
- Need compute resources and vendor expertise
  - Server farm environment for formal coverage and final signoff
  - Vendor expertise to address some difficult properties

End-to-End Checking Methodology

Task	Planning
Management	Formal expertise Schedule & milestones
Block	Identify and Evaluate
Function	Describe and Prioritize
Complexity	Decompose and Map

Block

- Identify blocks for E2E formal
- Evaluate to determine effort

Function

- Describe E2E functionality
- Prioritize them based on importance/risk

Complexity

- Decompose, divide-and-conquer
- Map them to one or more formal TBs

End-to-End Checking Methodology

Task	Planning	Implementation
Management	Formal expertise Schedule & milestones	Allocate formal engineer resources
Block	Identify and Evaluate	Capture Interfaces
Function	Describe and Prioritize	End-to-End Checkers
Complexity	Decompose and Map	Abstraction Techniques

End-to-End Checking Methodology

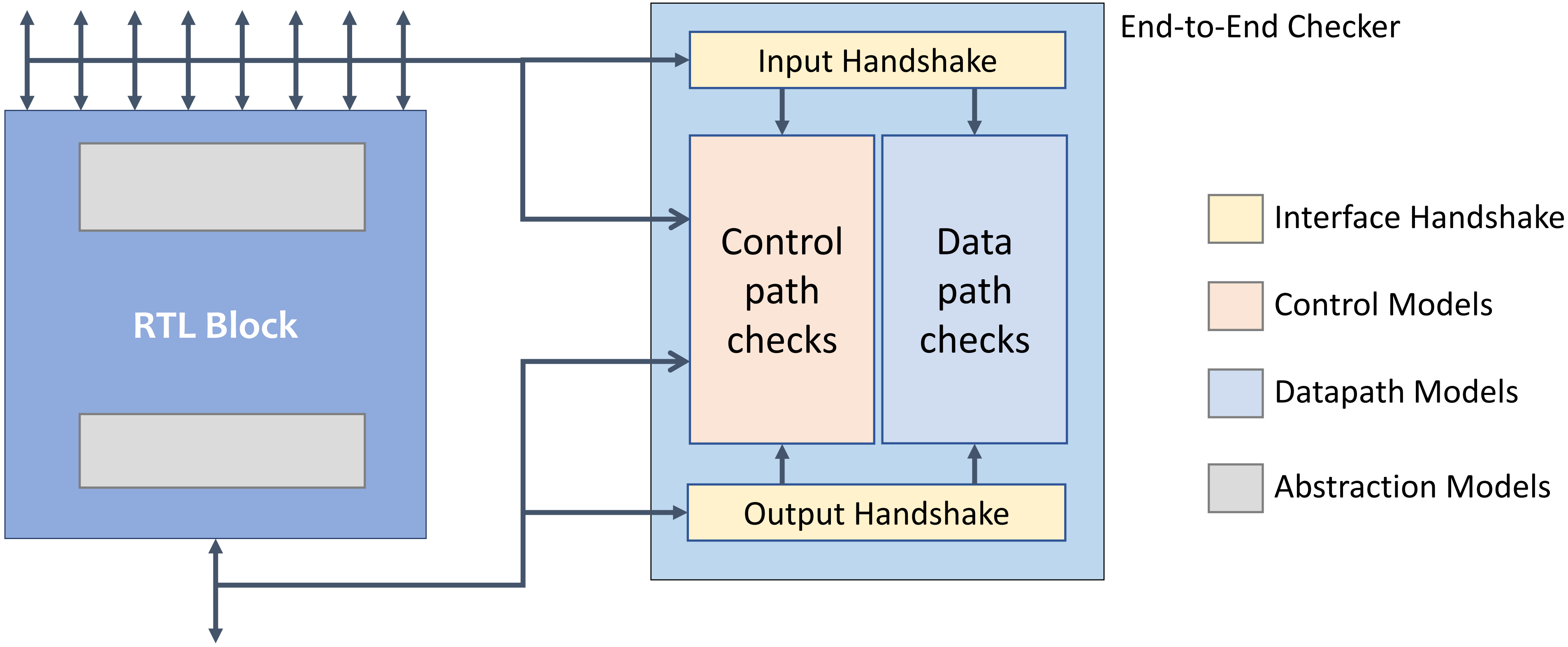
Task	Planning	Implementation	Closure
Management	Formal expertise Schedule & milestones	Allocate formal engineer resources	Plan extra compute, vendor resources
Block	Identify and Evaluate	Capture Interfaces	Validate Constraints
Function	Describe and Prioritize	End-to-End Checkers	Conclusiveness
Complexity	Decompose and Map	Abstraction Techniques	Formal Coverage

Agenda

- Formal Verification Usage Levels
- End-to-End Checking Methodology
- End-to-End Checkers
- Abstraction Techniques and Modeling
- Testcases
  - Parameterized Multi-cast Crossbar Design
  - GPU Level 2 Cache Request Coalescer (LRC) unit
  - NOC Configurable Cache Controller

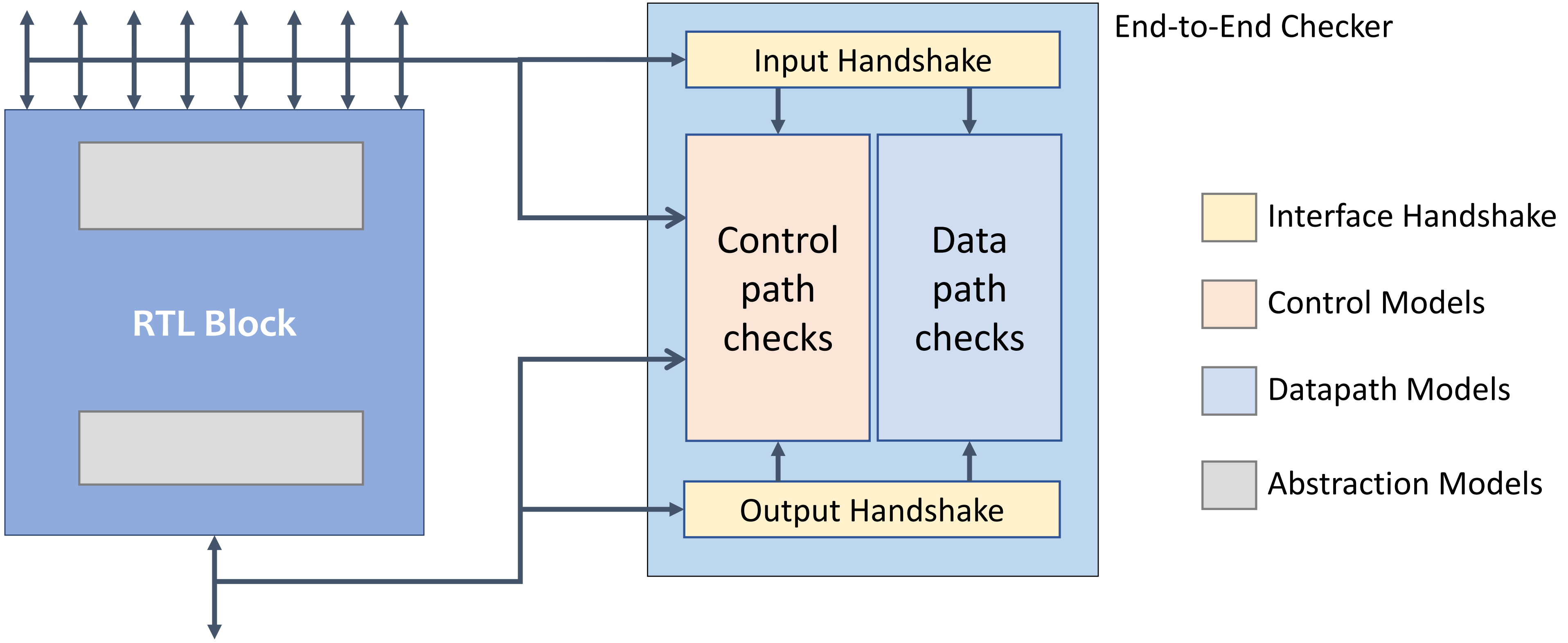
End-to-End Checkers

Developing formal-friendly reference model could be as big an effort as writing RTL



End-to-End Checkers

Developing formal-friendly reference model could be as big an effort as writing RTL



Abstraction Techniques

Abstraction Technique	Design Complexity	Formal Efficiency
Case splitting	Multiple runs with different cases reducing design complexity per run/case	Reduce COI, reduce state space per run/case
Cut-point/ Black box	Eliminate logic driving cut-points/inside blackbox	Reduce COI, state space; controlled with constraints

Abstraction Techniques

Abstraction Technique	Design Complexity	Formal Efficiency
Case splitting	Multiple runs with different cases reducing design complexity per run/case	Reduce COI, reduce state space per run/case
Cut-point/ Black box	Eliminate logic driving cut-points/inside blackbox	Increase flexibility but controlled with constraints
Reset abstraction	n.a.	Reduce access depth
Counter abstraction	n.a.	Reduce the length of counting

Abstraction Modeling 1

Abstraction Model	Design Complexity	Formal Efficiency
Symmetric data elements [7]	Eliminate multiple dimensional data elements; add single dimension abstraction model	Reduce COI and state space with symmetry

Abstraction Modeling 1

Abstraction Model	Design Complexity	Formal Efficiency
Symmetric data elements [7]	Eliminate multiple dimensional data elements; add single dimension abstraction model	Reduce COI and state space with symmetry

RTL model	Abstraction model

element_type [SIZE-1:0] element;
 	
element [addr] = wr_data;
rd_data = element [addr];
	element_type abs_element;
	
	if (addr == sym_addr) abs_element = wr_data;
	if (addr == sym_addr) rd_data = abs_element;

$stable (sym_addr)

Abstraction Modeling 2

Abstraction Model	Design Complexity	Formal Efficiency
Memory abstraction [7]	Represent one location instead of the full size of the memory	Reduce COI and state space with symmetry

```
RTL memory:            reg [WIDTH-1:0] mem [DEPTH-1:0];
abstraction memory:    reg [WIDTH-1:0] mem;
assume property:       (sym_addr < DEPTH) ##1 $stable(sym_addr)
abstraction write:     if (wr && (wr_addr == sym_addr)) mem <= wr_data;
abstraction read:      if (rd && (rd_addr == sym_addr)) rd_data = mem;
```

Abstraction Modeling 3

Abstraction Model	Design Complexity	Formal Efficiency
FIFO [7]	Eliminate logic before cut-points; add abstraction model	Reduce the depth of the FIFO

```
wire [LOG_DEPTH-1:0]     sym_depth;
assume property:         (sym_depth > 1 && sym_depth < DEPTH) ##1 $stable(sym_depth)
abstraction model:       if (wr_ptr == sym_depth) wr_ptr <= 0;
                         else wr_ptr <= wr_ptr + 1;
```

Abstraction Modeling 4

Abstraction Model	Design Complexity	Formal Efficiency
Data independence (Wolper Coloring) [6]	Eliminate all storage elements; add Wolper FSMs	Reduce COI with pattern

The rules for generating and verifying the Wolper sequence are:

1. If the first 1 is seen, next one should be 1

   wolper_1st_1_seen_next_1: (first_one && !second_one && input_valid) |-> (colored_input == 1'b1)

2. If two 1's are seen, only 0's should be seen

   wolper_2nd_1_seen_forever_0: (second_one && input_valid) |-> (colored_input == 1'b0)

Abstraction Modeling Summary

Abstraction Modeling	Design Complexity	Formal Efficiency
Symmetric data elements [7]	Eliminate multiple dimensional data elements; add single dimension abstraction model	Reduce COI and state space with symmetry
Memory abstraction [7]	Represent one location instead of the full size of the memory	Reduce COI and state space with symmetry
FIFO [7]	Eliminate logic before cut-points; add abstraction model	Reduce the depth of the FIFO
Data independence (Wolper Coloring) [6]	Eliminate all storage elements; add Wolper FSMs	Reduce COI with pattern
Tagging [9]	Represent one tag instead of the complete linked list	Reduce COI

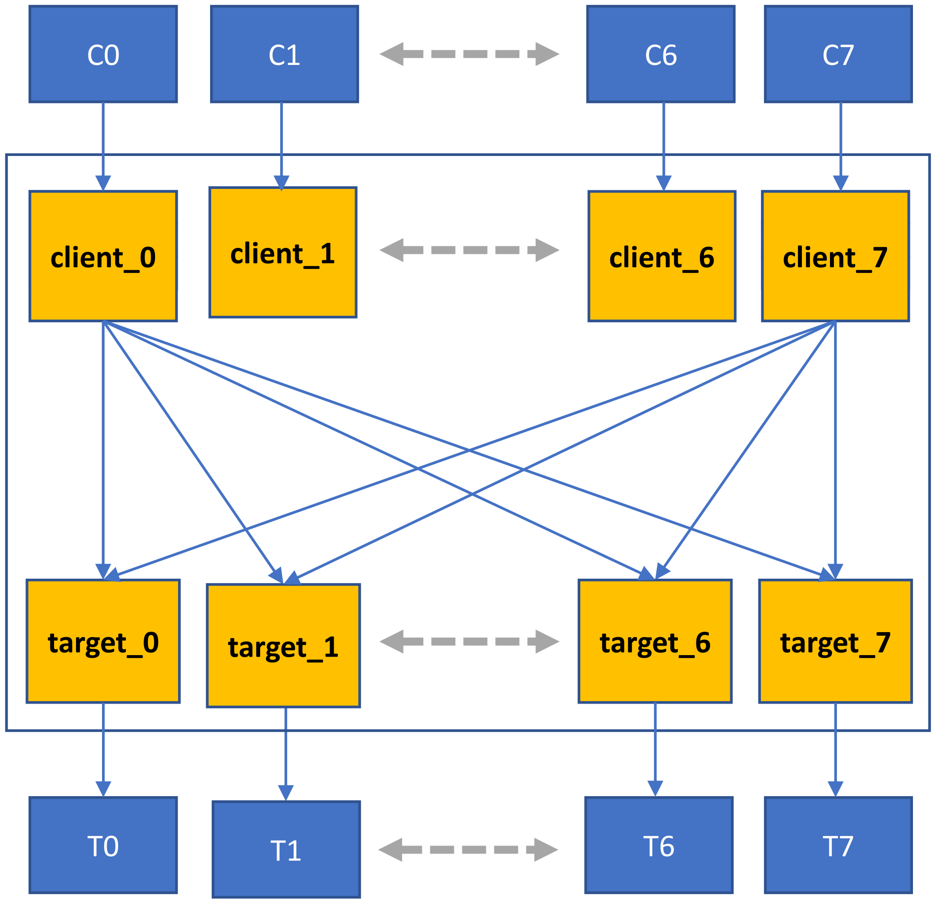
Agenda

- Formal Verification Usage Levels
- End-to-End Checking Methodology
- End-to-End Checkers
- Abstraction Techniques and Modeling
- Testcases
  - Parameterized Multi-cast Crossbar Design
  - GPU Level 2 Cache Request Coalescer (LRC) unit
  - NOC Configurable Cache Controller

Parameterized Multi-cast Crossbar Design

- 8x8 Crossbar design
  - each client can send request to 1+ targets
  - Each target has an arbiter to decide which request gets forwarded based on priorities

- Abstraction Deployed
  - symbolic variables used to select a client/target and implemented all of the checkers for the symbolic client and target pair.
  - Formal explore all possible values for the symbolic variables



8x8 Multicast Crossbar

Control Path and Data Path Checkers

Multi-cast Crossbar Design:

- Control path end-to-end checkers:
  - An arbitration checker (a combination of two checkers) for the arbitration scheme
  - A consistency checker to ensure no spurious grant is given to a client
  - Performance checkers to ensure operations are performed in each cycle when the conditions are met.

Control Path and Data Path Checkers

Multi-cast Crossbar Design:

- Control path end-to-end checkers:
  - An arbitration checker (a combination of two checkers) for the arbitration scheme
  - A consistency checker to ensure no spurious grant is given to a client
  - Performance checkers to ensure operations are performed in each cycle when the conditions are met.
- Data path end-to-end checkers:
  - Data integrity checkers to ensure correct transfer
    - from read data input port to buffer
    - from buffer to store output port.
    - data is not corrupted, duplicated, reordered, or dropped.
  - Wolper coloring technique: doesn't require data storage

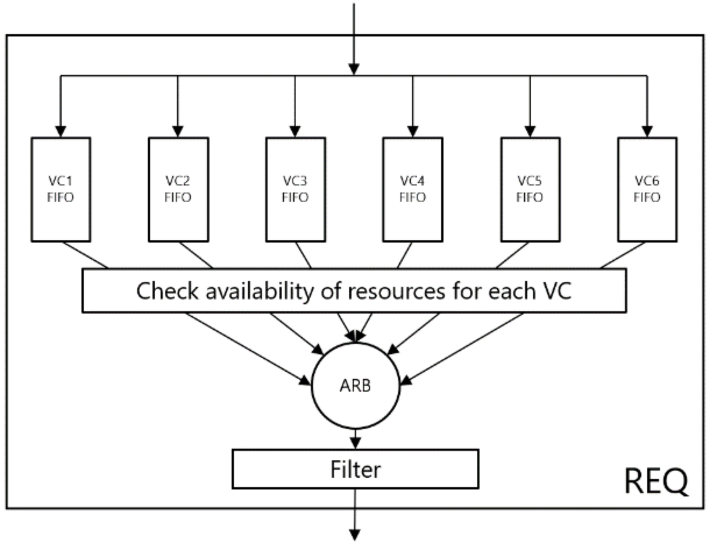
Parameterized Multi-cast Crossbar Design

Task	Planning	Implementation	Closure
Management	Formal expert (6+ yr) (20% time)	Formal engineer (2+ yr) Schedule: 1.5 months	8-core, 48GB memory server
Block	Divide and conquer: n.a.	Capture Interfaces: Client inputs/outputs Target inputs/outputs	Validate Constraints: Simulation integrated
Function	Prioritize: Data correctness Arbitration workload Sequence of data flow	End-to-End Checkers: Data integrity (Wolper) Target arbitration Forward progress checkers	RTL Bugs: 73 known bugs found
Complexity	Decompose: n.a.	Abstraction Techniques: Use symmetric elements; symbolic variable on client and target pair	Formal Coverage: Line: 100% Condition: 100%

Ipshita Tripathi, Ankit Saxdna, et al., "Process & Proof for Formal Signoff - Live Case Study," DVCon 2016

GPU Level 2 Cache Request Coalescer (LRC) unit

- Risk of top-level deadlock bugs
  - Top-level simulation coverage is insufficient
  - Blocks with embedded stall conditions introduce dependencies
- Developed a novel approach for deadlock detection
  - Proved the absence of deadlock across multiple virtual channels in the L2 Request Coalescer
- Repeatable method to detect deadlocks in complex designs



Saurabh Chaurdia, Oski Technology
Arun Khurana, Oski Technology
Naveen Kumar, Oski Technology
Aditya Chaurasiya, Oski Technology
Yogesh Mahajan, NVIDIA
Prasenjit Biswas, NVIDIA

GPU Level 2 Cache Request Coalescer (LRC) unit

Task	Planning
Management	Formal expert (9+ yr) (20% time)
Block	Divide and conquer: Submodules: Req, Rsp
Function	Prioritize: All IP block, all checks are important
Complexity	Decompose: ILC (submodule) blackbox Design Shrinking (FIFO and CAM) Partition VC path to reduce latency

GPU Level 2 Cache Request Coalescer (LRC) unit

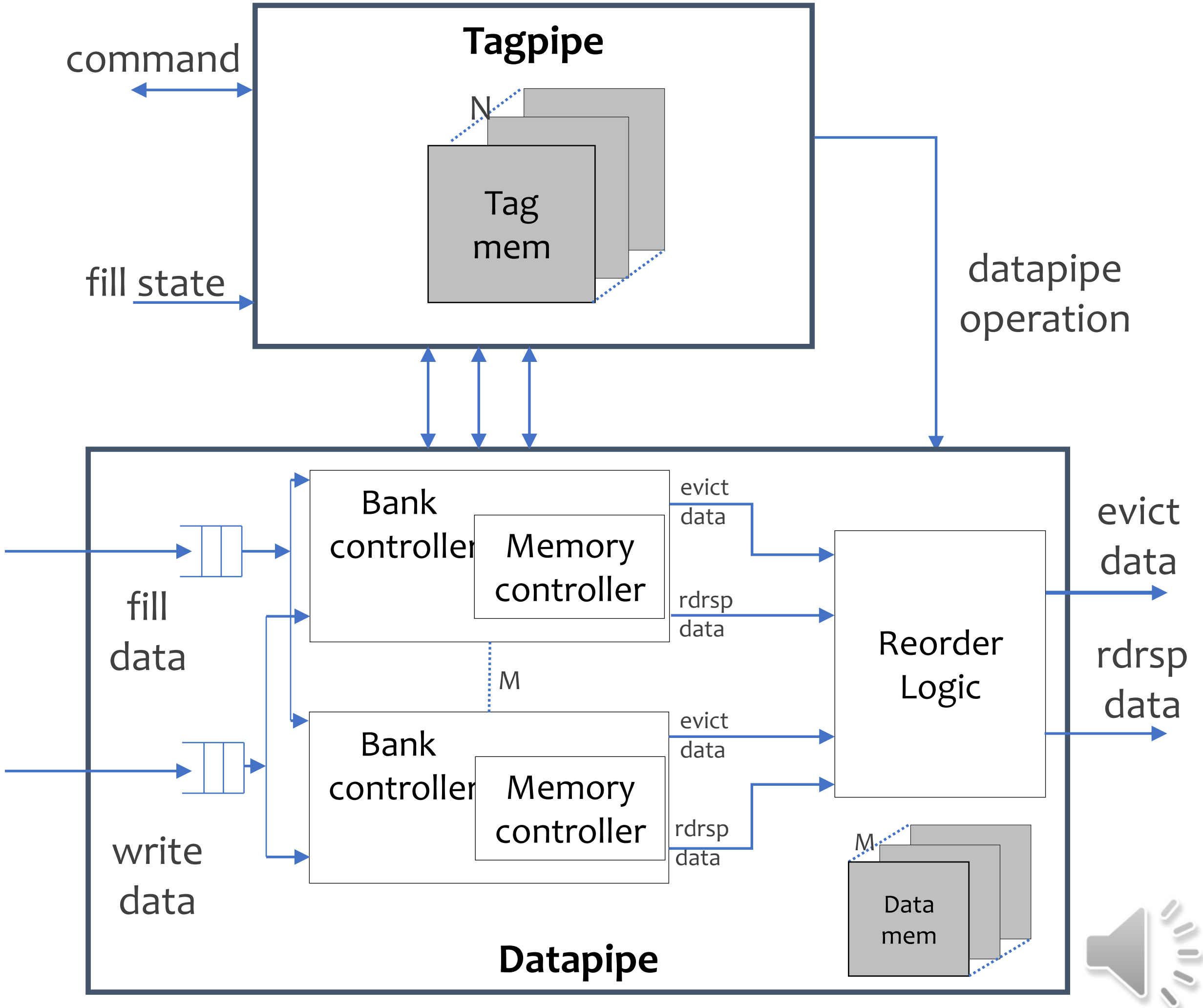
Task	Planning	Implementation
Management	Formal expert (9+ yr) (20% time)	Formal engineer (1+ yr) Schedule: 6.5 months
Block	Divide and conquer: Submodules: Req, Rsp	Capture Interfaces: Xbar Interface L2 interface
Function	Prioritize: All IP block, all checks are important	End-to-End Checkers: Request coalescing Data integrity Response replay Forward progress
Complexity	Decompose: ILC (submodule) blackbox Design Shrinking (FIFO and CAM) Partition VC path to reduce latency	Abstraction Techniques: Counter abstraction Wolper's method for data consistency Symbolic address/CAM ID modeling

GPU Level 2 Cache Request Coalescer (LRC) unit

Task	Planning	Implementation	Closure
Management	Formal expert (9+ yr) (20% time)	Formal engineer (1+ yr) Schedule: 6.5 months	16-core, 256GB memory server
Block	Divide and conquer: Submodules: Req, Rsp	Capture Interfaces: Xbar Interface L2 interface	Validate Constraints: Simulation integrated; cross-proved
Function	Prioritize: All IP block, all checks are important	End-to-End Checkers: Request coalescing Data integrity Response replay Forward progress	RTL Bugs: 57 bugs found 7 corner-case issues
Complexity	Decompose: ILC (submodule) blackbox Design Shrinking (FIFO and CAM) Partition VC path to reduce latency	Abstraction Techniques: Counter abstraction Wolper's method for data consistency Symbolic address/CAM ID modeling	Formal Coverage: Line: 100% Condition: 100%

NOC Configurable Cache Controller

- Simulation-only unable to deliver required level of confidence for IP products
  - Too many configurations to test
  - Cannot afford failures of untested scenarios that render chip unusable
- Deployed formal sign-off methodology
  - 70+ bugs found
  - >40% of bugs considered simulation-resistant
- Confident that the last bug was found



NOC Configurable Cache Controller

Task	Planning	Implementation	Closure
Management	Formal expert (10+ yr, 25% time) Schedule: 5-6 months	Sr. Formal engineer (50% time) 2x Formal engineer (2+ yr)	16-core, 64GB server 16-core, 512GB server
Block	Divide and conquer: Submodules: arbiters, cacheline controller, DDR controller		
Function	Prioritize: All LRU arbiter (module) Cacheline (SV bind) Tag flow path (SV bind) Data flow path (SV bind) 4x interfaces (SV bind)		
Complexity	Decompose: Tag and Data flow paths were decomposed		

NOC Configurable Cache Controller

Task	Planning	Implementation	Closure
Management	Formal expert (10+ yr, 25% time) Schedule: 5-6 months	Sr. Formal engineer (50% time) 2x Formal engineer (2+ yr)	16-core, 64GB server 16-core, 512GB server
Block	Divide and conquer: Submodules: arbiters, cacheline controller, DDR controller	Capture Interfaces: Cmd and Register interfaces Data SRAM interface DDR RAM interface Tag <> data interface	Validate Constraints: Simulation integrated; cross-proved
Function	Prioritize: All LRU arbiter (module) Cacheline (SV bind) Tag flow path (SV bind) Data flow path (SV bind) 4x interfaces (SV bind)	End-to-End Checkers: Tag flow: - Tag state, Eviction address/state - Replacement policy Data flow: - Write/read data integrity - Eviction data	Total 496 properties 76% proven 24% bounded 76 bugs 29 bugs are simulation resistant
Complexity	Decompose: Tag and Data flow paths were decomposed	Abstraction Techniques: Reset abstractions Cut-points Symbolic sets for symmetric data in tag and data memories Data coloring for data consistency	Formal Coverage: Functional coverage Assertion precondition coverage Checkers reach required proof depth

End-to-End Checking Methodology

Task	Planning	Implementation	Closure
Management	Formal expertise Schedule & milestones	Allocate formal engineer resources	Plan extra compute, vendor resources
Block	Identify and Evaluate	Capture Interfaces	Validate Constraints
Function	Describe and Prioritize	End-to-End Checkers	Conclusiveness (inconclusives)
Complexity	Decompose and Map	Abstraction Techniques	Formal Coverage

Summary

- Block-level Formal Signoff with End-to-End Checking Methodology
  - End-to-End Checkers
  - Abstraction Techniques and Modeling
  - Comprehensive for block-level formal signoff

- Major benefits
  - **Reduce time to First Bug:** Shift-Left "Avoidable Bugs"
  - **Reduce time to Last Bug:** Eliminate "Inevitable Bugs"

- Acknowledgement
  - The support of the whole Oski Team in Gurugram, India.